Syntax and Semantics in Minimalist Grammars

Gregory M. Kobele

ESSLLI '09

This chapter presents a grammar for a fragment of English A-movement.[1] The constructions accounted for include raising, passivization, and control. Section 1 motivates our basic syntactic theory, culminating in an account of raising and passivization, along with a novel account of expletive *it*, which immediately explains the ban on superraising in terms of independently motivated notions of intervention and case assignment. Section 2 presents a directly compositional semantics for our syntactic theory, one which maps derivations incrementally to model-theoretic objects, making no reference to syntactic trees. We extend our fragment with quantifiers, and show how we can derive quantifier scope ambiguities without a seperate operation of quantifier raising (QR). The tense clause boundedness of QR is easily shown to be simply and directly replicable in our system in terms of independently motivated assumptions about feature checking. We extend our fragment once more to account for control, which, given our semantic operations, is naturally implemented in terms of movement to theta positions. This treatment of control in our system derives semantically the impossibility of scope reconstruction into a control complement, and syntactically something like the minimal distance principle (MDP), both of which purely on the basis of our independently motivated treatment of quantifiers and scope taking on the one hand and locality conditions on movement on the other. While other movement approaches to control have been claimed both to make erroneous predictions about the interaction of control predicates with passivization, as well as to be unable to account for the grammatical (although apparently typologically marked) status of MDP-violating *promise*-type control verbs, we show that our system handles both in a natural and simple way. Although we are able to account for the existence and behaviour of *promise*-type con-

[1]The present document is (a very slightly revised version of) chapter two of Kobele [48].

trol verbs without any additional stipulations (in fact, given the grammatical decisions which came before, there are no other analytical options), they are clearly and well-definedly more complex than *persuade*-type verbs, a fact on which it seems possible to begin to build an explanation of the acquisitional and typological markedness of the former with respect to the latter.

1 Introducing Minimalism

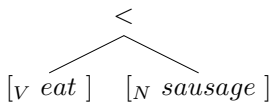
In this section we detail the particular version of minimalism that we will adopt in this dissertation. This is done in a quasi-socratic fashion, with a minimal theory being developed around basic argument structural facts, and then gradually revised and extended to deal with more, and diverse, data. Where applicable, we mention alternative possible extensions (such as feature percolation (as in HPSG) in lieu of movement), and motivate our theoretical decisions.

Our goal as linguists is to explain all aspects of natural language: its use, function, evolution, and acquisition. Instead of tackling all these difficult questions at once, linguists have attempted to break them down into more or less coherent subproblems, with the hope of eventually reconnecting the separated threads of investigation. Here, our goal is to explain all aspects of the syntax and semantics of English. As above, so below; we divide English into simplified fragments, and attempt to construct precise and complete accounts of these fragments, with the hope that a thorough understanding of these simplified fragments of English will suggest new ways to understand other aspects of other constructions. The fragments identified here abstract away from various complexities unearthed over fifty years of investigation into English and other languages. As such, there remain a great number of questions about how these myriad complexities should be treated, which will not be answered here. In particular, we abstract away from the particulars of case assignment, agreement and the like, treating feature checking as an unanalyzed operation. In other words, while we assume for simplicity that features are checked in a particular, lexically determined order, we make no assumptions about what feature checking consists in (whether unification, matching, etc). The treatment of raising and passivization presented herein bears obvious similarities to previous accounts. This is no surprise, as the novelty of this account lies not in the basic ideas about the nature of passivization or raising, but instead in their unification into a single simple yet complete account.

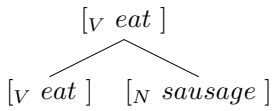
1.1 Syntactic Structures

We start with the intuitive picture that words (or something like them) are the building blocks of sentences. Linguistic expressions (non-atomic ones, at least) can be broken up into subexpressions, which can then be recombined to form other expressions. We take as a basic syntactic operation one that combines two expressions into a third. We will call this operation *merge*.

In the resultant expression, (some of) the properties of one of the arguments are inherited, to the exclusion of those of the other. This argument is said to be the *head* of the resultant expression. As an example, consider the expressions $[_V\ eat\ ]$ and $[_N\ sausage\ ]$ (where the subscripts on the brackets indicate the syntactic category of the expression), which merge to form the expression $[_V\ eat\ sausage\ ]$, which inherits the category of $[_V\ eat\ ]$, and not of $[_N\ sausage\ ]$. In this example, $[_V\ eat\ ]$ is the head of $[_V\ eat\ sausage\ ]$. Generative linguists standardly represent this information in terms of tree structure; the expression $[_V\ eat\ sausage\ ]$ is represented as the tree



where atomic expressions (words with their syntactic properties or *features*) are at the leaves, and the internal nodes are 'labels', indicating the head of the complex expression (here, labels are either $<$ or $>$, and indicate the head by 'pointing' toward it). Note that anything will do as a label, as long as it unambiguously indicates the head of the complex expression. Labels are sometimes taken to themselves be expressions [17], in which case the above tree would appear as



This move makes possible a 'set-theoretic' representation of the structure of linguistic expressions, where the above tree is replaced with the set theoretic object

$$\{\{[_V\ eat\ ]\}, \{[_V\ eat\ ], [_N\ sausage\ ]\}\}$$

Assuming that taking labels to be expressions allows one to unambiguously determine which of the two daughter expressions serve as the head of the complex one, it is easy to see that this set-theoretic representation is equivalent to a tree representation, albeit one which doesn't encode the relation of linear precedence between the daughters, at least not straightforwardly. However,

many influential proposals take there to be a functional relationship between hierarchical structure and linear order [45], and with this assumption these three representations are easily seen to be merely notational variants of each other (and thus it would be as strange to argue that one of them is privileged in some sense with respect to mental representation as it would be to argue that, in the context of the set theoretic notation, the set brackets are mentally represented as being curly ('{' or '}'), and not round ('(' or ')')).[2] We will adopt the first notation (with 'arrows' at internal nodes) in this section, as it is easier on the eye.

1.2 Features

Consider the following sentences.

(1) *John will devour.

(2) John will devour the ointment.

Something is amiss with sentence 1. Somehow, speakers of English know that when the verb *devour* occurs in a sentence, its semantic object (the thing devoured) needs to be expressed. This is in contrast to the nearly synonymous verb *eat*, whose semantic object needn't be overt in the sentences in which it appears.

[2]To show the equivalence of these notations we proceed as follows. Given a tree with labels pointing at the heads (without loss of generality we take the labels to be drawn from the set $\{<,>\}$), we first define a homomorphism f over trees, which simply 'forgets' the linear ordering of the terminals. Note that the resulting trees are still ordered, but by the 'inherits the properties of' (or the 'projects over') relation, not the standard linear precedence relation. If we take hierarchical structure to encode linear order as in [45], then f^{-1} is defined (and is a top-down delabeling, if we just want to ensure SPEC-HEAD-COMP order). In the following, 'ℓ' is a variable over leaves:

$$\begin{aligned} f([_{<} x\ y]) &= [f(x)\ f(y)] \\ f([_{>} x\ y]) &= [f(y)\ f(x)] \\ f(\ell) &= \ell \end{aligned}$$

We then define a bijection $\cdot'$ between set-theoretic representations and the 'unordered' trees above in the following manner:

$$\begin{aligned} \{\{x\},\{x,y\}\}' &= [x'\ y'] \\ \{\{x\}\}' &= [x'\ x'] \\ \ell' &= \ell \end{aligned}$$

(3) John will eat.

(4) John will eat the ointment.

The fact that *devour* (but not *eat*) requires an overt argument is a brute one—it does not seem to be derivable from anything else. The obvious contender, that this fact can be derived from the semantic information associated with these words, is made less appealing by the lack of any obvious generalization about which of the semantic properties of words are predictive of the optionality of semantic arguments (*consume*, which is a near synonym of *eat*, behaves like *devour* in this respect), as well as by the fact that in other languages, synonyms of *eat* may behave like the English *devour* in that they require their semantic argument to be expressed overtly. Asante Twi (Kwa:Niger-Congo) is one such:[3]

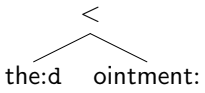
(5) **Kwasi bedi.*
Kwasi FUT.eat
"Kwasi will eat."

(6) *Kwasi bedi fufu.*
Kwasi FUT.eat fufu.
"Kwasi will eat fufu."

Consequently, any complete description of our linguistic competence will have to record that *devour* requires a syntactic object. Furthermore, *devour* only requires one object. In other words, once *devour* obtains a syntactic object, its requirement is fulfilled. In our present system we represent these facts by simply recording, for each word, which properties and requirements it has. This recording is done in terms of features, which come in two polarities: a *categorial* feature, say `f`, indicates that the lexical item has property F, and a *selection* feature, `=f`, indicates that the lexical item requires another with property F. Two features 'match' just in case one is `f` and the other `=f`, for some F. Merge then combines expressions with matching features, and then *checks*, or deletes, those features from the resulting expression. In the resulting expression, that argument projects that had the `=f` feature. The intuition is that the expression with the `=f` feature is acting as a functor, and the expression with the `f` feature as its argument. In more traditional

[3]Asante Twi has a seperate lexeme, *didi*, for the intransitive usage of *eat*. Despite the similarity in form between *di* and *didi*, this is the only transitive-intransitive pair in the language which can reasonably be analyzed as deriving from a single lexical item via an operation of reduplication (which is otherwise abundantly attested in the language). Thus, their superficial similarity notwithstanding, they are not plausibly related in a synchronic grammar of Twi.

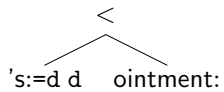
terms, the expression with the `=f` feature is the head, and the one with the `f` its dependent.[4] There are non-trivial correlations between the order of heads and dependents of various categories within a language [32]. For example, languages in which objects precede their verbs (dependent–head order) also tend to have nouns precede adpositions (postpositions), and sentences precede complementizers. Similarly, languages in which objects follow their verbs (head–dependent order) tend to have nouns following adpositions (prepositions) and sentences following complementizers. English has head–dependent order (verbs precede objects, adpositions their nouns, etc), and so we attempt to give this a principled explanation by stipulating that merge combines lexical heads and dependents in that order. Thus, given lexical items `the::=n d` and `ointment::n`, merge builds the following complex expression



In English, if there are two dependents, the dependents flank the head. One example is the subject-verb-object word order of English, assuming that both subject and object are dependents of the verb. Another example is given by the Saxon genitive construction (7), assuming that the marker *'s* is the head of the construction, a two-argument version of *the*.

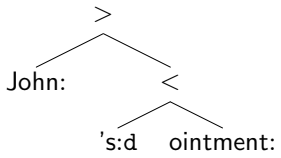
(7) John's ointment

Accordingly, we stipulate that merge combines non-lexical heads and dependents with the dependent preceding the non-lexical head. Adding lexical items `'s::=n =d d` and `John::d` we can derive (7) in two steps. First, we merge *'s* and *ointment* to get



and then we merge the above expression with *John* to get

[4] In the idiom of the day we are assuming that all syntactic features are 'uninterpretable' in the sense that the only formal feature which is allowed to be present in a complete sentence is the categorial feature of the head of that sentence. The conception of grammar that naturally follows from this is of a *resource-sensitive* system [31, 73], which perspective is present also in the categorial type-logic community [68, 74].

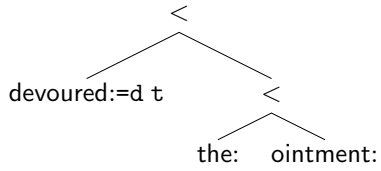


Note that it is crucial here that the syntactic features of an expression are checked in a paricular order—otherwise we couldn't distinguish syntactically between *ointment's John* and the above. We call the expression first merged with a lexical item its *complement*, and later merged expressions are *specifiers*.

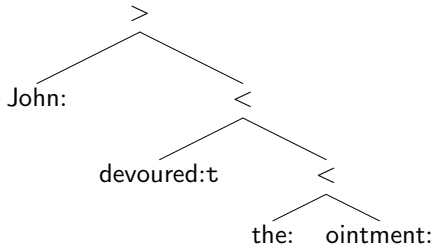
To derive a simple transitive sentence like (8),

(8) John devoured the ointment

we add the lexical item devoured::=d =d t to our lexicon. We merge *devoured* with the complex expression *the ointment* that we derived above to get



and then we merge the above expression with *John* to get

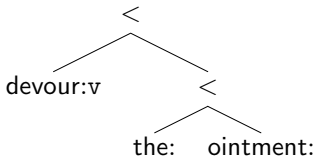


Now consider the following sentences

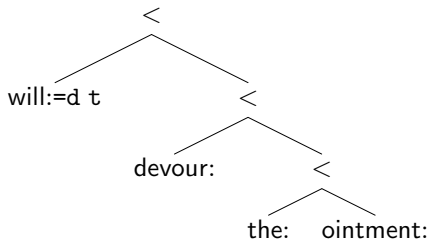
(9) John will devour the ointment

(10) John is devouring the ointment

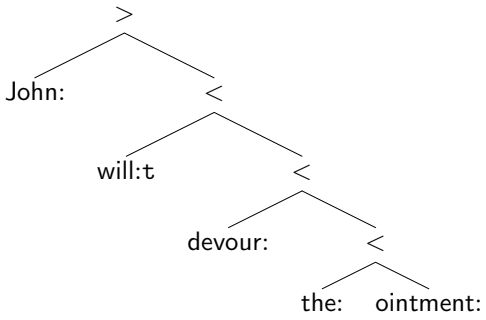
To generate the first of these (9) we need to add to our lexicon the expressions will::=v =d t and devour::=d v. Then we can merge *devour* with *the ointment*



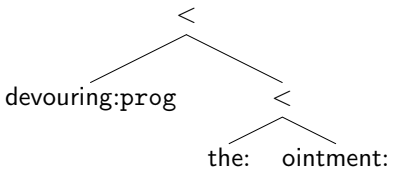
and then *will* with *devour the ointment*



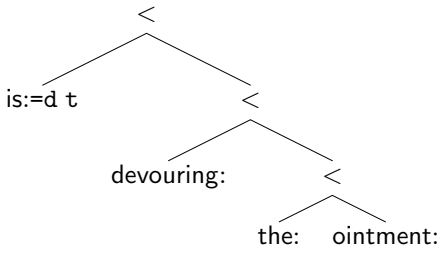
and finally *will devour the ointment* with *John* to get



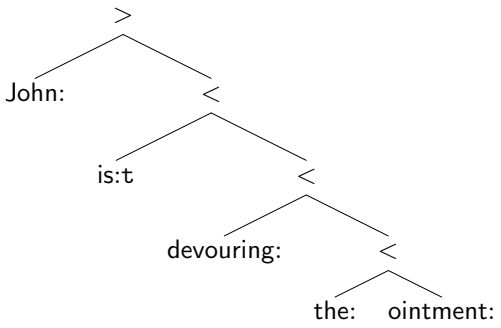
To generate sentence 10, we need to extend our lexicon again with the expressions is::=prog =d t and devouring::=d prog. Now we simply merge *devouring* with *the ointment*



and *is* with *devouring the ointment*



and finally *is devouring the ointment* with *John* to get



1.3 Head Movement

So far so good, but now consider the following paradigm of sentences

(11) John devours the ointment

(12) John devoured the ointment

(13) John will devour the ointment

(14) John has devoured the ointment

(15) John had devoured the ointment

(16) John will have devoured the ointment

(17) John is devouring the ointment

(18) John was devouring the ointment

(19) John will be devouring the ointment

(20) John has been devouring the ointment

(21) John had been devouring the ointment

(22) John will have been devouring the ointment

To describe these twelve sentences, we might postulate the existence of the lexical items in figure 1. However, there seem to be regularities in the English auxiliary system that are not captured by this straightforward analysis. For example, all expressions with an `t` feature have an additional `=d` feature, in comparison with their counterparts without the `t` feature. Furthermore, every one of the *devour* lexical items has at least one `=d` feature (those that have more just have one more, which is predictable given the comment

will::=v =d t	has::=perf =d t	is::=prog =d t	devours::=d =d t
	had::=perf =d t	was::=prog =d t	devoured::=d =d t
	have::=perf v	be::=prog v	devour::=d v
		been::=prog perf	devoured::=d perf
			devouring::=d prog

Figure 1: The English auxiliary system (I)

above). An elegant description of the English auxiliary system, going back to [14], has it that the 'underlying' structure of the system is invariant, with *-en* and *-ing* 'starting out' adjacent to *have* and *be*, respectively:

(will, -s, -ed) have -en be -ing V

Chomsky [14] proposed the existence of transformations that rearrange each of the above affixes so as to be at the right of the next element in the sequence above. The details of this analysis have been refined through the years [4, 91] to talk about movement of heads to other heads (figure 2).[5]

Head movement has enjoyed a rather checkered existence in the GB community, as it appears to violate standard constraints on movement (such as the landing site of movement needing to c-command its trace) and more recently on permissible syntactic operations in general (such as the extension condition, which has it that all syntactic operations must target the root of the trees they are defined over). Some have proposed to eliminate head movement from linguistic theory alltogether [59]. I mean to take no position on this issue here (but see [83] for a systematic evaluation of different approaches to head-like movement). One advantage of having head movement is that it allows us to straightforwardly correlate the structural position of words with their inflectional form without interfering with other syntactic operations. An influential school of thought has it that head movement is not a seperate syntactic operation, but is rather a phonological reflex of morphological properties of expressions [6, 11, 63]. Still, head movement appears sensitive to syntactic boundaries, and not phonological ones, as the actual string distance between the affix and its host is potentially unbounded. For

[5]Head movement does not commit us to a morphemic morphological theory (i.e. Item and Process or Item and Arrangement [39]). It is fully compatible with a Word and Paradigm approach [2, 76, 89]—heads are viewed either as lexemes or as morphological features (*devour* corresponding to DEVOUR and *-s* corresponding to 3^{rd} person singular present tense), and complex heads are then lexemes (which identify the paradigm) associated with various features (which determine the cell in the paradigm).

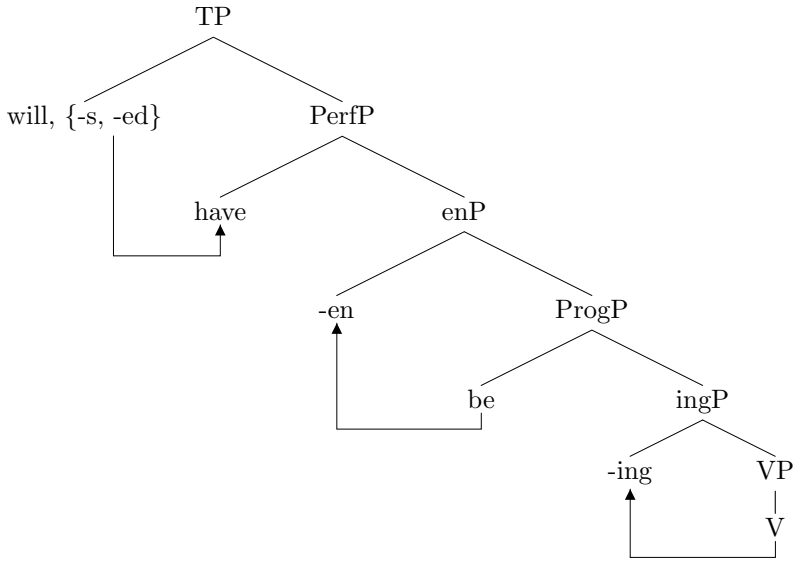


Figure 2: Head movement of auxiliaries in English

example, subject auxiliary inversion in questions is commonly analyzed as head movement of the inflectional element in T^0 to the complementizer position C^0, and the subject, a full DP of potentially unbounded size, intervenes between these two heads (figure 3). So it seems that we cannot leave head

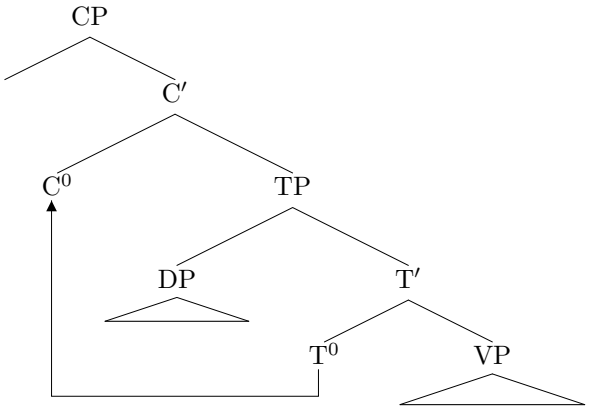


Figure 3: T-to-C movement

movement out of the domain of syntax altogether. Instead, we decide to resolve the morphological requirements of the merged head at each merge step

(as done in [82, 85]).

An affix may either trigger raising of the head of its complement (standard head movement), or may lower itself onto the head of its complement (affix hopping) in order to satisfy its morpho-phonological dependence.[6] For simplicity, we assume that the *syntactic* effects of this morphological combination (whether the base raises to the affix or the affix lowers to the base) are determined by the affix itself. As the 'base' in question is always the head of the complement (i.e. the first merged phrase) to the affix, we indicate on the complement-selecting `=f` feature itself whether the lexical item bearing said feature is an affix, and, if it is, whether it triggers raising of the head of its complement, or it lowers onto the head of its complement. In the first case (raising), we write `=>f`, and in the second (lowering), we write `f=>` . Using head movement, we can describe the English auxiliary system more succinctly (figure 4).[7] [8]

will::=perf =d t	have::=en perf	be::=ing prog	devour::=d v
-s::perf=> =d t	-en::=>prog en	-ing::=>v ing	
-ed::perf=> =d t	ϵ::=>prog perf	ϵ::=>v prog	

Figure 4: The English auxiliary system (II)

We derive the sentence "John is devouring the ointment" as follows. First, we merge *devour* with the phrase *the ointment*

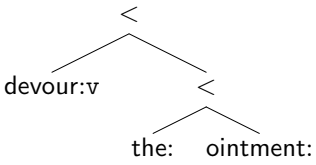
[6]Note that we are assuming that affixes are all structurally higher than the bases to which they attach. This is by no means a necessary assumption, but it not only seems to hold given our assumptions thus far about English, but also seems to be relatable to a cross-linguistic tendency for more 'abstract' meanings to be expressed via bound morphemes more frequently than more 'contentful' meanings [10]. In the government and binding style perspective we adopt here, more contentful expressions tend to occupy hierarchically subordinate positions to the more abstract, or 'functional' expressions.

[7]Although now we generate terminal sequences like

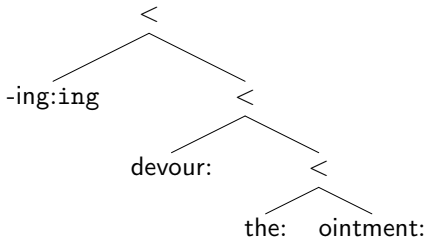
John be -s kiss -ing Mary

which is yet another step removed from the data. Thus, the apparent elegance of this approach needs to be evaluated in conjunction with the modifications necessary to the morphological module implicitly appealed to.

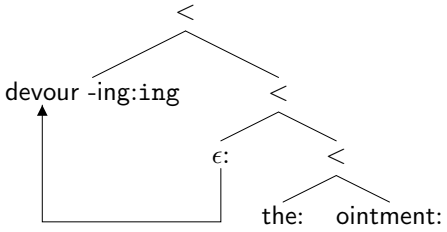
[8]The 'empty' lexical items in figure 4 serve the same function as lexical redundancy rules—they encode generalizations about predictable patterns in the lexicon; without them we would have three entries for every (transitive) verb differing only in their categorial feature (this would still constitute an improvement over the lexicon in figure 1, which requires five entries for each verb in English).



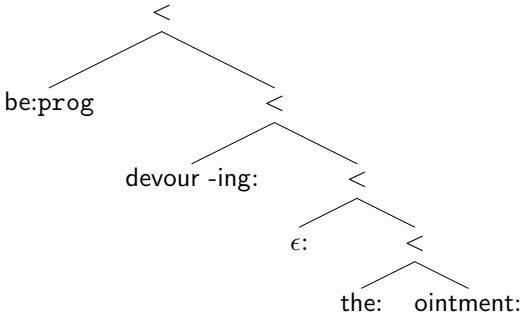
We then merge *-ing* with *devour the ointment*



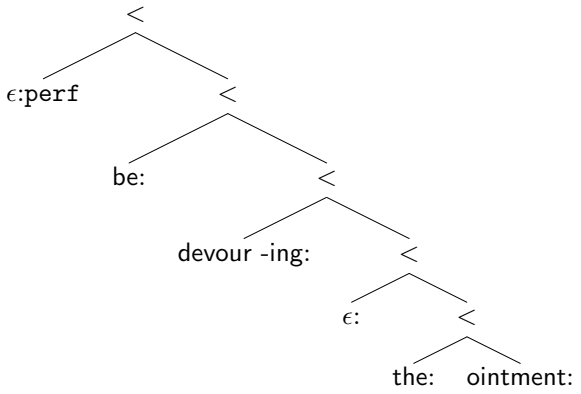
and the morphological properties of the merged affix trigger morphological readjustment—the head *devour* raises to the head *-ing*:



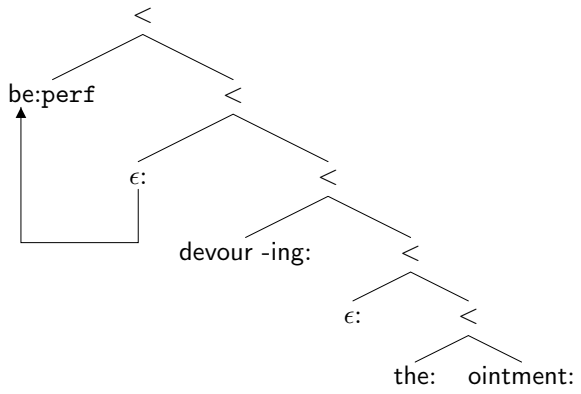
and *be* with *devouring the ointment*



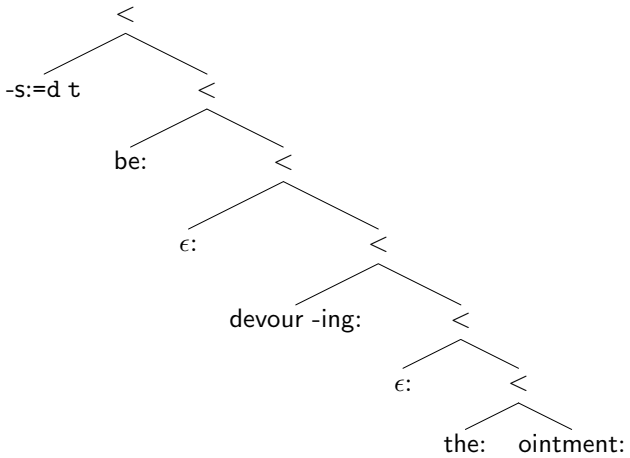
and ϵ::`=>prog perf` with *be devouring the ointment*



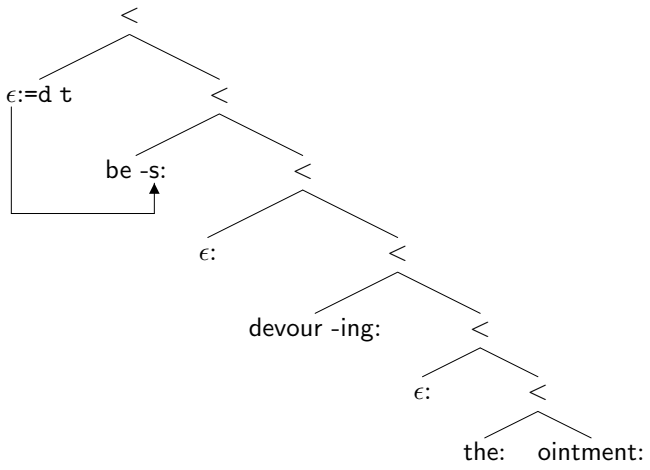
and again, the morphological properties of the merged affix trigger raising of the head *be* to the higher head:



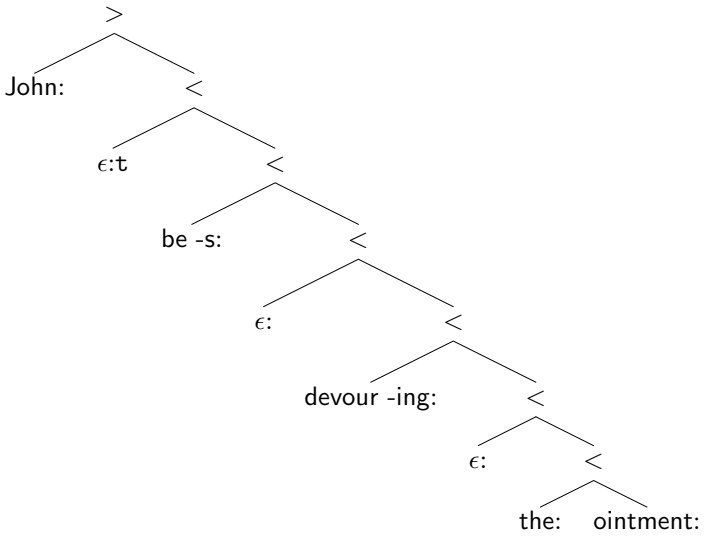
and *-s* with the above phrase



and once more the morphological properties of the suffix *-s* trigger morphological readjustment, this time lowering *-s* to the head of its complement



and finally *is devouring the ointment* with *John*



This approach bears more than just a passing similarity to [63], where head movement is also viewed as the result of a morphological operation (there called 'm-merger') which applies during the course of the derivation. Head movement as implemented here is a representative of a 'post-syntactic' (really: 'extra-syntactic') readjustment operation (as are commonplace in Distributed Morphology [24, 35]), and serves to illustrate some interesting properties of the copy mechanism introduced in chapter 4 of Kobele [48].

1.4 Phrasal Movement

Consider the following sentences.

(23) A pirate ship appeared (in the distance).

(24) *A pirate ship to appear (in the distance).

(25) A pirate ship seemed to appear (in the distance).

(26) *A pirate ship seemed appeared (in the distance).

It appears that non-finite clauses (sentences with *to*) do not license surface subjects. Assuming that *will* and *to* have the same category,[9] we can explain the above data with the simple addition of the expression `to::=perf t` to our lexicon, and of the sentential complement-taking verb `seem::=t v`. Note however that this move makes the choice of subject independent of the choice of (lower) verb. While it has been argued that so-called external arguments of verbs should be selected not by the verb itself but by a structurally higher projection [53, 61, 72], this is not as straightforward as our current theory would seem to predict.[10]

(27) There appeared a pirate ship (in the distance).

(28) *There sunk a pirate ship (in the distance).

(29) There seemed to appear a pirate ship.

(30) *There seemed to sink a pirate ship.

While the precise factors that license expletive-*there* subjects are not well understood, it seems that the question as to whether expletive-*there* is licensed is fully determined by the choice of verb (phrase) in the lower clause (*appear* vs *sink*), and is independent of the choice of auxiliary, and of whether *seem*-type verbs intervene, and, if so, of how many. Our current theory is not capable of accounting for these 'long-distance' dependencies in a simple and

[9] Although this is not true in terms of surface distribution, as shown by the position of negation in the examples below, it is good enough for our purposes here.

1. John will not have kissed the lizard (by daybreak).
2. It was best for John to not have kissed the lizard.
3. *John not will have kissed the lizard.
4. It was best for John not to have kissed the lizard.

[10] A treatment of *there*-insertion can be found in chapter 3 of Kobele [48], where long-distance agreement is investigated.

revealing way.[11] There are two basic options available to us at this point. We could either make the information relevant to the decision about whether expletive-*there* is a permissible surface subject available at the point when the surface subject is merged in its canonical position, or we can introduce the surface subject at the point when this information becomes available, and then 'move' it to its surface position. The former option is exemplified by Generalized Phrase Structure Grammar [30] and its decendents. These formalisms can be seen as making information about the derivation available in a 'store', which can be modified through the course of the derivation. Derivation steps can then be made contingent on properties of this store (e.g. 'insert *there* if the store contains `[+there]`'). Wartena [93] provides a detailed investigation of this perspective on non-local dependencies.

The pursuit of the latter option is one of the hallmarks of the transformational tradition in generative grammar. According to this perspective, we have at some intermediate stage of the derivation a structure in which the main clause subject is in the lower clause (figure 5). We thus need to specify

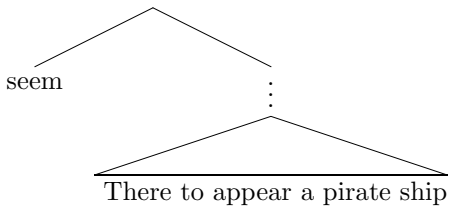


Figure 5: An intermediate stage in the derivation of 29

how the subsequent 'movement' of the subject occurs. As the movement seems to be to particular positions only, and of the subject only (cf (31) and (32)),[12] we conclude that movement is not free, i.e. there are restrictions governing it.

(31) *Will have there$_i$ seemed t_i to be a pirate ship.

[11] All dependencies in our current theory are handled selectionally, and with simplex categories. Thus the lexical items between the subject (or rather, the projection introducing the subject) and the lower verb (or rather, the minimal phrase in which it is determinable whether the expletive is introducable) need to be doubled, so as to allow for the requisite upward percolation of this information. However, as appropriate grammars can be written so as to avoid duplication of contentful material, the grammar size increase is negligible (i.e. a constant). Thus, the intuition about 'simplicity' is unfortunatly little more than a sound bite at this point.

[12] The phrases with index i are intended to have been moved from the positions occupied by the trace t with the same index.

(32) *A pirate ship$_i$ will have seemed there to be t_i.

We decide to make the question of what can move, and what allows movement to it, an idiosyncratic property of lexical items. One question to ask is whether the features driving movement are the same as the features driving merger. To simplify matters (by cutting down on the number and kind of interactions in the system), we assume that movement features and merger features are drawn from disjoint sets.[13] Accordingly, we take movement features to be divided into those that license movement (`+f`) and those that indicate a requirement to be moved (`-f`). Where can things move to? One long-standing observation is that movement is generally to a c-commanding position (33).

(33) *(It) t_i hopes that John$_i$ is raining

(John hopes that it is raining)

In other words, movement is upwards in the tree. It is commonly assumed that movement is always to the root of the tree (a consequence of Chomsky's [16] 'Extension Condition'). We adopt this here. We adopt also a general principle of 'immediacy', which (informally) requires that an expression move as soon as possible. More precisely, an expression with licensee feature `-x` will move as soon as the root of the tree makes available a licensor feature `+x`. This principle allows us to derive something like Rizzi's [75] Relativized Minimality. Given our principle, we know that if a tree contains more than one expression with the same accessible licensee feature, it is not part of a convergent derivation.[14] This means that move is a fully deterministic operation (i.e. a function)—given an expression whose root has first feature `+x`, it will be in the domain of move just in case it has exactly one proper constituent whose head has first feature `-x`. In that case, that constituent moves to the specifier of the root.

Koopman and Sportiche [52] note that the same considerations that motivate a raising analysis for *seem*-type clauses (that the information about what kind of subject is permitted is present already in the embedded clause verb) suggest that the subject is introduced below the auxiliaries as well. Let

[13]Relaxing this assumption (as done in [84]) might provide the foundation for a natural account of expletive subjects (*it*), which have a sort of 'last resort' quality, in that they seem to appear in case positions (`+k` in our notation), in case there is no other element which needs case in the tree.

[14]Constraints on movement can be incorporated into the domain of the move operation. Different constraints result in different generative power [29, 49, 65]. Thus, although we adopt here the principle of immediacy, and thereby something like the minimal link condition, other constraints on extraction are straightforwardly adaptable to this system.

us christen the feature that drives movement of the subject to the surface subject position 'K', and let us systematically substitute the feature `+k` for the feature `=d` in our auxiliaries. Our current inflectional lexical items are shown in figure 6.

will::=perf +k t	have::=en perf	be::=ing prog
-s::perf=> +k t	-en::=>prog en	-ing::=>v ing
-ed::perf=> +k t	ϵ::=>prog perf	ϵ::=>v prog
to::=perf t		

Figure 6: The English auxiliary system (III)

Then the sentences

(34) John has arrived.

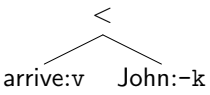
(35) John seems to have arrived.

(36) John seems to seem to have arrived.

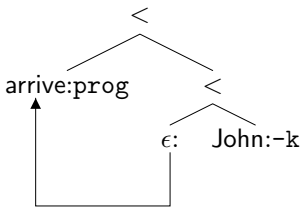
⋮

can be derived using the lexical items John::d -k, arrive::=d v, seem::=t v and the auxiliaries from figure 6.

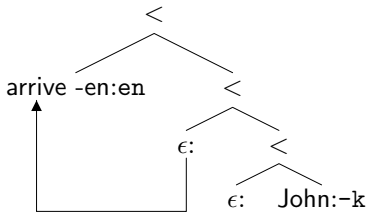
We derive the sentence *John has arrived* by first merging *arrive* and *John*,



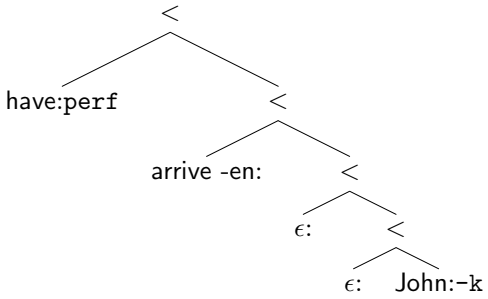
merging ϵ::=>v prog with the above expression (with the concommittant morphological readjustment)



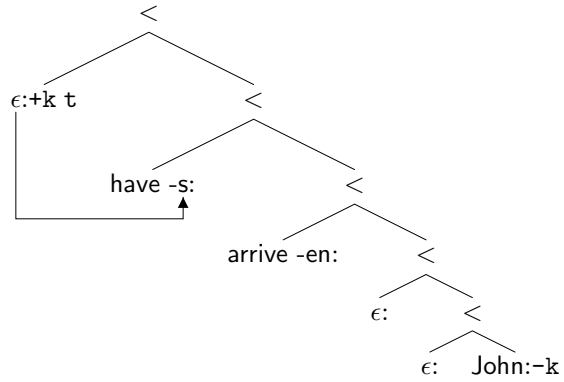
merging *-en* with the above (and resolving the morphological requirements of the merged affix)



merging *have* with the above

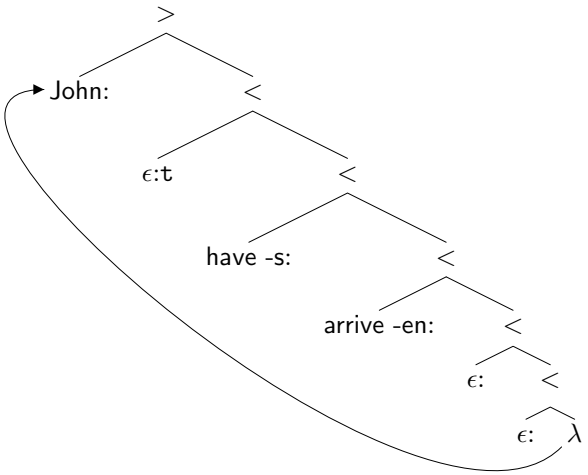


merging *-s* with the above (and lowering onto the head of its complement)

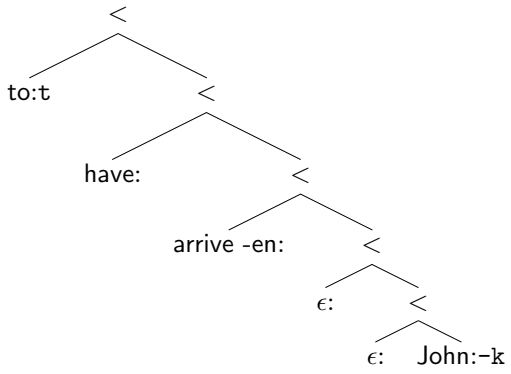


and then finally moving *John* in the above expression.[15]

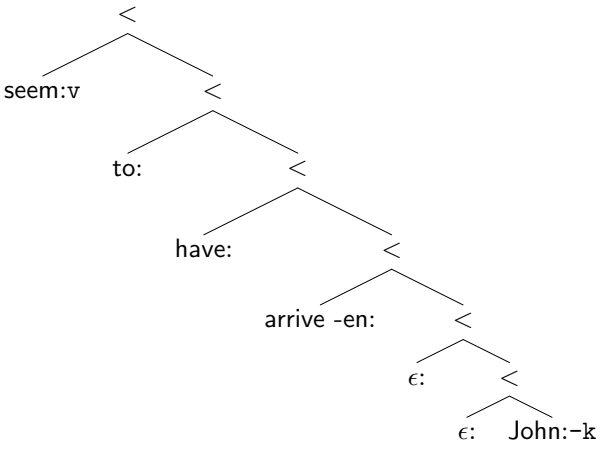
[15]For convenience, we represent the source of phrasal movements with a λ.



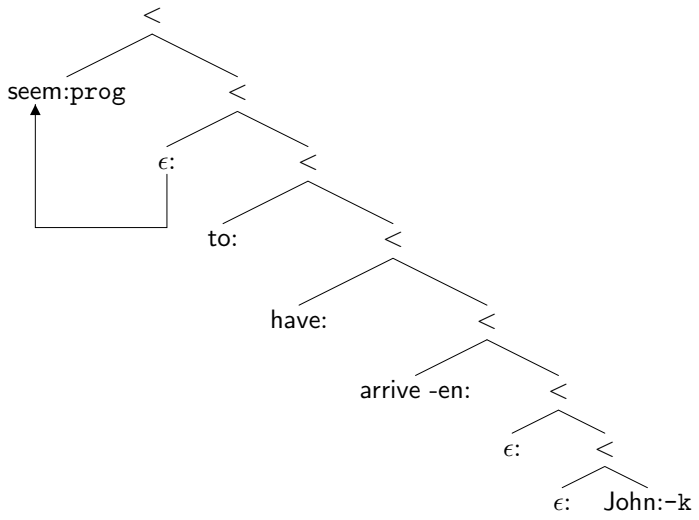
The sentence 'John seems to have arrived' can be derived by merging *to* with the expression *have arrived John* derived above



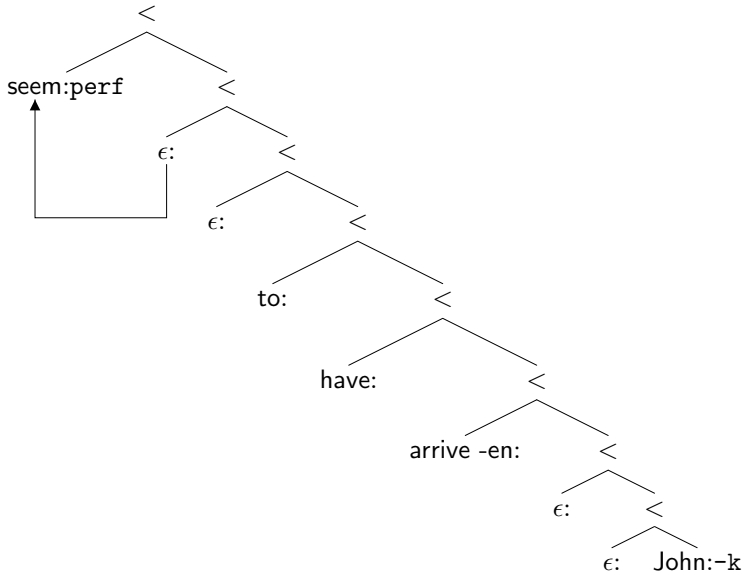
merging *seem* with the above



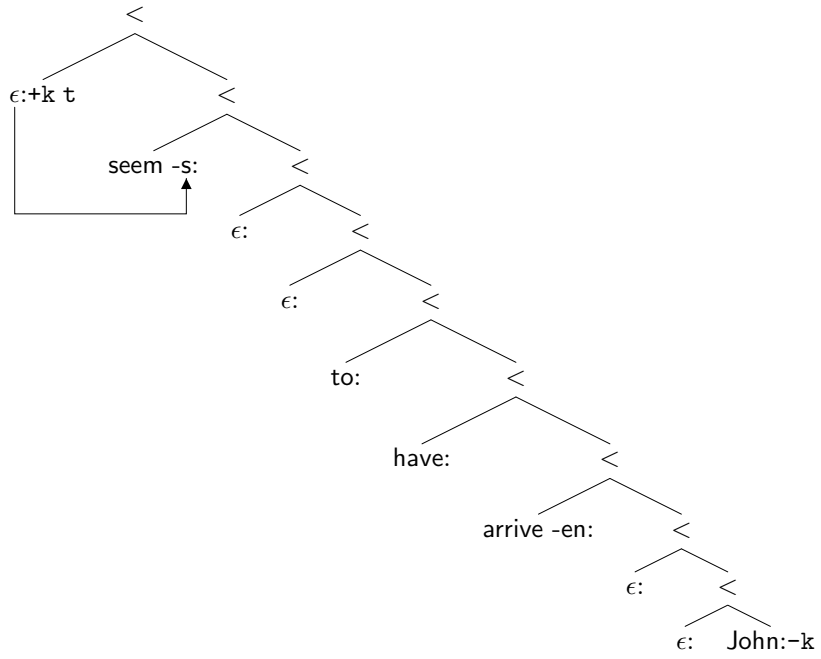
merging ϵ::=>v `prog` with the above expression (and raising the head of its complement)



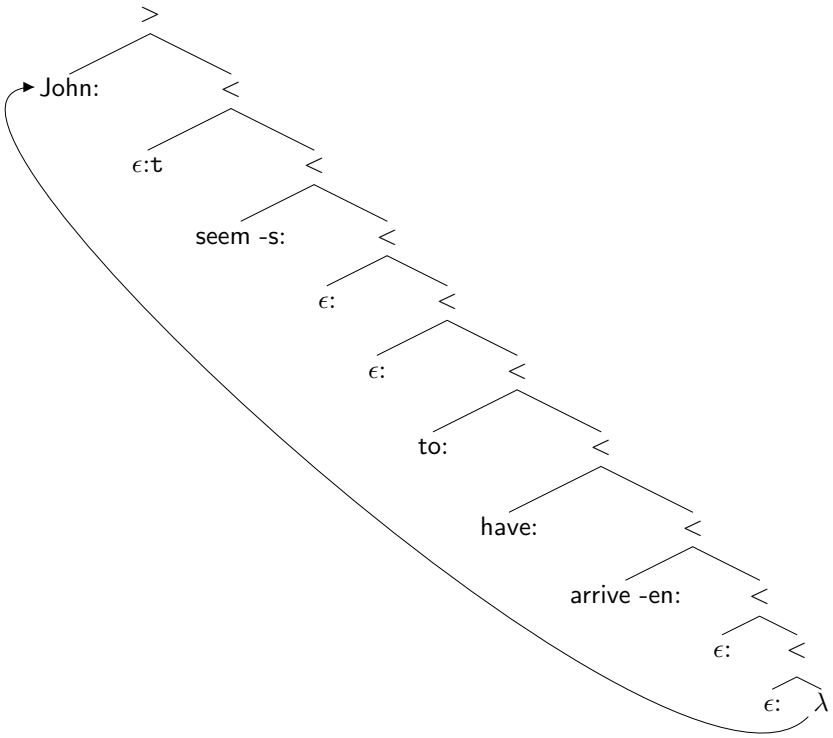
merging ϵ::=>prog `perf` with the expression above (and raising)



merging -*s* with the above expression (and lowering)



and then finally applying move to *seems to have arrived John*



Our current implementation of 'raising to subject' phenomena does not involve successive cyclic movement (see [69] for a protracted defense of this position). We shall see, however, that in the same sense that movement, as described here, simply *is* copy movment, it is also successive cyclic in the strong sense of involving every specifier position between it and its landing site. Thus, the lack of successive cyclicity is only apparent.

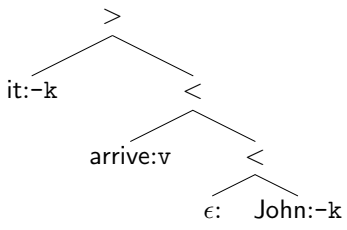
Within the context of our treatment of the raising to subject construction, our principle of immediacy allows for a simple account of the impossibility of so-called super-raising in English (as in 39 below). As is well known, raising sentences (such as 35) have non-raised counterparts (such as 37) in which the subject position of the matrix clause is filled by an expletive *it*.

(37) It seems John has arrived.

(38) *It seems John to have arrived.

(39) *John seems it has arrived.

As tensed clauses in our grammar uniformly bear `+k` features, it is natural, in the light of the sentences above, to treat expletive *it* as bearing a `-k` feature. We allow *it* to be freely introduced at the `v` level (i.e. above the merge position of the subject, and below the `+k` feature of `t`) by means of the lexical items ϵ::`=>v =expl v` and it::`expl -k`.[16] A derivation for the super-raising sentence in 39 would go through an intermediate form such as the below (merging ϵ::`=>v =expl v` and then *it* with *arrive John* from above)



which will run afoul of our principle of immediacy as soon as a tense head is merged (in the matrix clause in 38, and in the embedded clause in 39). The grammatical sentence 37 is derived by merging the expletive into the structure after the lower clause subject has checked its `-k` feature (i.e. once

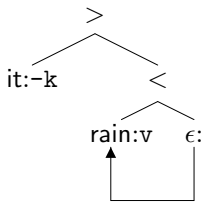
[16] Our implementation of expletive insertion here thus differs from the common treatment in terms of merger licensed by what is here a `+k` feature (with concomitant appeal to numerations and subnumerations and preferences for merge over move). This approach is formally blocked for us by our decision to treat movement and merger features as distinct (see the discussion in footnote 13).

seem has been merged). There is then no conflict with the principle of immediacy when the matrix tense is merged.

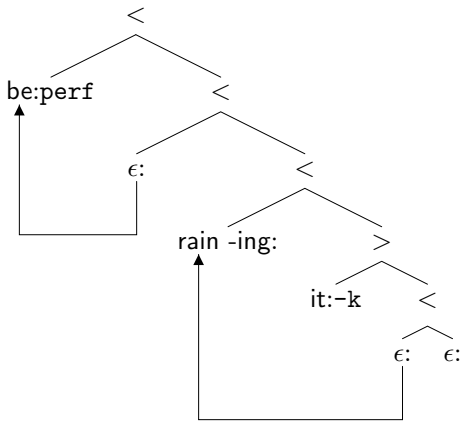
This treatment of expletives immediately extends to encompass 'weather-it', as in 40 below.

(40) It is raining.

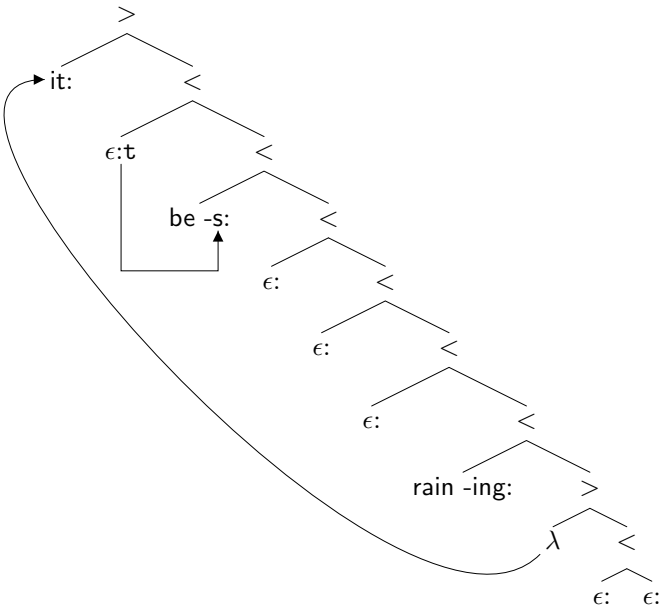
Including the expression rain::v into our lexicon, we derive the sentence *it is raining* in seven steps. We first merge ϵ::=>v =expl v with *rain*, and then continue by merging the resulting expression with *it*.



We then merge *-ing* with the above expression, followed by *be* and then ϵ::=>prog perf.



Finally we merge *-s* with the expression above, and then move *it* to subject position.[17]



We will assume that having the feature sequence '`d -k`' is constitutive of the property of 'being a DP', and therefore that objects as well as subjects are in need of being moved to a position higher than their base position. Thus "the" has the type `=n d -k`. In light of this, consider now the following sentences.

(41) John will have been devouring the ointment.

(42) The ointment will have been being devoured.

(43) George expects the ointment to be devoured.

(44) The ointment is expected to be devoured.

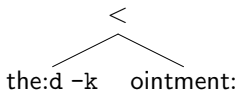
The pair 41 and 42 suggest to us that the external argument (here, the agent *John*) can be dissociated from the verb, and that, in that case, the internal argument moves to the canonical subject position to check its `-k` feature.

[17]This analysis makes sentences like "It seems to have rained" ambiguous, as *it* may be introduced in either clause. This seems to be a case of 'spurious ambiguity' (where there are two or more different derivations of a sentence corresponding to the same meaning), analogous to the situation in various deduction systems, where numerous (typically an infinite number) proofs are derivable for any given sequent. The ideal of one proof–one meaning can be recovered by either placing constraints on what is to count as an acceptable proof in the current system (normalization), or by developing a novel representation that equates irrelevantly different proofs (one example of such is given by proof nets in Linear logic and the Lambek calculus).

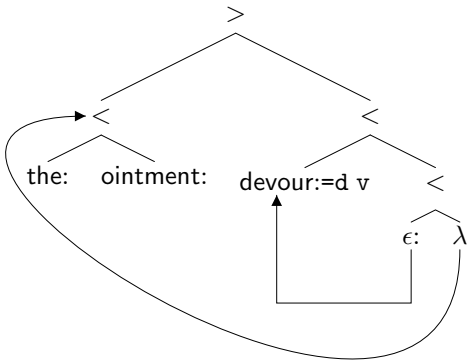
In other words, the possibility of checking a `-k` feature internal to the verb phrase is dependent upon whether the external arugment is present. This is essentially Burzio's generalization [9]:

> A verb which lacks an external argument fails to assign accusative case.

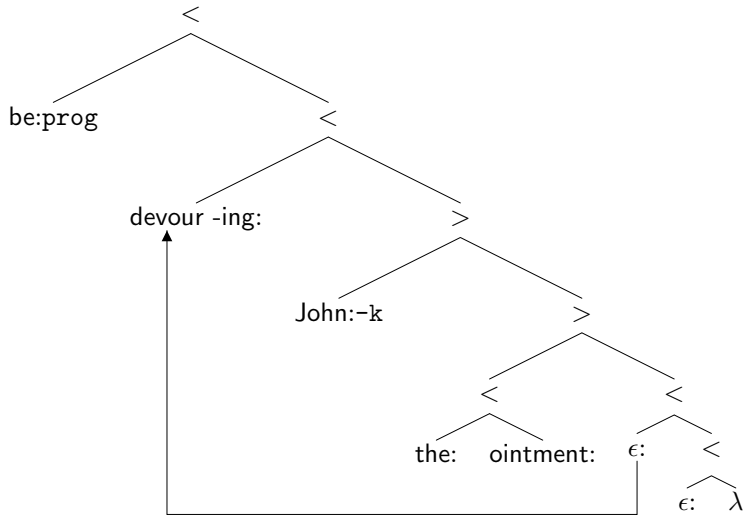
In a basic transitive sentence like 41, our principle of immediacy forces us to check the `-k` feature of the object before the subject is introduced (a position familiar from [51, 52]). We assign to the lexeme "devour" the type `=d V`, and recast Burzio's generalization in our system with the lexical item ϵ::`=>V +k =d v`. Then 41 can be derived by first merging *the* and *ointment*.



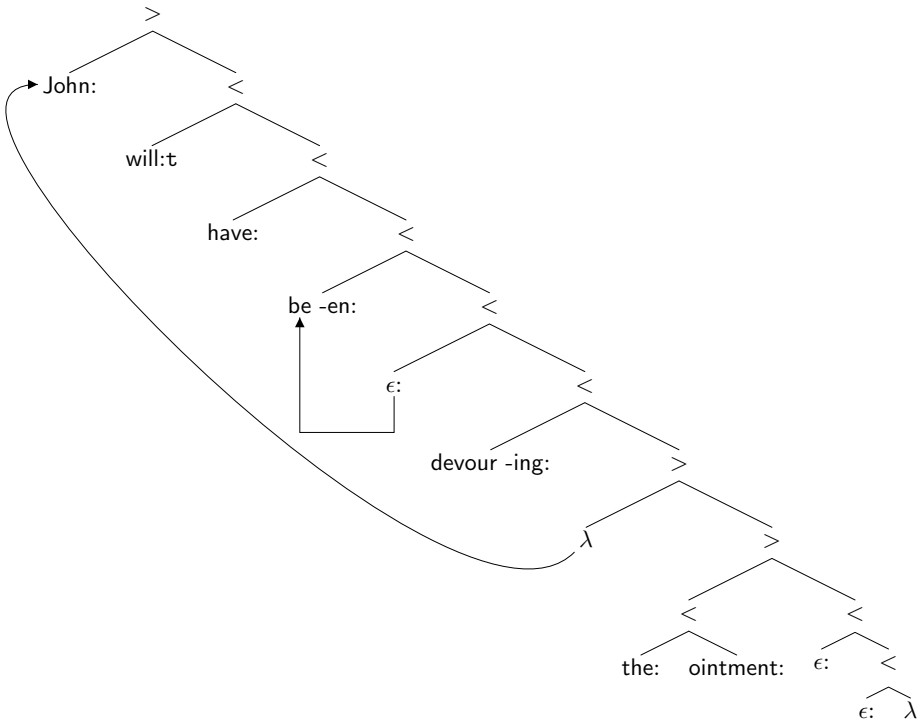
Then *devour* is merged with *the ointment*, and then ϵ::`=>V +k =d v` with the resulting expression (with the appropriate morphological readjustments), followed by movement of *the ointment*.



We continue by merging the expression above with *John*, and then merging *-ing* with the result, and then *be* with the result of that.

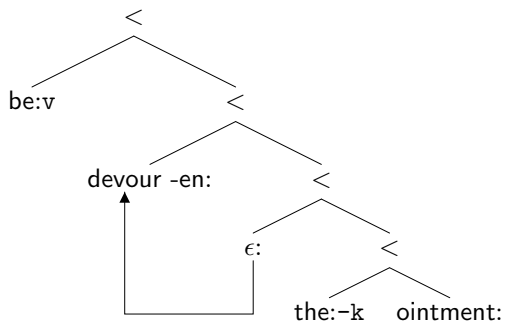


Next we merge *-en* with *be devouring John the ointment* (and then readjusting morphologically), followed by merging *have* with the result, and finally we merge *will* with *have been devouring John the ointment* and then move *John* in the resulting expression.

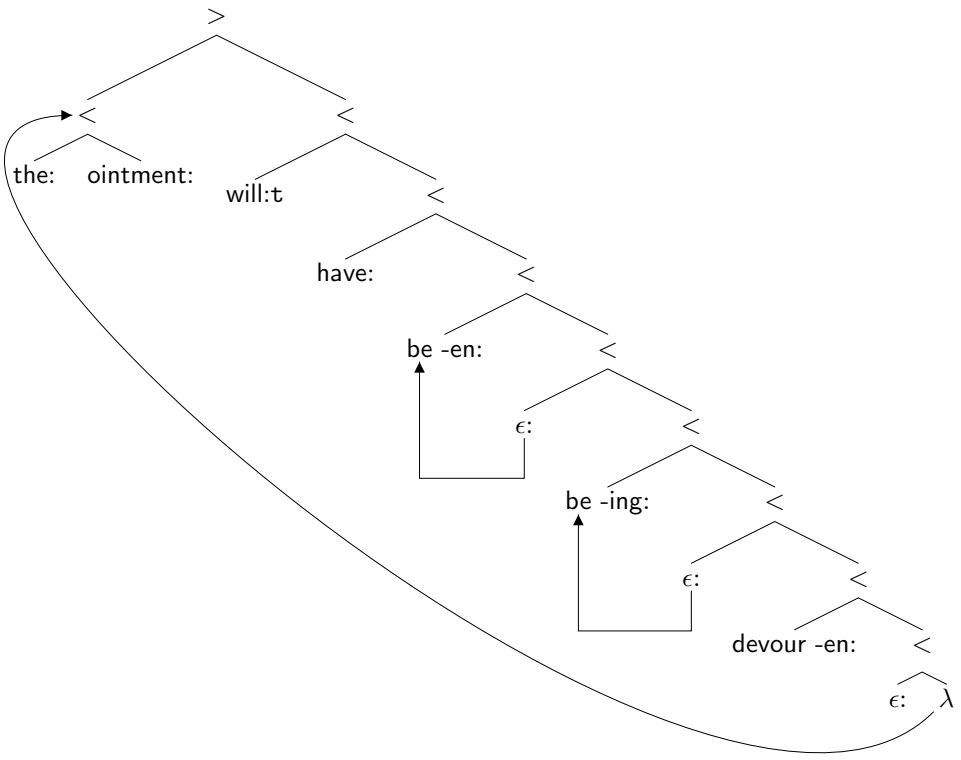


Sentence 42, with its passive 'be' and its associated perfective participial morphology, can be nicely accomodated with our existing head movement

operation. We add the expressions be::=pass v and -en::=>V pass to our lexicon. Sentence 42 can then be derived by first merging passive *-en* with the expression *devour the ointment* derived above followed by merging passive *be* with *devoured the ointment*.

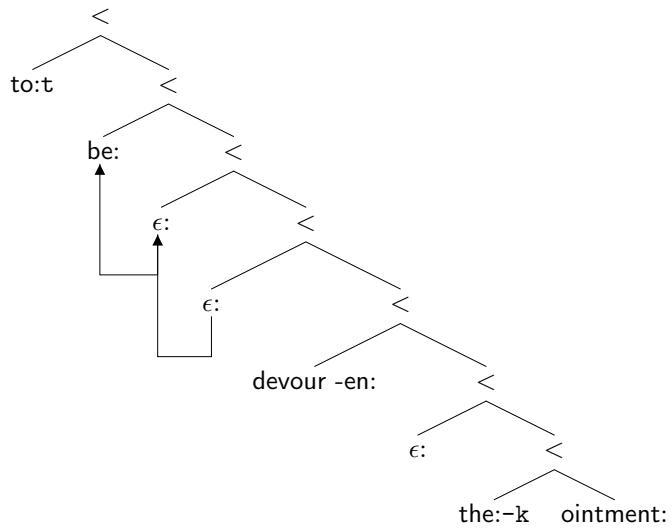


We then successively merge *-ing*, *be*, *-en*, *have* and *will* (performing the necessary morphological operations when appropriate). Finally, we move *the ointment* in the resulting expression.



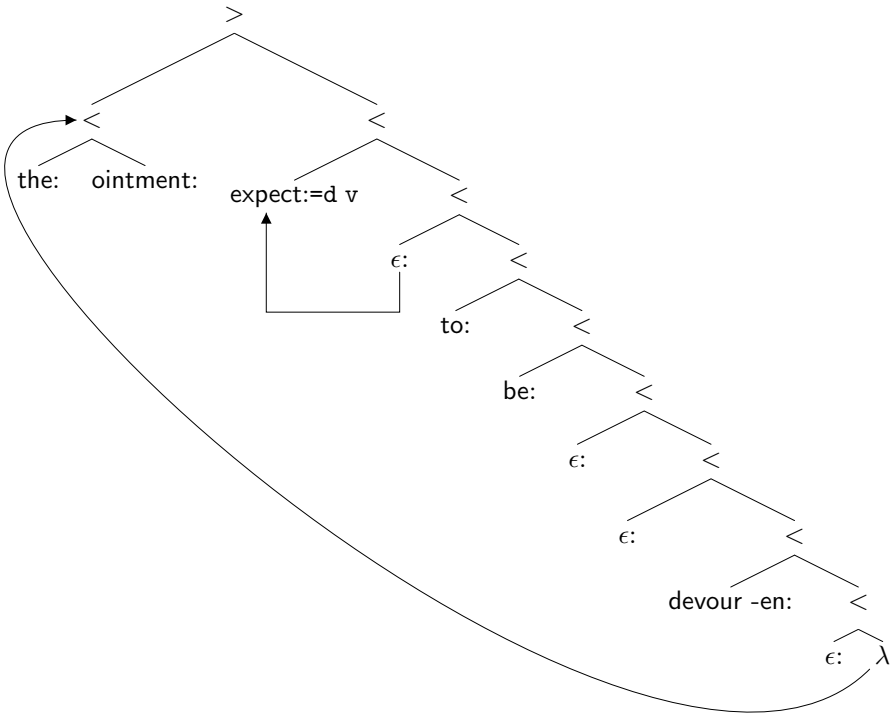
To derive sentences 43 and 44 we need only add the lexical item expect::=t V. Note that, descriptively, we can have raising to object, and then

passivization, and then more raising without any additional stipulations—raising and passive feed each other. Formally, object to subject movement in passives is motivated by the same thing that motivates raising in raising constructions. Sentence 43 is derived by successively merging ϵ::`=>v prog`, ϵ::`=>prog perf`, and *to* (with the appropriate morphological readjustments) with the expression *be devoured the ointment* derived above.

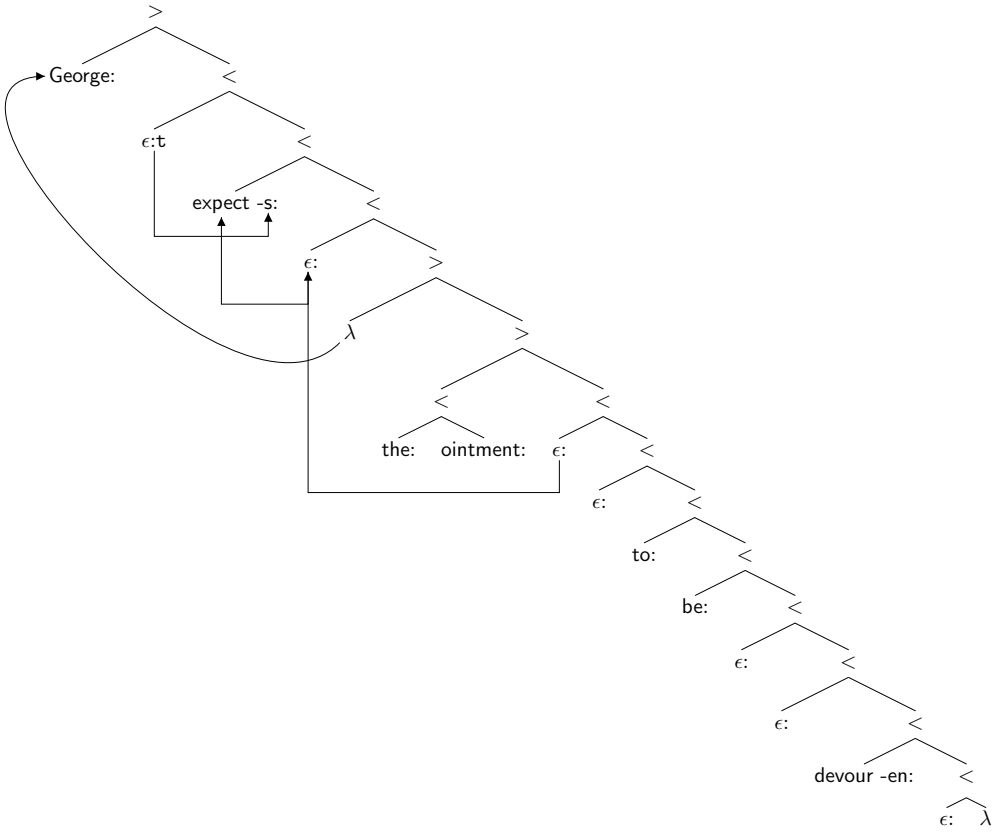


We then merge *expect* and then ϵ::`=>V +k =d v` with the above, followed by movement of *the ointment* into what might be considered to be the object position of the matrix clause (an analysis forcefully argued for in [71]).[18]

[18]Really we have *two* 'object' positions: one in which the object checks its `d` feature, which we can think of as the 'deep' object position, and one in which the object checks its `-k` feature, which can be thought of as the 'surface' object position.



Next we successively merge George::d -k, ϵ::=>v prog, ϵ::=>prog perf, and -s::perf=> +k t. Finally, we move *George* in the resulting expression.



1.5 Interim Summary

This section has motivated the particular assumptions we have made about syntax by showing how natural analyses of A-movement constructions can be directly implemented in our formal framework. For concreteness' sake, our assumptions regarding the architecture of the syntactic component of the grammar are the following:

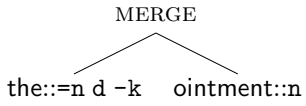
1. we have two feature-driven operations, one binary (merge) and one unary (move)
2. features are checked in a particular order
3. features come in attractor-attractee pairs, and checking is symmetric
4. there are constraints on movement such as the principle of immediacy

We have not discussed many architectural options that are or have been hot topics in the minimalist tradition. Nothing has been said about numerations, lexical subarrays, preferences for merge over move, trans-derivational

economy constraints, covert movement, sidewards movement, adjunction, scrambling, the EPP, or Agree.[19] Also, we have not yet explored how phases fit into the picture we have painted here. In this system, phases are admissible—nothing changes even if every node is a phase. This will be taken up in the next section.

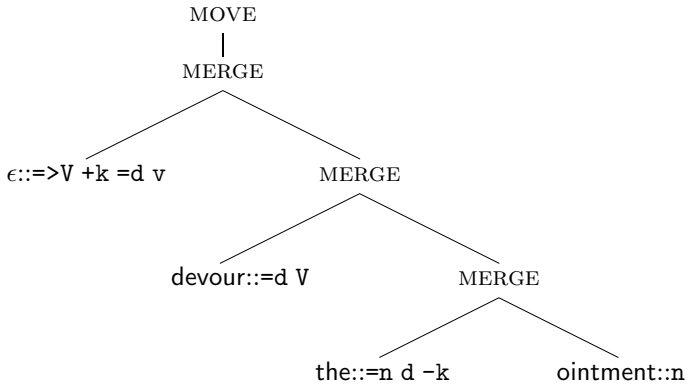
2 Introducing Phases

Recall that our (immediate) goal is to specify the associations between the pronounced form of a sentence on the one hand and the range of interpretations that it may have on the other. The way we have approached this task of defining a relation is by defining a set of abstract objects, together with two operations, which we may call Π and Λ, which map these abstract objects into (mental representations of) pronounced forms, and meanings respectively. In our current system, our lexical items together with our operations of merge and move determine a set of trees; those that can be derived starting from lexical items and repeatedly applying merge or move. Even more abstractly, we can view a derivation as a description of (or even: a recipe for) the process of constructing its object. So a derivation of the expression *the ointment* proceeds by first selecting the lexical items *the* and *ointment*, and then merging them together. We can represent this as a tree:



This derivation can be continued by merging *devour* with the expression *the ointment*, and then by merging the active voice ϵ::=>V +k =d v with the expression just derived, and then moving *the ointment*. We can represent this as a tree:

[19]Some of these have been explored within the context of the assumptions made here elsewhere. In particular, Stabler [84, 85] discusses covert and sidewards movement, and Frey and Gärtner [27] discuss adjunction and scrambling. Covert movement and Agree are discussed and formalized in chapter 3 of Kobele [48].



We can define the set of all possible derivations over a particular lexicon in the following way.

1. First, we take each lexical item to be a derivation which stands for itself

2. Second, given derivations α and β, we describe the merger of the expression stood for by α with the expression stood for by β with the derivation $\textsc{merge}(\alpha, \beta)$, or, as a tree

3. Finally, given a derivation α, we describe the result of applying the operation of move to the expression stood for by α by the derivation $\textsc{move}(\alpha)$, or, as as a tree

The set of possible derivations defined above includes those that do not stand for any well-formed expression (e.g. $\textsc{move}(\textit{the})$). Derivations that do stand for well-formed expressions we will call successful, or convergent. Those that do not we will say have crashed.[20] Note that a successful derivation stands for just a single expression, so we can look at a (successful) derivation as a name of an expression. Note also that expressions may have different names (see footnote 17).

[20]We could define a notion of 'where' an unsuccessful derivation crashes in terms of maximal successful subderivations.

Our conception of grammar thus far can be expressed in the following terms. We have a derivation (*Deriv*) which stands for (i.e. specifies how to construct) a tree (*Tree*), which itself determines a pairing of form and meaning, as given by the maps Π (which turns the tree into a PF-legible object *PF*) and Λ (which transforms the tree into an LF-legible object *LF*). This is schematized in figure 7.[21] The procedural metaphor dominating much

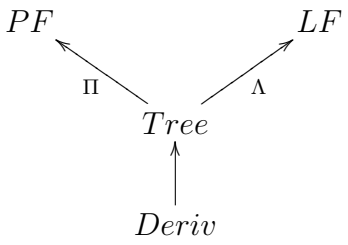


Figure 7: Specifying the associations between sounds and meanings

of syntax, while intuitive, has had the effect of obscuring the distinction between derivational structure, derived (tree) structure, and the mapping between them. Disentangling these notions, we see that there are actually three components to our theory of syntax, the derivation, the derived tree, and the relation between them, and allows us to formulate the question of whether all three components are necessary (i.e. useful) in specifying the relation between form and meaning (i.e. our knowledge of language).

Phases, from our current perspective, should be understood as constraining the relationships between the levels in figure 7 above. To say that the derivation proceeds in phases says that the mappings Π and Λ can be given a recursive bottom-up (what Chomsky [12] calls a 'cyclic') characterization, whereby the mappings Π and Λ are defined such that their output on a particular subtree is determined within a certain structural 'window.' The size of this window is generally taken to be defined in terms of the categorial status of the various nodes in the tree, with at least CPs and v*Ps (active voice phrases) specifying the upper bounds on these windows (where the material between a v*P and a CP may be taken into consideration when computing an expression's PF or LF representation, but nothing more). As pointed out in [62], given the possibility of iterated adjunction (of, say, relative clauses, adjectives, or adverbs) and of iterated raising constructions, the window of

[21]The terms 'PF' and 'LF' are ambiguous in modern generative linguistics, sometimes being used as a name for the process of constructing a representation that is used by non-syntactic systems, and sometimes as a name for the representation so constructed. Here, we reserve the terms 'PF' and 'LF' for the representations used by these interface levels (and also as a cover term for these levels themselves), and use 'Π' and 'Λ' to refer to the process of constructing such a representation from a syntactic object.

context that the mappings to the interfaces may take into account is unbounded in size (and thus the mappings are not guaranteed to be finitely specifiable—an undesirable consequence of a too unrestrictive theory). If the range of possible distinctions that could be drawn were given a principled upper bound, we could eliminate the derived tree altogether, encoding the finitely relevant information about a subtree by parameterizing the mappings Π and Λ (making them homomorphisms with state, or transducers).[22] This has the effect of eliminating both derived structure as well as the relation between derivation and derived structure from our grammar, leaving us with a 'directly compositional' picture of syntax (as shown in figure 8), according to which

> [. . .]syntactic structure is merely the characterization of the process of constructing a [form–meaning pair], rather than a representational level of structure that actually needs to be built[. . .] ([88], pp *xi*)

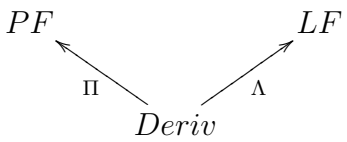


Figure 8: Directly interpreting derivations

We shall see how this project can be realized in minimalist grammars in the remainder of this section. We first show (§ 2.1) how the mapping Π from the derived tree to PF can be computed without needing to construct the derived tree in the first place. We then (§ 2.2) show how to construct an LF-legible representation directly from the derivation, modifying Heim and Kratzer's [37] semantic theory to this more dynamic perspective. To achieve this we make precise a long-standing intuition that "chains and not chain-members are the elements input to principles of interpretation" ([8], pp. 130). We end this section by extending our grammar for A-movement to account for cases of obligatory control, as well as for basic quantifier scope ambiguities.

[22]This is investigated in Kobele et al. [50], where it is shown that in the grammars presented here the mappings Π and Λ can be realized by deterministic top-down tree transducers with regular look-ahead [25]. As the set of well-formed derivation trees of minimalist grammars is recognizable, the form-meaning mappings computed by minimalist grammars are bimorphisms of type $B(M, M)$, where M is the class of tree homomorphisms [80].

2.1 To PF Without Trees

We begin with the observation that much of the tree structure we are representing expressions as having is functionally inert—no operation of the (syntactic) grammar 'cares' about the internal structure of subtrees that have no syntactic features. Consider the expressions in figure 9. These expressions

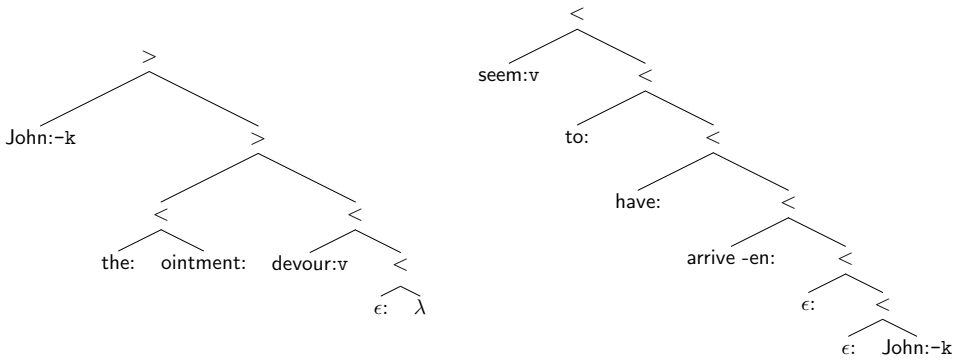


Figure 9: Syntactically indistinguishable expressions

are indistinguishable syntactically—the heads of both have the feature sequence `v`, and both have exactly one constituent with licensee features, the heads of which share the same feature sequence: `-k`. The position within the tree-structures of the moving constituent is not relevent in our current formalism, and so needn't be explicitly represented. Eliminating the syntactically superfluous information contained in the derived structures above, we can represent these expressions in the following manner.

$$(\text{the ointment, devour, } \epsilon) : \texttt{v}, (\text{John, } \texttt{-k})$$

$$(\epsilon, \text{ seem, to have arrive -en}) : \texttt{v}, (\text{John, } \texttt{-k})$$

More generally, we can go from derived trees to these more minimal representations in two steps. First, we build a list of all of the constituents in the derived tree t whose heads have a licensee feature `-x`, and remove these constituents from t (in case a constituent whose head bears a licensee feature itself containts another constituent with the same property, we remove this latter constituent from the former and put it into the list). For each sub-tree s in this list, we replace it with the pair $(yield(s), \delta_s)$, where δ_s is the feature sequence of the head of s, and $yield(s)$ is the interpretation of s at the PF interface (which we will represent as a string of lexemes). Finally, for t' the result of removing each such subtree from t, we replace t' with the

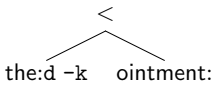
object $(edge_{t'}, head_{t'}, interior_{t'}) : \delta_{t'}$, where $\delta_{t'}$ is the feature sequence of the head of t', $edge_{t'}$ is the interpretation of the specifiers of the head of t' at the PF interface (the spell-out of the material in the specifiers of the head of t'), $head_{t'}$ is the interpretation at the PF interface of the head of t', and $interior_{t'}$ is the interpretation at the PF interface of the complement of the head of t'. Schematically,

(Spec, Head, Comp) : `features`, Moving sub-constituents

Lexical items like John::`d -k` are considered as abbreviations for representations like (ϵ, John, ϵ)::`d -k`.

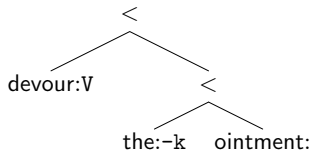
We work through a derivation of the sentence "John devoured the ointment," showing at each step both the derived tree, as well as its abbreviation according to our convention. The crucial fact of note is that the operations of merge and move can be *directly defined* over these reduced expressions, rendering the derived tree unnecessary for the computation of a PF legible representation. Precise definitions of the generating functions are given in appendix A.2.

We begin by merging the::`=n d -k` and ointment::`n` to get the expression below, which is abbreviated by the representation below it.



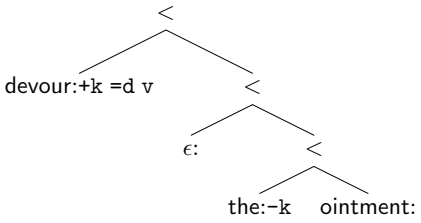
(ϵ, the, ointment) : `d -k`

Next we merge devour::`=d V` with the above expression.



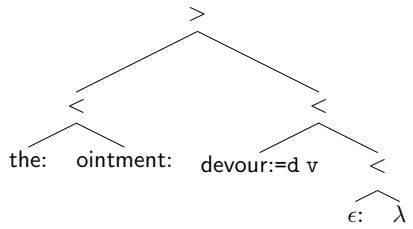
(ϵ, devour, ϵ) : `V`, (the ointment, `-k`)

We merge ϵ::`=>V +k =d v` with the expression above, performing the accompanying morphological readjustment, to get the below.



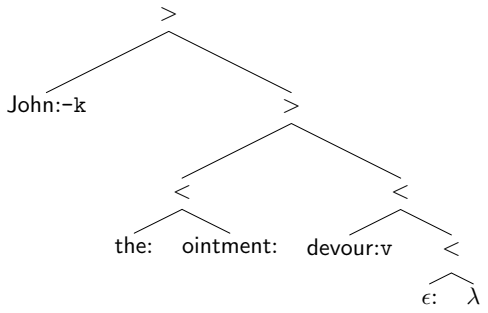
$(\epsilon,$ devour, $\epsilon)$: +k =d v, (the ointment, -k)

Move applies to the expression thus derived, to get



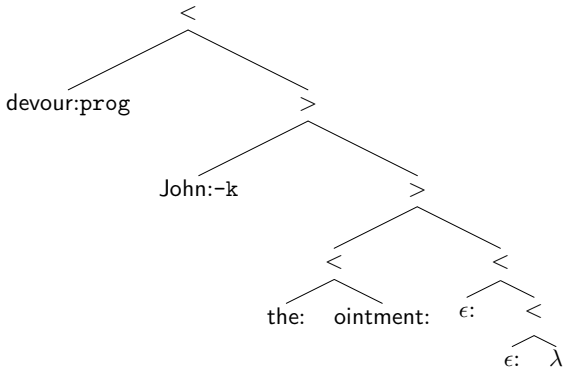
(the ointment, devour, ϵ) : =d v

The above expression merges with John::d -k.



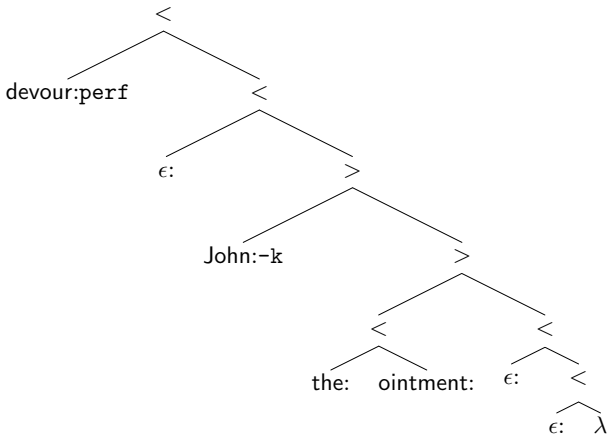
(the ointment, devour, ϵ) : v, (John, -k)

Next the lexical item ϵ::=>v prog is merged with the above.



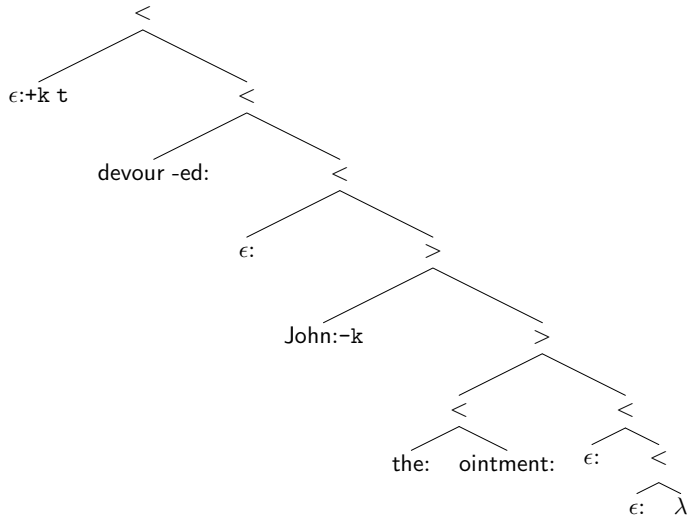
(ϵ, devour, the ointment) : prog, (John, -k)

Then we merge ϵ::=>prog perf with the expression above.



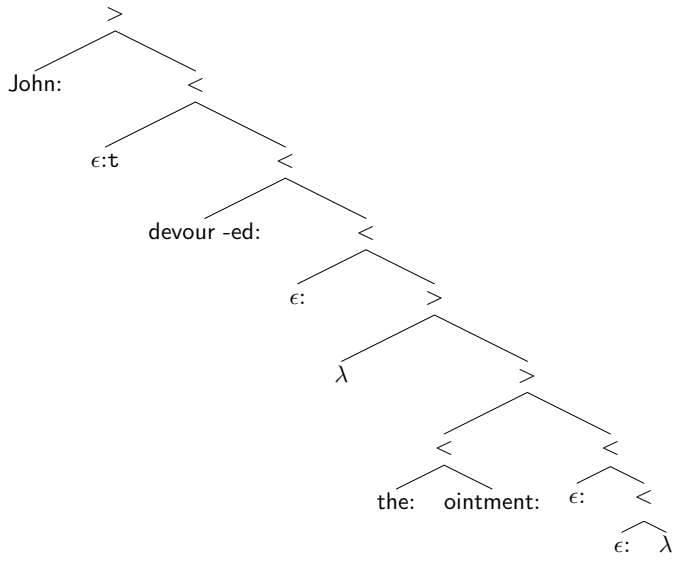
(ϵ, devour, the ointment) : perf, (John, -k)

Finally we merge -ed::perf=> +k t.



(ϵ, ϵ, devour -ed the ointment) : +k t, (John, -k)

Move applies to the above expression, yielding the desired



(John, ϵ, devour -ed the ointment) : t

2.1.1 Successive Cyclic Movement

As mentioned in § 1.4, our implementation of minimalist grammars over trees did not involve successive cyclic movement in any obvious way. Before moving on to the presentation of a compositional semantics for minimalist grammars (§ 2.2), we will see how a strong version of successive cyclicity is nonetheless maintained in our formalism.

There is near-universal agreement that non-finite TPs in English (those headed by *to*) are defective in some sense, in comparison with finite TPs. In our present system, this 'defectivity' is captured by the lack of a +k feature on *to* (i.e. non-finite T doesn't assign case). There is still disagreement in the transformational-generative community over whether DPs which undergo raising move through the non-finite T.[23] This movement has a different character than the others we have or will encounter in this thesis. Unlike our other movements (thus far driven all by -k features), successive cyclic movement does not appear to check features of the expressions moving—the self-same DP may raise once, twice, or not at all.

Consider the two derived trees in figure 10, which represent the raising and non-raising analysis of the same point in the derivation of the expression *John seems to have arrived*, respectively. In our new notation, the same expression represents both of the trees in figure 10.

(ϵ, to, have arrive -en) : t, (John, -k)

[23]This question is currently posable as: "Does non-finite T have an EPP feature?"

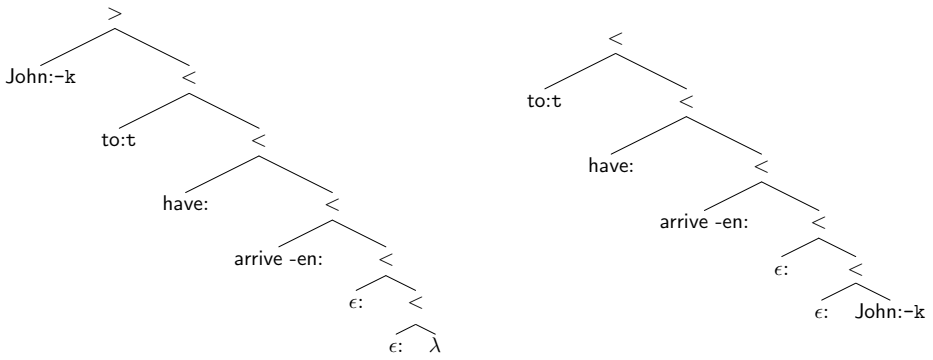


Figure 10: Two accounts of the structure of the non-finite TP

Thus, we could just as easily interpret moving expressions as successive cyclicly moving to *every* intermediate specifier between feature-driven movement positions. The essential observation, which also holds for the copy theory of movement, is that information about a moving constituent is carried along *and is therefore potentially available* to every intermediate node between where it is first introduced, and where it finally ends up.

2.2 Direct Compositionality

An adequate semantic representation, in the context of the computational theory of mind that is the mainstay of modern cognitive science, is one over which the appropriate inferential relations (of entailment, synonymy, etc) can be simply defined. These relations are often given model-theoretic counterparts by means of associating semantic representations with model-theoretic objects (e.g. sets of models in which the representation is 'true'). This allows us to give precise accounts of semantic relations between sentences without committing ourselves to a particular mode of mental symbol manipulation. Many find it desirable to 'directly interpret' either the derivation tree or the derived tree into some model theoretic object. This amounts to taking the tree (derivation or derived) as the semantic representation itself, with the 'interface map' just being the identity function. This is a logical possibility, and one which, under one reading, cannot help but be right—we can always compose the interface map with the map from semantic representations to model theoretic objects. However, there is also a sense in which this is a bold conjecture. Under this reading, the proposal is that the derivation tree will provide the right kind of structure over which to simply define the rules of

inference for the 'language of thought'. (See [28, 92, 94] for discussion, and progress on this front.)

The main idea on the syntactic side of work on the syntax-semantics interface has been that expressions may be interpreted in places in which they do not appear on the surface, but that those places in which they may be interpreted are characterizable in terms of positions through which they have moved in the course of the derivation. As we will be directly mapping our derivations into model-theoretic objects, and will therefore not have the use of derived trees which record the positions through which our objects have moved, we will need to decide upon an appropriate interpretation of our objects *as they move through each of their intermediate positions*. An extremely simplistic implementation of this idea in the context of our current assumptions is to associate a semantic value with each feature of an expression, and that as each feature is checked, its associated semantic value interacts appropriately with the semantic value associated with the feature of the expression that it checks/is checked by. Our approach bears obvious (and non-coincidental) resemblances to [20], perhaps the main difference lies in our restrictions on access to the quantifier store, which enforce that quantificational elements can take scope only in their chain positions.

The next section discusses the basic idea underlying the model-theoretic semantics for minimalist grammars detailed in § 2.4 and put to work in the domain of quantifier scope (§ 2.5) and control (§ 2.7).

2.3 Semantics in Chains

Sentences like 45 are commonly thought to be compatible with two seemingly different states of affairs.

(45) Exactly one maggot will devour more than two carcasses.

The subject wide scope reading of the above sentence has it that the set of maggots who end up eating more than two carcasses will be a singleton set. In such a situation, the discourse may be continued with an utterance of 46.

(46) John will too.

According to another interpretation, there are more than two carcasses which have a single solitary maggot worming its way through them, although there may be many more carcasses which are chock full of the little creatures, and each maggot may well be dining upon a smörgåsbord of dead entities. In this situation, an utterance of 46 would be infelicitous.

These readings are logically distinct, in the sense that neither of them entails the other. To see this, consider a situation in which there are an

equal number of maggots and carcasses, say three. If maggot one is eating all three of the carcasses, maggot two is eating carcass one, and maggot three is eating carcass three, then the subject wide scope reading of sentence 45 is true of this situation (as there is exactly one maggot (one) which is eating more than two carcasses, the others are eating just one apiece). The subject narrow scope reading is not (as only carcass three is being eaten by just one maggot). This is depicted in figure 11. If, on the other hand, we imagine

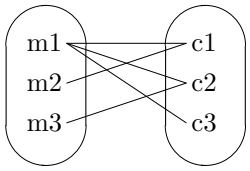


Figure 11: A model for the subject-wide scope reading of sentence 45

the same maggots and carcasses engaged in a different pattern of feeding behaviour, where maggot one is eating carcass one, maggot three is eating carcasses two and three (and maggot two is slowly starving to death), the wide scope reading is false (as no maggot is eating more than two carcasses), but the narrow scope reading is true (as all three carcasses are being eaten by a single maggot—carcass one by maggot one, and carcasses two and three by maggot three). This is depicted in figure 12.

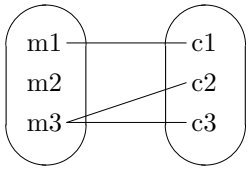
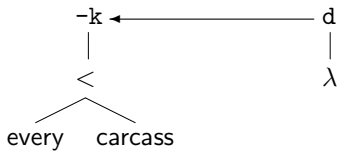


Figure 12: A model for the subject-narrow scope reading of sentence 45

Quantified noun phrases (QNPs) are usefully thought of as making two general meaning contributions to the clauses in which they appear. The first is the role they play in the clause (i.e. is *exactly one maggot* the devourer or the devouree). The second is their logical priority with respect to other elements (as Hintikka [38] calls it). For example, the two readings of sentence 45 assign the same grammatical role to the two QNPs, but invert their logical priority (with the subject being logically prior to the object in the subject-wide reading, and the object prior to the subject in the subject-narrow reading). This bipartite meaning contribution is usually represented

by means of variables (which serve to saturate the appropriate argument position) and variable binding operators (which demarcate the semantic scope of the quantifier).

Our expressions already record information about their future syntactic relationships in their features. For example, from the expression every carcass:d -k we garner that *every carcass* will be selected by another with an appropriate feature (either =d, =>d, or d=>), and will then move to a position licensing its case feature (where it will be pronounced). We might just as well have represented this information in terms of the following, 'chain'-like representation (as in [81]).



Given our discussion about the bipartite meaning contribution of QNPs, this representation is suggestive, in that to the two meaning components we wish to be expressed correspond two syntactic positions. A straightforward implementation of this intuition is to simply associate the slot-filler component of the meaning of this expression with the category feature, and the quantificational component with the licensee feature.

The intuition, then, is that to each 'link' in a chain we can associate that link's meaning contribution when incorporated into the structure. This will be a useful intuition to cultivate, as it captures the essence of our strategy—everything else is just details (where the devil resides).

In § 2.4 we spell out these details in all of their model-theoretic glory. While we have no "variables in the syntax," and more generally no levels in our syntactic theory at all, we will make use of variable assignments in the sense of Tarski [90] (and thus sentences denote sets of satisfying assignments, as in [54]). This allows us to identify a class of model theoretic objects which act as abstraction operators, leaving us with the benefits of translation into an intermediate language with variables and variable binders. This treatment of abstraction operators, while implicit in the literature, is obscured by treating (at least notationally) variable assignments as objects not on the same level as individuals, truth values, and functions between them. We follow § 2.4 with a proposal about how to implement a direct and incremental

model-theoretic translation of derivations in § 2.5. Particular to the Principles and Parameters tradition in syntax is the notion of a chain, and we show how quantifiers and quantifier scope is dealt with in our system. We implement the ideas of Hornstein [41], whereby there is no separate mechanism of Quantifier Raising (QR) by which quantifiers are moved to their scope positions. Instead, Hornstein suggests that a QNP may reconstruct in any of its chain positions for the purposes of taking scope. In § 2.6 we show how our syntax-semantics mapping assigns reasonable model theoretic objects to the raising and passive sentences from § 1.4. Finally, in § 2.7, we tackle control constructions, again adapting Hornstein's [40, 42] proposal to treat control as movement to a θ position. Such a proposal fits in quite naturally with the system developed here, and allows for an elegant treatment of the syntax and semantics of control clauses, in particular of the difference between raising and control constructions in terms of the possibility of scope reconstruction.

2.4 Model-Theoretic Glory

Our models have the following denotation domains:

1. E is the set of *entities*
2. $T = \{\mathbf{true}, \mathbf{false}\}$ is the set of *truth values*
3. $G = [\mathbb{N} \to E]$ is the set of *assignments*

Given $g, h \in G$ we 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 $j = i$). So $\approx_i$ is an equivalence relation, for every $i \in \mathbb{N}$. We write $[g]_i$ for the set $\{h : g \approx_i h\}$. We write $x \in y$ as an abbreviation for $y(x) = \mathbf{true}$.

We will call functions in the set $[G \to E]$ *individuals*, those in $[G \to T]$ *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*. We 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).

There are two kinds of individuals that are of interest to us. We will call the constant functions *names*, and for each $e \in E$ denote by $\mathbf{e}$ the function taking each $g \in G$ to e. Those individuals f that for some $i \in \mathbb{N}$, $f(g) = g_i$ for all $g \in G$ we will call *variables*, and denote with $\mathbf{x_i}$ the function taking each $g \in G$ to g_i.

It will be useful to have a name for the following determiners.

$$\begin{aligned}\mathbf{every}(A)(B)(g) = \mathbf{true} \text{ iff for every } f \in [G \to E] \\ \text{if } g \in A(f) \text{ then } g \in B(f)\end{aligned}$$

$$\begin{aligned}\mathbf{some}(A)(B)(g) = \mathbf{true} \text{ iff for some } f \in [G \to E] \\ g \in A(f) \text{ and } g \in B(f)\end{aligned}$$

We will also name the following families of functions (abstraction over x_i and the 'Geach' combinator respectively).

$$\begin{aligned}&\text{for each } i \in \mathbb{N}, \\ &\lambda_i : [G \to T] \to [G \to E] \to G \to T \\ &\lambda_i(H)(f)(g) = \mathbf{true} \text{ iff there is some } h \in H \text{ such that} \\ &h \approx_i g \text{ and } f(g) = h_i\end{aligned}$$

$$\begin{aligned}\mathtt{G} : [\alpha \to \gamma] \to [\beta \to \alpha] \to \beta \to \gamma \\ \mathtt{G}xyz = x(yz)\end{aligned}$$

An Arithmetical Example

Let's begin with a simple arithmetical language, with non-logical symbols the constant $\mathtt{0}$, and the unary function symbol $'$. In addition, our language contains the logical symbols $\mathtt{=}$, &, and ¬, as well as the punctuation symbols '(' and ')'.

The sets of terms of type $\tau \in \{\mathbf{Num}, \mathbf{Bool}\}$ are defined inductively to be the smallest sets $\mathbf{Term}_\tau$ such that

1. $\mathtt{0} \in \mathbf{Term_{Num}}$
2. $t' \in \mathbf{Term_{Num}}$, if $t \in \mathbf{Term_{Num}}$
3. $(t_1 \mathtt{=} t_2) \in \mathbf{Term_{Bool}}$, if $t_1, t_2 \in \mathbf{Term_{Num}}$
4. $\neg(\phi) \in \mathbf{Term_{Bool}}$, if $\phi \in \mathbf{Term_{Bool}}$
5. $(\phi \,\&\, \psi) \in \mathbf{Term_{Bool}}$, if $\phi, \psi \in \mathbf{Term_{Bool}}$

A model for our language is determined by a structure $\mathcal{M} = \langle E, 0, \sigma \rangle$. The interpretation function $[\![\cdot]\!]_\mathcal{M}$ is defined inductively over terms in the following manner (recall that $G = E^\mathbb{N}$ and $T = \{\mathbf{true}, \mathbf{false}\}$).

1. $[\![\mathsf{0}]\!]_{\mathcal{M}} = f : G \to E$ such that for any $g \in G$, $f(g) = 0$
2. $[\![t']\!]_{\mathcal{M}} = f : G \to E$ such that for any $g \in G$, $f(g) = \sigma([\![t]\!]_{\mathcal{M}}(g))$
3. $[\![(t_1 = t_2)]\!]_{\mathcal{M}} = \{g : [\![t_1]\!]_{\mathcal{M}}(g) = [\![t_2]\!]_{\mathcal{M}}(g)\}$
4. $[\![\neg(\phi)]\!]_{\mathcal{M}} = G - [\![\phi]\!]_{\mathcal{M}}$
5. $[\![(\phi \,\&\, \psi)]\!]_{\mathcal{M}} = [\![\phi]\!]_{\mathcal{M}} \cap [\![\psi]\!]_{\mathcal{M}}$

We determine the interpretation of the formula $(\mathsf{0}' = \mathsf{0}'') \in \mathbf{Term_{Bool}}$ as follows.

$$\begin{aligned} g \in [\![(\mathsf{0}' = \mathsf{0}'')]\!]_{\mathcal{M}} \text{ iff } & [\![\mathsf{0}']\!]_{\mathcal{M}}(g) = [\![\mathsf{0}'']\!]_{\mathcal{M}}(g) \\ \text{iff } & [\![\mathsf{0}']\!]_{\mathcal{M}}(g) = \sigma([\![\mathsf{0}']\!]_{\mathcal{M}}(g)) \\ \text{iff } & \sigma([\![\mathsf{0}]\!]_{\mathcal{M}}(g)) = \sigma(\sigma([\![\mathsf{0}]\!]_{\mathcal{M}}(g))) \\ \text{iff } & \sigma(0) = \sigma(\sigma(0)) \end{aligned}$$

Thus, $[\![(\mathsf{0}' = \mathsf{0}'')]\!]_{\mathcal{M}}$ is either G or $\emptyset$ depending upon whether $\sigma(0) = \sigma(\sigma(0))$ in $\mathcal{M}$ or not. Note that all $\phi \in \mathbf{Term_{Bool}}$ denote denote either G or $\emptyset$, and that all $t \in \mathbf{Term_{Num}}$ denote names (constant functions in $[G \to E]$).

Adding Variables and Variable Binders

Next we extend the non-logical vocabulary of our language to include a denumerably infinite set of variable symbols $\mathbf{Var} = \{\mathsf{x}_0, \mathsf{x}_1, \mathsf{x}_2, \ldots\}$. We extend the logical vocabulary with the quantifier symbol $\forall$. The following two cases are added to the definition of terms.

6. $\mathbf{Var} \subseteq \mathbf{Term_{Num}}$
7. $(\forall x)\phi \in \mathbf{Term_{Bool}}$, if $\phi \in \mathbf{Term_{Bool}}$ and $x \in \mathbf{Var}$

The definition of $[\![\cdot]\!]_{\mathcal{M}}$ needs to be extended to include these cases.

6. $[\![\mathsf{x_i}]\!]_{\mathcal{M}} = \mathbf{x_i}$
7. $[\![(\forall \mathsf{x_i})\phi]\!]_{\mathcal{M}} = \mathbf{every}(\mathtt{G}(E))(\lambda_i([\![\phi]\!]_{\mathcal{M}}))$

Note that $g \in (\mathtt{G}(E))(f)$ iff $f(g) \in E$, and therefore that

$$\begin{aligned} g \in \mathbf{every}(\mathtt{G}(E))(\lambda_i([\![\phi]\!]_{\mathcal{M}}))(g) & \\ \text{iff for every } f \in [G \to E]\, & g \in \lambda_i([\![\phi]\!]_{\mathcal{M}}))(f) \\ \text{iff for every } f \in [G \to E] \text{ there is some } & h \in [\![\phi]\!]_{\mathcal{M}} \\ \text{such that } h \approx_i g \text{ and } & f(g) = h_i \end{aligned}$$

To see better how this works, consider the interpretation of the sentence $(\mathsf{x}_0 = 0'') \in \mathbf{Term_{Bool}}$, which we can calculate as follows.

$$\begin{aligned} g \in [\![(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}} \text{ iff } & [\![\mathsf{x}_0]\!]_{\mathcal{M}}(g) = [\![0'']\!]_{\mathcal{M}}(g) \\ \text{iff } & \mathbf{x_0}(g) = \sigma(\sigma(0)) \\ \text{iff } & g_0 = \sigma(\sigma(0)) \end{aligned}$$

That is, $[\![(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}}$ is the set of all assignments which assign the value $\sigma(\sigma(0))$ to the index 0. The denotation of $(\forall \mathsf{x}_0)(\mathsf{x}_0 = 0'') \in \mathbf{Term_{Bool}}$ is given as a function of the denotation of this subformula.

$$\begin{aligned} g \in [\![(\forall \mathsf{x}_0)(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}} \text{ iff for every } f \in [G \to E] \text{ there is an } h \in [\![(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}} \\ \text{such that } h \approx_0 g \text{ and } f(g) = h_0 \\ \text{iff } f(g) = \sigma(\sigma(0)) \text{ for every } f \in [G \to E] \end{aligned}$$

That is, $[\![(\forall \mathsf{x}_0)(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}}$ will be G if $E = \{0\}$, and will be $\emptyset$ otherwise. Note that $[\![(\forall \mathsf{x}_1)(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}} = [\![(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}}$.

Application and Abstraction

Now we add another type to our language, $\mathbf{Num} \to \mathbf{Bool}$, along with the symbol $\diagup$. The set of terms of type τ is expanded as follows.

8. $(\diagup x)\phi \in \mathbf{Term_{Num \to Bool}}$ if $\phi \in \mathbf{Term_{Bool}}$ and $x \in \mathbf{Var}$
9. $(\alpha(t)) \in \mathbf{Term_{Bool}}$ if $\alpha \in \mathbf{Term_{Num \to Bool}}$ and $t \in \mathbf{Term_{Num}}$

The interpretation function is extended in the following manner so as to be defined over these new terms.

8. $[\![(\diagup \mathsf{x_i})\phi]\!]_{\mathcal{M}} = \lambda_i([\![\phi]\!]_{\mathcal{M}})$
9. $[\![(\alpha(t))]\!]_{\mathcal{M}} = [\![\alpha]\!]_{\mathcal{M}}([\![t]\!]_{\mathcal{M}})$

The sentence $(\diagup \mathsf{x}_0)(\mathsf{x}_0 = 0'')$ denotes a function which assigns to each $f \in [G \to E]$ the set $\{g : f(g) = \sigma(\sigma(0))\}$. We can calculate this as follows. For any $f \in [G \to E]$,

$$\begin{aligned} g \in [\![(\diagup \mathsf{x}_0)(\mathsf{x}_0 = 0'')]\!](f) \text{ iff } & g \in \lambda_0([\![(\mathsf{x}_0 = 0'')]\!])(f) \\ \text{iff } & \exists h \in [\![(\mathsf{x}_0 = 0'')]\!].\ h \approx_0 g \text{ and } f(g) = h_0 \\ \text{iff } & f(g) = \sigma(\sigma(0)) \end{aligned}$$

As special cases, we see that $[\![(\diagup \mathsf{x}_0)(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}}(\mathbf{0}) = [\![(0 = 0'')]\!]_{\mathcal{M}}$ and that $[\![(\diagup \mathsf{x}_0)(\mathsf{x}_0 = 0'')]\!]_{\mathcal{M}}(\mathbf{x_1}) = [\![(\mathsf{x}_1 = 0'')]\!]_{\mathcal{M}}$.

We are now ready for some of the complications of natural language.

2.5 Quantifiers and Scope

In this section, we develop an approach to the interpretation of minimalist grammars that exploits the fact that an expression may be 'active' for multiple steps of a derivation (as long as it has unchecked licensee features). The particular semantic modes of combination we explore here involve storage of semantic objects, and subsequent retrieval of these objects. In specifying the mapping from syntax to semantics we may place restrictions both on the kinds of access one has to the stored elements (e.g. whether it be pop and push, or enqueue and dequeue, or random access), as well as on when one may access them. Clearly, there are many possibilities to be explored. As regards the shape of the store, we adopt an array-like structure (stack shaped stores are explored in [47]), as this allows us to straightforwardly implement the widely-held belief in the close connection between the c-command relation in syntax and the logical priority relation in semantics. As regards access to the store, we adopt a strong position here, permitting a single retrieval from the store during each movement step in the derivation. This allows us to straightforwardly implement the widely-held belief in the close connection between chain positions and scope positions. These restrictions already provide us with a sort of 'island effect'—no expression may remain in storage after its window of 'activity' has closed.[24] As this is not a treatise on semantics, but rather a proof of concept of the possibility of directly interpreting minimalist grammars, the reader will forgive me for not providing new proposals with better coverage of the emprical data, but only of showing how old ideas may be incarnated in the formal system implemented here.

The Basics

We begin by considering simple intransitive sentences like 47 below.

(47) Some abbot died.

We can treat *some* on a par with *the*, assigning to it the syntactic type `=n d -k`. *Abbot* is, as are other common nouns, of type `n`, and *die*, like other unaccusative verbs, is of type `=d v`. The derivation starts out by merging *some* with *abbot*, which, in Heim and Kratzer's [37] system, is interpreted as the application of the denotation of the noun to the function denoted by the determiner. We adopt this idea here, allowing that function application is a

[24] Others ([20]) have considered the addition of non-logical conditions to the storage, such as the requirement that it be empty at various syntactically determined positions (such as at canonical syntactic islands, for example). This provides us with another locus of variation for our semantic theory, but this is one we shall not take advantage of.

possible mode of semantic combination associated with an instance of merger (see figure 13). To be of the right semantic type to allow for application to *some*, *abbot* should denote a function from individuals (type $[G \rightarrow E]$) to sets of assignment functions (type $[G \rightarrow T]$), a predicate. Let's call the function denoted by *abbot* **abbot**. Then the denotation of the result of merging *some* with *abbot* is **some**(**abbot**), which is itself a function from predicates to sets of assignments. The next step in the derivation of sentence 47 is to merge *die* with *some abbot*. *Die* denotes a predicate, which is of the type appropriate for application to the denotation of *some abbot*. Let's call the predicate denoted by *die* **die**. Allowing that semantic application of the denotation of the syntactic functor to the denotation of the syntactic argument is another possible mode of semantic combination associated with merger (see figure 13), the denotation of the result of merging *die* with *some abbot* is **some**(**abbot**)(**die**), which is a set of assignment functions. We ignore the semantic contribution of tense and aspect, and, for the moment, of movement. Thus, at the end of the derivation, we are left with the set of assignments **some**(**abbot**)(**die**). An assignment function g is in this set just in case there is some individual f who is both an abbot ($g \in \mathbf{abbot}(f)$) and died ($g \in \mathbf{die}(f)$).

$$[\![merge(\alpha, \beta)]\!] \rightarrow [\![\alpha]\!]([\![\beta]\!]) \quad \text{(FA)}$$
$$[\![merge(\alpha, \beta)]\!] \rightarrow [\![\beta]\!]([\![\alpha]\!]) \quad \text{(BA)}$$

Figure 13: Modes of Semantic Combination (I)

Quantifiers in Object Positions

Turning now to transitive sentences like 48 with quantificational DPs in the object position, we run into familiar problems.

(48) George shaved some abbot.

We calculate the denotation of the result of merging *some* and *abbot* as before, but now the verb, *shave*, denotes not a predicate, but a function from individuals to predicates, and thus cannot combine with the generalized quantifier denotation of *some abbot*. One approach to this problem, advocated by Keenan [46], among others, is to take the denotation of a DP to be not a function from predicates to sets of assignments, but rather a function from n+1-ary relations to n-ary relations, 'valencey reducers', as it

were. While certainly a logical possibility, and a workable one at that, this approach doesn't take advantage of the particular structure of our current syntactic theory. Instead, we adopt the 'Quantifying-In' approach put forth in [67], whereby a quantificational DP may make multiple semantic contributions in a sentence—first marking its grammatical role (with a variable), and then marking its scope (with a variable binder). Intuitively, we allow a DP, when merged, to introduce a variable, storing its normal, quantificational meaning for later insertion [20]. We will allow a stored meaning to be retrieved not ad libitum, but rather only when its associated DP moves—an incremental approach to 'reconstruction.'

To implement this intuition, we add a final possible semantic mode of composition to the merge rule; we feed a new variable to the denotation of the syntactic functor, and STORE the quantificational meaning (as shown in figure 14).

	Store	
$[\![merge(\alpha,\beta)]\!] \rightarrow [\![\alpha]\!]([\![\beta]\!])$	$store(\alpha)^\frown store(\beta)$	(FA)
$[\![merge(\alpha,\beta)]\!] \rightarrow [\![\beta]\!]([\![\alpha]\!])$	$store(\alpha)^\frown store(\beta)$	(BA)
$[\![merge(\alpha,\beta)]\!] \rightarrow [\![\alpha]\!](\mathbf{x_i})$	$store(\alpha)^\frown \mathtt{G}([\![\beta]\!])(\lambda_i)^\frown store(\beta)$	(Store)

Figure 14: Modes of Semantic Combination (II)

The calculation of the interpretation of sentence 48 proceeds as follows. First, *some* and *abbot* are merged, denoting (via function application) **some**(**abbot**). Next, we merge *shave* with *some abbot*. The only viable mode of semantic combination available to us (given the respective types of the denotations of these expressions) is our newly introduced storage mode. Thus the denotation of the verb phrase *shave some abbot* is $\mathbf{shave}(\mathbf{x_0})$ with the function $\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)$ in storage.

We next merge the active voice head ϵ::`=>V +k =d v` with our VP. We ignore any semantic effect this may have, treating the denotation of this merger as identical to the denotation of the merged VP (formally, the active voice head denotes the identity function over properties). But now again we are in trouble: we next have to move the phrase *some abbot*, which checks its case feature, rendering it syntactically inert, but we can neither leave its stored meaning in the store (because we then lose the connection we are trying to maintain between an expression's scope and its chain positions), nor combine the stored meaning with the meaning of the expression as a whole in a straightforward way (as the types still do not match). Intuitively,

what we want is for the stored meaning to be retrieved *after* the subject is merged, saturating the **shave** relation. Thus, we want the phrase *some abbot* to move again, after the subject is introduced. We implement this by adding a new licensing feature type, 'Q', and assigning the determiner *some* the type `=n d -k -q`, and the active voice head the type `=>V +k =d +q v`. Now, when we move *some abbot* to check its `-k` feature, we leave its stored meaning untouched (and thus associated with the move operation must be an 'empty' mode of semantic combination (see figure 15)). We next merge *George*, which denotes the name $\mathbf{g}$, the result of which merger denotes the set of assignments $\mathbf{shave}(\mathbf{x_0})(\mathbf{g})$, with stored $\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)$. Now when we move *some abbot* to check its `-q` feature, we apply the set of assignments $\mathbf{shave}(\mathbf{x_0})(\mathbf{g})$ to the stored $\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)$ (see figure 15), yielding

$$\begin{aligned}\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)(\mathbf{shave}(\mathbf{x_0})(\mathbf{g}))& \\ &= \mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{g}))) \\ &= \{h:\ \text{for some } f \in [G \to E]\,,\ \mathbf{g} \text{ shaved } f(h) \\ &\qquad \text{and } f(h) \text{ is an abbot}\}\end{aligned}$$

Our final modes of semantic combination are as schematized in figure 15 (for precision see appendix A.3). The $\mathcal{Q}$ in the figure represents the stored meaning of the moving constituent.

	Store	
$[\![merge(\alpha,\beta)]\!] \to [\![\alpha]\!]([\![\beta]\!])$	$store(\alpha)^\frown store(\beta)$	(FA)
$[\![merge(\alpha,\beta)]\!] \to [\![\beta]\!]([\![\alpha]\!])$	$store(\alpha)^\frown store(\beta)$	(BA)
$[\![merge(\alpha,\beta)]\!] \to [\![\alpha]\!](\mathbf{x_i})$	$store(\alpha)^\frown \mathtt{G}([\![\beta]\!])(\lambda_i)^\frown store(\beta)$	(Store)
$[\![move(\alpha)]\!] \to [\![\alpha]\!]$	$store(\alpha)$	(Id)
$[\![move(\alpha)]\!] \to \mathcal{Q}([\![\alpha]\!])$	$store(\alpha) - \mathcal{Q}$	(Retrieve)

Figure 15: Modes of Semantic Combination (III)

Before we move on to a discussion of quantifier scope interactions, a word is in order about the introduction of the Q licensing feature—which lexical items have `-q` and `+q` features? In other words, is the `+q` feature on our active voice head (newly of type `=>V +k =d +q v`) optional, or not? If we decide to make it optional, we need some way of blocking sentences like 49 below, in which a QNP in a lower clause checks its `-q` feature in the higher clause.

(49) *John thinks some abbot (that) George shaved.

This kind of movement of a DP seems to be in need of blocking anyways, as (many) quantifiers seem to be limited in their scopal positions to their closest c-commanding tensed head. We might correctly rule sentence 49 out, and thereby implement this generalization, by requiring that our tensed lexical items bear a `+q` feature (so *will*, *-s*, and *-ed* have the type `=perf +k +q t`). Given the grammaticality of sentences without quantified DPs, such as 50 below, we need to at least allow all DPs, quantificational or not, to bear a `-q` feature.

(50) John devoured George.

Again, we need to ensure that sentences like the ungrammatical below (51) are not generated.[25]

(51) *George John devoured.

There are two possibilities that suggest themselves. We might either allow moving expressions to be pronounced in positions other than the highest through which they move (i.e. covert movement), or we might decide that the active voice head does, after all, have an obligatory `+q` feature. Were we to introduce covert movement (à la [85]) we could stipulate that all movement driven by the Q feature is covert. This has a number of advantages, among them the possibility of a straightforward account of inverse linking.[26] As adding covert movement to our current system is irrelevant for the syntax-semantics interface we have developed here, we take the conservative path, and adopt the second option, that of making the `+q` feature on the active voice head obligatory. (Covert movement is discussed in chapter 3 of Kobele [48].) This makes the `-q` feature on every DP obligatory (as DPs without `-q` features will not be permitted in either subject or object positions).

Interleaving Chains

Hornstein [40, 42] proposes to do away with the standard quantifier raising (QR) operation (an operation usually conceived of as being different

[25]This word order is fine, if *George* is topicalized. While topicalization *could* be driven by the same features driving standard DP/QNP movement, here I will simply assume that it is not.

[26]As in sentences like

i. Someone in every left-of-right-of-center organization is a mole.

which has a reading (arguably the most natural one) according to which the universal quantifier *every* outscopes the existential quantifier *some*.

from normal feature-driven movement), instead allowing the scope-bearing element to be positioned by *any* movement. Given our new Q feature, introduced above, the movements of the subject and object cross, which gives rise to the possibility of inverse scope when the subject is merged without storage. We go through a derivation of the sentence below, showing how the two interpretations arise via our semantic rules.

(52) Something devoured everyone.

We take *something* and *everyone* to be typed with other DPs (and thus to have the type `d -k -q`), and to denote **some**(G(E)) and **every**(G(E)) respectively, where E is the universe of the model (ignoring the distinction between 'something' and 'someone'). This sentence has the single derivation below:[27]

1. merge(devour::`=d V`, everyone::`d -k -q`)
2. merge(ϵ::`=>V +k =d +q v`, 1)
3. move(2)
4. merge(3, something::`d -k -q`)
5. move(4)
6. merge(ϵ::`=>v prog`, 5)
7. merge(ϵ::`=>prog perf`, 6)
8. merge(-ed::`perf=> +k +q t`, 7)
9. move(8)
10. move(9)

Assuming that all lexical items other than *something*, *devour*, and *everyone* are semantically vacuous (i.e. denote the identity function over the appropriate type), the subject narrow scope reading of sentence 52 is calculated from the derivation above in the following manner (the notation α; $\boxed{\beta}$ represents

[27]The number of readings sentences have can increase exponentially with the number of quantifiers. Much work in computational semantics has investigated the question of how to compactly represent this information (leading to the development of Hole Semantics [7] and Minimal Recursion Semantics [21], among others). Here, treating with Cooper [20] the semantic scope of DPs as (partially) independent of the syntactic derivation, our derivation tree *is* an underspecified semantic representation.

the denotation α with β in storage). We simplify the notation whenever possible, writing $x(yz)$ for $\mathtt{G}xyz$.

1.	$\mathbf{devour}(\mathbf{x_0})$; $\boxed{\mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	Store
2.	$\mathbf{devour}(\mathbf{x_0})$; $\boxed{\mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	FA
3.	$\mathbf{devour}(\mathbf{x_0})$; $\boxed{\mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	Id
4.	$\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))$; $\boxed{\mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	BA
5.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))))$	Retrieve
6.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))))$	FA
7.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))))$	FA
8.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))))$	FA
9.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))))$	Id
10.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))))$	Id

We can calculate the subject wide scope reading of sentence 52 in the following way. Note that we may retrieve the stored function either in step 9 or 10.[28]

1.	$\mathbf{devour}(\mathbf{x_0})$; $\boxed{\mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	Store
2.	$\mathbf{devour}(\mathbf{x_0})$; $\boxed{\mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	FA
3.	$\mathbf{devour}(\mathbf{x_0})$; $\boxed{\mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	Id
4.	$\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1})$; $\boxed{\mathtt{G}(\mathbf{some}(\mathtt{G}(E)))(\lambda_1), \mathtt{G}(\mathbf{every}(\mathtt{G}(E)))(\lambda_0)}$	Store
5.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))$; $\boxed{\mathtt{G}(\mathbf{some}(\mathtt{G}(E)))(\lambda_1)}$	Retrieve
6.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))$; $\boxed{\mathtt{G}(\mathbf{some}(\mathtt{G}(E)))(\lambda_1)}$	FA
7.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))$; $\boxed{\mathtt{G}(\mathbf{some}(\mathtt{G}(E)))(\lambda_1)}$	FA
8.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))$; $\boxed{\mathtt{G}(\mathbf{some}(\mathtt{G}(E)))(\lambda_1)}$	FA
9.	$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))$; $\boxed{\mathtt{G}(\mathbf{some}(\mathtt{G}(E)))(\lambda_1)}$	Id
10.	$\mathbf{some}(\mathtt{G}(E))(\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))))$	Retrieve

2.6 Raising and Passive

Raising

The same possibilities that exist for non-surface scopal relationships between subjects and objects in simple transitive sentences seem to be preseved under

[28]We also stand in need of a way to guarantee that the variables introduced during the Store rule are globally new/fresh. A system of explicit substitutions influenced by the $\lambda\sigma$-calculus [1] is developed in appendix B (viewing object level substitutions as instructions to systematically permute assignment functions).

raising. For example, there is a reading of sentence 53 (53.ii) according to which the object is logically prior to the raised subject.

(53) Something seems to be devouring everyone.

i. (it seems that) there is a particular entity (Grendel, say), such that that entity is devouring everyone.

ii. (it seems that) for each person, there is some entity or other that is devouring him.

The existence of unraised equivalents to 53 in which the verb *seem* seems to be acting as a sentential operator, motivates the semantic treatment of *seem* as a function over sentence-type denotations. However, our extensional semantics doesn't allow for enough distinctions to be drawn between sentences to provide for an adequate treatment of the inferential patterns involving *seem*. The standard move to make is to adopt a richer set of truth-values; instead of the simple boolean algebra **2**, we move to the algebra $[W \to \mathbf{2}]$ of functions from an index set W (of worlds) to truth values.[29] Our useful functions (e.g. **every** and λ_i) are given the obvious reinterpretations.

$$\begin{aligned}\mathbf{every}(A)(B)(g)(w) = \mathbf{true} \text{ iff for every } f \in [G \to E] \\ \text{if } w \in A(f)(g) \text{ then } w \in B(f)(g)\end{aligned}$$

$$\begin{aligned}\lambda_i(H)(f)(g)(w) = \mathbf{true} \text{ iff there is some } h \approx_i g \text{ such that} \\ w \in H(h) \text{ and } f(g) = h_i\end{aligned}$$

We can now define sentential operators (like **necessarily**) which quantify over possible words.

$$\mathbf{necessarily}(H)(g)(w) = \mathbf{true} \text{ iff for every } v \in W,\ v \in H(g)$$

Letting *seem* denote $\mathbf{seem} : [G \to W \to T] \to G \to W \to T$, we can derive the following two readings for sentence 53.

i. $\mathbf{some}(\mathtt{G}(E))(\lambda_1(\mathbf{seem}(\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))))))$

ii. $\mathbf{seem}(\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0})))))$

Although we have argued that, in a well-defined sense, movement just is successive cyclic, clearly our current semantic modes of combination, which allow for retrieval from storage only during feature driven movements, discriminate

[29] Treating possible worlds in this manner (as just a richer set of truth values) entails that names are "rigid designators" [55] as they have no world parameter.

between feature driven and successive cyclic movements. If we wanted to allow for the possibility of QNPs to take scope in intermediate, successive cyclic positions (which has been argued against in the case of A-movement [57, 69, 71]),[30] we could simply allow retrieval to apply freely throughout the derivation, making our proposal even more similar to Cooper's [20] original one.[31] This would allow us to generate the currently ungenerable reading iii according to which the raised subject scopes over the object but beneath the raising predicate.

iii. $\mathbf{seem}(\mathbf{some}(\mathtt{G}(E))(\lambda_1(\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{devour}(\mathbf{x_0})(\mathbf{x_1}))))))$

It is worth saying again what is going on here, just to allay any possible confusion. Our *syntax* is strongly successive cyclic, as witnessed by the fact that our semantics can be tweaked so as to make use of this.[32] It is our *semantics* that either takes advantage of this successive cyclicity, or ignores it. The same fact is true of (one kind of) copying—our *syntax* 'copies.'[33] Whether we design our semantics (more generally, our interface maps) to take advantage of this or not is a separate issue.

Passive

We note that a passive sentence like 54 below is roughly synonymous with (the subject narrow scope reading of) the active 55 with an existentially quantified subject.

(54) Everyone was devoured.

(55) Something devoured everyone.

This observation leads naturally to the idea that the passive voice head existentially quantifies over the external argument of the verb. Formally, we can

[30]Indeed, these same authors would prohibit QNPs from taking scope in their first merged positions, allowing only scope taking from moved-to positions (translating into our terms). This would be simple to implement in our system; instead of allowing the modes of store and FA/BA to be in free variation, we force storage for moving elements (those which are introduced by the merge3 rule in A.2).

[31]Keeping, of course, the restriction that after an element has finished its feature-driven movements, its stored semantic contribution must have already been retrieved.

[32]This allows us to formulate in a notation-independent way just what 'successive cyclicity' means. Successive cyclicity refers to whether information about a moving object is *in principle* available at a particular point in the derivation. (Copying, then, refers to *how much* of this information is available.) Note that what we are here talking about as 'syntax' is the derivation tree.

[33]For more on this see chapter 3 of Kobele [48].

assign to the passive voice head the same denotation as we assign to the QNP *something*. The lack of an 'inverse scope' reading in 54 is a consequence of the fact that we can't store the denotation of a non-moving expression (i.e. a trivial chain).

$$[\![\texttt{-en::=>V pass}]\!] = \mathbf{some}(\mathtt{G}(E))$$

Again, assuming the semantic vacuity of the other functional heads in our lexicon, we assign to sentence 54 the denotation below.

$$\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{x_0}))))$$

As noted in § 1.4, raising and passivization feed one another, giving rise to sentences like 56 below.

(56) Everyone is expected to devour John.

We let *expect* denote $\mathbf{expect} : [G \to W \to T] \to [G \to E] \to G \to W \to T$, which combines with a proposition and an individual to yield a proposition. Not allowing for free retrieval, we generate two readings for 56, the subject-wide scope (SWS) and the subject-narrow scope (SNS) readings, as shown below.

(SWS) $\mathbf{every}(\mathtt{G}(E))(\lambda_0(\mathbf{some}(\mathtt{G}(E))(\mathbf{expect}(\mathbf{devour}(\mathbf{j})(\mathbf{x_0})))))$

(SNS) $\mathbf{some}(\mathtt{G}(E))(\mathbf{expect}(\mathbf{every}(\mathtt{G}(E))(\mathbf{devour}(\mathbf{j}))))$

2.7 Control

Alongside verbs like *seem* and *expect* we find the superficially similar *want* and *persuade*, as exemplified below.

(57) John seemed to shave an abbot.

(58) John wanted to shave an abbot.

(59) George expected John to shave an abbot.

(60) George persuaded John to shave an abbot.

As is well-known, the similarities in form between these sentences conceal a structural distinction. While sentences like 57 and 59 entail the passivized lower-clause versions 61 and 63 below, 58 and 60 do not.

(61) ($\vdash$) an abbot seemed to be shaved.

(62) ($\nvdash$) an abbot wanted to be shaved.

(63) ($\vdash$) George expected an abbot to be shaved.

(64) ($\nvdash$) George persuaded an abbot to be shaved.

Furthermore, expletive subjects are permitted in the raising clauses 65 and 67 but not in 66 or 68.

(65) It seemed to be raining.

(66) *It wanted to be raining.

(67) George expected it to be raining.

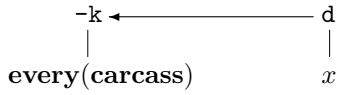
(68) *George persuaded it to be raining.

These two kinds of verbs then seem to form natural classes. We have already encountered the raising class, the second is called control. Sentences with control verbs have been analyzed as involving obligatory deletion under identity with the controller of the lower clause subject (equi-NP deletion), as involving a relationship of 'control' between the controller and an unpronounced unique-to-control element in the lower clause (PRO), as well as being syntactically identical to raising sentences, with the differences being cashed out in more semantic terms. Despite their differences, these approaches are all implementations of a common idea; control verbs bear the same kind of relation to their DP satellites as do other verbs—DPs in control clauses are semantic arguments of the control verb.

Control as Movement

Recently [40, 42, 58, 60, 70], proposals have emerged which treat control as being mediated by movement. While such a move encounters difficulties (to be discussed shortly), it offers a simple and elegant account of a surprising range of data, providing a new perspective on old facts. Furthermore, as we shall soon see, treating control as movement is easy to accomodate within our directly compositional version of minimalism.

The semantic intuition the reader was asked to cultivate in § 2.3 was illustrated with the help of the following picture.



In particular, the semantic force of this expression was broken up into two components, an argument component and a quantificational component. Now, the merge operation is always associated with argument saturation (FA, BA, and Store), not because of some mystical connection between the operation of merger and function application, but rather because we have situated the argumental force of an expression in its categorial feature, and merge is currently the only operation that deals with categorial features.[34] Intuitively, what we will do is to allow the same categorial feature to contribute its associated meaning again and again. Given that control appears to be iterable without bound (69), we decide to allow for asymmetric feature checking.

(69) George persuaded John to want to shave an abbot.

We allow categorial features to come in two flavours, `f` and `*f`, corresponding to whether asymmetric feature checking is allowed, or not. Categorial features like `*f` behave like their `f` brethren in all respects seen thus far. They differ, however, in being subject to merger without checking, and movement. Since it seems that all and only DPs are controllable, we assign these new categorial features accordingly; a 'DP' is now anything with the syntactic type `*d -k -q`. We need now three additional syntactic modes of combination. We need to be able to merge something with a starred categorial feature, and not delete it. We also need to be able to move something with a categorial feature. This control movement comes in two varieties. First, we might check the categorial feature driving the control movement. Second, we might not check the categorial feature, saving it for later control movement. We will name these operations cmerge (for 'control merge') and cmove1 and cmove2 (for 'control move'). Semantically, cmerge works as follows. First, a new variable is created, and fills the argument position our DP is cmerged into. Then we put the quantificational meaning of the DP along with a copy of the new variable into the store.

$$[\![cmerge(\alpha,\beta)]\!] \rightarrow [\![\alpha]\!](\mathbf{x_i}) \qquad store(\alpha)^\frown\langle \mathbf{x_i}, \mathtt{G}([\![\beta]\!])(\lambda_i)\rangle^\frown store(\beta)$$

Control movement saturates an argument position of a predicative expression with the semantic variable (f) associated with the moving DP in the store. If we do not eliminate the feature on this moving DP (`*d`), we leave the store unchanged, allowing for future control movements.

$$[\![cmove2(\alpha)]\!] \rightarrow [\![\alpha]\!](f) \qquad store(\alpha)$$

[34]Sometimes, such as when the subject takes narrow scope, both the argumental force and the quantificational force of an expression are realized simultaneously. This does not affect the point being made.

If we decide to eliminate the *d feature, ceasing the control movements, we may either retreive the scopal information ($\mathcal{Q}$) associated with our DP as well, giving it narrow scope, or we may leave the scopal information in the store, allowing for wider scope.

$$[\![cmove1(\alpha)]\!] \rightarrow \mathcal{Q}([\![\alpha]\!](f)) \qquad store(\alpha) - \langle f, \mathcal{Q}\rangle$$
$$[\![cmove1(\alpha)]\!] \rightarrow [\![\alpha]\!](f) \qquad (store(\alpha) - \langle f, \mathcal{Q}\rangle)^\frown \mathcal{Q}$$

We work through a derivation of sentence 70 below, which we present in the treeless form discussed in 2.1. After each step in the derivation, we exhibit the denotation of the resulting expression below it. When there is a choice, we prefer wider scope.

(70) Every barber promised George to be persuaded to shave an abbot.

We assign the object control verb *persuade* the type =t =d V, and *promise* the type =d +k =t =d +q v.[35] In a tree, these type assignments indicate that the object of *persuade* is introduced higher than its clausal complement, and that the object of *promise* is introduced lower than its clausal complement. Assuming the existence of an argument structure template, according to which arguments across different verbs occupy canonical positions, this might be taken to suggest that the DP object of *persuade* is of a different type than that of *promise*. This typing is forced upon us by our principle of immediacy, which derives something like the minimal distance principle (MDP) [78], which is a coding up of the observation that control across another DP is generally not possible. *Promise*-type verbs constitute counter-examples to a naïve version of the MDP, one in which 'across-ness' is calculated in terms of linear order in the surface string. We will come back to this shortly.

1. merge(a::=n *d -k -q, abbot::n)

   (ϵ, a, abbot) : *d -k -q

   some(**abbot**)

2. merge(shave::=d V, 1)

   (ϵ, shave, ϵ) : V, (a abbot, -k -q)

   shave($\mathbf{x_0}$), $\boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)}$

[35] The type assigned to *promise* is very nearly the composition of the standard VP type (=d V) with the active voice type (=>V +k =d +q v). In fact, it is, with an additional =t stuck in. This is done in part because *promise* doesn't passivize well, and in part because doing otherwise would necessitate revision and or duplication of functional lexical items. The most natural treatment of verbs like *promise* in our current system is this one, which attributes the awkwardness of passivization to grammatical factors.

3. merge(ϵ::=>V +k =d +q v, 2)

(ϵ, shave, ϵ) : +k =d +q v, (a abbot, -k -q)

$\mathbf{shave}(\mathbf{x_0})$, $\boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)}$

4. move(3)

(ϵ, shave, ϵ) : =d +q v, (a abbot, -q)

$\mathbf{shave}(\mathbf{x_0})$, $\boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)}$

5. merge(every::=n *d -k -q, barber::n)

(ϵ, every, barber) : *d -k -q

$\mathbf{every}(\mathbf{barber})$

6. cmerge(4, 5)

(ϵ, shave, ϵ) : +q v, (a abbot, -q), (every barber, *d -k -q)

$\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})$, $\boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0), \langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

7. move(6)

(a abbot, shave, ϵ) : v, (every barber, *d -k -q)

$\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1}))$, $\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

8. merge(ϵ::=>v prog, 7)

(ϵ, shave, a abbot) : prog, (every barber, *d -k -q)

$\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1}))$, $\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

9. merge(ϵ::=>prog perf, 8)

(ϵ, shave, a abbot) : perf, (every barber, *d -k -q)

$\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1}))$, $\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

10. merge(to::=perf t, 9)

(ϵ, to, shave a abbot) : perf, (every barber, *d -k -q)

$\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1}))$, $\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

11. merge(persuade::=t =d V, 10)

(ϵ, persuade, to shave a abbot) : =d V, (every barber, *d -k -q)

$\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1}))),$
$\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

12. cmove2(11)

(ϵ, persuade, to shave a abbot) : V, (every barber, *d -k -q)

$\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}),$
$\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

13. merge(-en::=>V pass, 12)

(ϵ, persuade -en, to shave a abbot) : pass, (every barber, *d -k -q)

$\mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1})),$
$\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

14. merge(be::=pass v, 13)

(ϵ, be, persuade -en to shave a abbot) : v, (every barber, *d -k -q)

$\mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1})),$
$\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

15. merge(ϵ::=>v prog, 14)

(ϵ, be, persuade -en to shave a abbot) : prog, (every barber, *d -k -q)

$\mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1})),$
$\boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$

16. merge(ϵ::=>prog perf, 15)

(ϵ, be, persuade -en to shave a abbot) : perf, (every barber, *d -k -q)

$$\mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1})),\ \boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$$

17. merge(to::=perf t, 16)

(ϵ, to, be persuade -en to shave a abbot) : t, (every barber, *d -k -q)

$$\mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1})),\ \boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$$

18. merge(promise::=d +k =t =d +q v, George::*d -k -q)

(ϵ, promise, ϵ) : +k =t =d +q v, (George, -k -q)

$$\mathbf{promise}(\mathbf{g})$$

19. move(18)

(ϵ, promise, ϵ) : =t =d +q v, (George, -q)

$$\mathbf{promise}(\mathbf{g})$$

20. merge(19, 17)

(ϵ, promise, to be persuade -en to shave a abbot) : =d +q v,
(George, -q), (every barber, *d -k -q)

$$\mathbf{promise}(\mathbf{g})(H),\ \boxed{\langle \mathbf{x_1}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)\rangle}$$
$$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$$

21. cmove1(20)

(ϵ, promise, to be persuade -en to shave a abbot) : +q v,
(George, -q), (every barber, -k -q)

$$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}),\ \boxed{\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$$
$$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$$

22. move(21)

(George, promise, to be persuade -en to shave a abbot) : v,
(every barber, -k -q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$
$H = \mathbf{some}(\texttt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

23. merge(ϵ::=>v prog, 22)

(ϵ, promise, George to be persuade -en to shave a abbot) : prog,
(every barber, -k -q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$
$H = \mathbf{some}(\texttt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

24. merge(ϵ::=>prog perf, 23)

(ϵ, promise, George to be persuade -en to shave a abbot) : perf,
(every barber, -k -q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$
$H = \mathbf{some}(\texttt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

25. merge(-ed::perf=> +k +q t, 24)

(ϵ, ϵ, promise -ed George to be persuade -en to shave a abbot) : +k +q t,
(every barber, -k -q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$
$H = \mathbf{some}(\texttt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

26. move(25)

(ϵ, ϵ, promise -ed George to be persuade -en to shave a abbot) : +q t,
(every barber, -q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$
$H = \mathbf{some}(\texttt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

27. move(26)

(every barber, ϵ, promise -ed George to be persuade -en to shave a abbot) : t

$\mathbf{every}(\mathbf{barber})(\lambda_1(\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}))$
$H = \mathbf{some}(\texttt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

2.8 Reflections on Control

We have assigned *promise* the type `=d +k =t =d +q v`, blocking object control by virtue of the fact that the object argument is selected for before the clausal complement housing the controlling DP is merged (and thus we arrive at the same broad clausal architecture suggested in Larson [56]). This aspect of the typing (that arguments that cannot be controlled must be selected before the introduction of the controller) is forced upon us by the architecture of our system, in particular, by our principle of immediacy. Here we will delve into the rationale for the rest of this type. It will turn out that this type is literally forced upon us by our system, there being no other option given the patterns of grammaticality and ungrammaticality in the data. Although it is widely known that subject control verbs resist passivization (cf. 71, 72), they allow for passivization when their clausal complement is finite (cf. 73, 74).

(71) George promised John to arrive on time.

(72) *John was promised to arrive on time.

(73) Mary promised John that George would arrive on time.

(74) John was promised that George would arrive on time.

We will show how to derive these facts. In so doing we will see that our principle of immediacy will need to be sharpened, taking on somewhat of a counterfactual flavour (though formally there is no counterfactuality involved; see appendix A.2). To fully appreciate the analysis and the system we have developed, it is instructive to begin by considering why we cannot assign *promise* the simple type `=d =t V` (the 'mirror image' of *persuade*'s `=t =d V`).

The type of *promise*

Given that 71 above means that George is supposed to arrive on time, and not John, (i.e. that the subject of the matrix clause and not the object is the controller), our principle of immediacy forces us to make the object position of *promise* inaccessible to elements within the subordinate clause (by ordering the `=d` feature that introduces the object before the `=t` feature that introduces the subordinate clause with all of its moving bits and pieces). We might be tempted by the promise of an elegant account of the contrast between object control verbs (of type `=t =d V`) and subject control verbs, and assign the type `=d =t V` to subject control verbs. While we can clearly

still derive the standard subject control constructions (such as 71), we now overgenerate wildly, predicting the existence of control through finite clauses.

1. *to arrive every barber*

   (ϵ, to, arrive) : t, (every barber, *d -k -q)

   $\mathbf{arrive}(\mathbf{x_0})$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

2. merge(promise::=d =t V, George::*d -k -q)

   (ϵ, promise, ϵ) : =t V, (George, -k -q)

   $\mathbf{promise}(\mathbf{g})$

3. merge(2, 1)

   (to arrive, promise, ϵ) : V, (George, -k -q), (every barber, *d -k -q)

   $\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0}))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

4. merge(-en::=>V pass, 3)

   (ϵ, promise -en, to arrive) : pass, (George, -k -q),
   (every barber, *d -k -q)

   $\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0})))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

5. merge(be::=pass v, 4)

   (ϵ, be, promise -en to arrive) : v, (George, -k -q),
   (every barber, *d -k -q)

   $\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0})))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

6. merge(ϵ::=>v prog, 5)

   (ϵ, be, promise -en to arrive) : prog, (George, -k -q),
   (every barber, *d -k -q)

   $\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0})))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

7. merge(ϵ::=>prog perf, 6)

   (ϵ, be, promise -en to arrive) : perf, (George, -k -q),
   (every barber, *d -k -q)

   $\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0})))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

8. merge(will::=perf +k +q t, 7)

   (ϵ, will, be promise -en to arrive) : +k +q t, (George, -k -q),
   (every barber, *d -k -q)

   $\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0})))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

9. move(8)

   (ϵ, will, be promise -en to arrive) : +q t, (George, -q),
   (every barber, *d -k -q)

   $\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0})))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

10. move(9)

    (George, will, be promise -en to arrive) : t, (every barber, *d -k -q)

    $\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0})))$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

As the expression derived in 10 is syntactically identical to the infinitival control clause we began with (in 1), they of necessity have identical distributions. Therefore, we predict erroneously the existence of ungrammatical form-meaning pairings like in 76 alongside the grammatical 75.

(75) Every barber wanted to arrive.

$$\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{want}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_0})))$$

(76) *Every barber wanted (that) George will be promised to arrive.

$$\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{want}(\mathbf{some}(\mathtt{G}(E))(\mathbf{promise}(\mathbf{g})(\mathbf{arrive}(\mathbf{x_0}))))(\mathbf{x_0})))$$

The problem is due to the fact that our only locality condition (the principle of immediacy) is a relativistic one, in the sense that it doesn't care about 'absolute distances', but only about intervention. By passivizing the verb phrase in step 4, we eliminate the subject position we want the controller to control from, thereby allowing the controller to float up through the clause. As *promise* (and subject control verbs in general) don't passivize well, an obvious fix is to simply prohibit *promise* from combining with the passive voice. Our type assignment to *promise* is a way of requiring that *promise* only combine with the active voice—putting `=d =t V` and `=>V +k =d +q v` together we get `=d =t +k =d +q v`, which, but for the order of the `=t` and `+k` features, is precisely the type we have assigned. Justifying this difference, we will discover a broader set of problems, which will lead us to a better understanding of the principle of immediacy.

The Immediacy of Syntax

Given that our merge and cmerge rules overlap in their application, we have no simple way to 'force' a non-finite clause to be either a raising, or a control structure. Thus, to every non-finite clause, there corresponds both a raising, and a control analysis (as in 77).

(77) *to arrive every barber*

raising (ϵ, to, arrive):t, (every barber, -k -q)

$\mathbf{arrive}(\mathbf{x_0})$, $\boxed{\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)}$

control (ϵ, to, arrive):t, (every barber, *d -k -q)

$\mathbf{arrive}(\mathbf{x_0})$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)\rangle}$

Likewise, there are two possibilities for combining *promise* with its DP object (as in 78).

(78) *promise some abbot*

merge (ϵ, promise, ϵ):=t +k =d +q v, (some abbot, -k -q)

$\mathbf{promise}(\mathbf{x_0})$, $\boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)}$

cmerge (ϵ, promise, ϵ):=t +k =d +q v, (some abbot, *d -k -q)

$\mathbf{promise}(\mathbf{x_0})$, $\boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_0)\rangle}$

There are thus four logical possibilities for combining *promise some abbot* with *to arrive every barber*, two of which are ruled out straightaway by our principle of immediacy (merge + raising and cmerge + control). Of the remaining two, the merge + control option converges, and nets us the familiar subject control reading. However, the cmerge + raising option also converges, and yields (among others) the following monstrosity, which has the paraphrase in 80.

(79) *Some abbot promised every barber to arrive.

$$\mathbf{some}(\mathbf{abbot})(\lambda_1(\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{promise}(\mathbf{x_1})(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})))))$$

(80) Some abbot promised himself that every barber would arrive.

By inverting the order of the `=t` and `+k` features in the type assignment to *promise* (from `=d =t +k =d +q v` to `=d +k =t =d +q v`), we force the object of *promise* to combine via standard merger, ruling out the deviant cmerge + raising possibility.

While this fixes the problem with *promise*, it is symptomatic of a more general malaise. If we combine *persuade* with a raising infinitival complement, we can derive a similar horror.

1. merge(persuade::`=t =d V`, *raising*)

   (ϵ, persuade, to arrive) : `=d V`, (every barber, `-k -q`)

   $\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))$, $\boxed{\texttt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0)}$

2. *some abbot*

   (ϵ, some, abbot) : `*d -k -q`

   $\mathbf{some}(\mathbf{abbot})$

3. cmerge(1, 2)

   (ϵ, persuade, to arrive) : `V`, (every barber, `-k -q`), (some abbot, `*d -k -q`)

   $\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})$, $\boxed{\begin{array}{l}\texttt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0),\\ \langle \mathbf{x_1}, \texttt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)\rangle\end{array}}$

4. merge(ϵ::=>V +k =d +q v, 3)

(ϵ, persuade, to arrive) : +k =d +q v, (every barber, -k -q),
(some abbot, *d -k -q)

$$\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1}),\ \boxed{\begin{array}{l}\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0),\\ \langle \mathbf{x_1}, \mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)\rangle\end{array}}$$

5. move(4)

(ϵ, persuade, to arrive) : =d +q v, (every barber, -q),
(some abbot, *d -k -q)

$$\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1}),\ \boxed{\begin{array}{l}\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0),\\ \langle \mathbf{x_1}, \mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)\rangle\end{array}}$$

6. cmove1(5)

(ϵ, persuade, to arrive) : +q v, (every barber, -q),
(some abbot, -k -q)

$$\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})(\mathbf{x_1}),\ \boxed{\begin{array}{l}\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_0),\\ \mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)\end{array}}$$

7. move(6)

(every barber, persuade, to arrive) : v, (some abbot, -k -q)

$$\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})(\mathbf{x_1})),\ \boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)}$$

8. merge(ϵ::=>v prog, 7)

(ϵ, persuade, every barber to arrive) : prog, (some abbot, -k -q)

$$\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})(\mathbf{x_1})),\ \boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)}$$

9. merge(ϵ::=>prog perf, 8)

(ϵ, persuade, every barber to arrive) : perf, (some abbot, -k -q)

$$\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})(\mathbf{x_1})),\ \boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)}$$

10. merge(will::=perf +k +q t, 9)

(ϵ, will, persuade every barber to arrive) : +k +q t, (some abbot, -k -q)

$$\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})(\mathbf{x_1})),\ \boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)}$$

11. move(10)

(ϵ, will, persuade every barber to arrive) : +q t, (some abbot, -q)

$$\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})(\mathbf{x_1})),\ \boxed{\mathtt{G}(\mathbf{some}(\mathbf{abbot}))(\lambda_1)}$$

12. move(11)

(some abbot, will, persuade every barber to arrive) : t

$$\mathbf{some}(\mathbf{abbot})(\lambda_1(\mathbf{every}(\mathbf{barber})(\lambda_0(\mathbf{persuade}(\mathbf{arrive}(\mathbf{x_0}))(\mathbf{x_1})(\mathbf{x_1})))$$

Although we cannot discriminate syntactically between raising and control infinitivials, we need somehow to block *persuade* (and *promise*) from merging with a raising clause. Looking at the deviant derivations, each has a point in which a control-merged DP (*d -k -q) coexists with a merged DP (-k -q). If we can block such a state of affairs, we will have solved our problem. The principle of immediacy states that a moving expression must check its features as soon as possible. Although at step 5 of the previous derivation, the moving subexpression *some abbot* does not have an accessible -k feature, it *would* have, had it been merged instead of cmerged. We can think of cmerging as a sort of wager—an expression is free to be cmerged, just as long as in so doing it doesn't end up having lost an opportunity to check its licensee features. This allows us to give a teleological slant to the principle

of immediacy: an expression wants to check its licensee features as soon as possible. While it cannot influence the course of the derivation on a global scale (like, demanding that another expression be merged or not), it can on a local scale (by choosing to be merged or cmerged). If it makes a decision that ends up having cost it the chance to check a licensee feature, the derivation crashes. Although this way of understanding the principle of immediacy has a definite 'transderivational economy' flavour, it is in fact locally evaluable, and is formally of the same complexity as our previous understanding of it.[36]

Now, finally, we are in a position to see that the type `=d +k =t =d +q v` is the *only* such that we could assign to *promise*. The 'uninverted' type `=d =t +k =d +q v` would make the lexical item useless; the principle of immediacy would rule out both the bad cmerge + raising option, in addition to the good merge + control option, as both would involve subexpressions with the same first licensee feature.

Passivization of Subject Control Verbs

Thus far we are able to derive the grammaticality patterns in 81 and 82 on the one hand, and 83 and 84 on the other.

(81) John promised George to shave an abbot.

(82) *George was promised to shave an abbot.

(83) John persuaded George to shave an abbot.

(84) George was persuaded to shave an abbot.

However, the patterns in the below still remain unaccounted for. In particular, subject control verbs with a single DP argument exceptionally permit passivization, which eliminates all selected DPs in the matrix clause.

(85) John promised George that every barber had shaved an abbot.

(86) George was promised that every barber had shaved an abbot.

(87) George hoped to shave an abbot.

(88) *George was hoped to shave an abbot.

[36]Formally, this amounts to requiring that

1. no two subexpressions may have the same first feature, and
2. no two subexpressions may have the same first *licensee* feature

See appendix A.2.

(89) George hoped that every barber had shaved an abbot.

(90) It was hoped that every barber had shaved an abbot.

As we were able to account for the distinction between 81 and 82 above only by stipulating that *promise* does not combine with the passive voice, it seems unlikely that we will be able to find a simple extension to deal with the examples 85 and 86. However, it seems that subject control verbs do indeed permit passivization, but only when their complement clause is finite. Currently, we have no way of expressing the difference between finite and non-finite clauses in a way that allows for the simple expression of this generalization (as currently both finite and non-finite clauses are of category `t`). Accordingly, we make a categorial distinction between finite and non-finite clauses, assigning to finite clauses the special category `s`.[37] As some expressions select for clausal complements, irrespective of their tensedness, we express that finite clauses are also clauses with the lexical item[38]

that :: =s t

The only change that this requires is the substitution of the category `s` for the category `t` in our tensed lexical items (i.e. *will*, *-s*, and *-ed*). We assign to *promise* the additional type `=d =s V`, the near mirror-image of the type of *persuade*. We then assign to *hope* (and other intransitive subject control verbs like *want*, and *expect*) the type `=t =d v`, which allows for the derivation of sentences 87 and 89, and correctly rules out 88. To allow for the ability of *hope* to passivize when it takes a finite clausal complement, we assign to it the additional type `=s V` (which allows for 90 while still correctly ruling out an ECM variant of 89). To intransitive raising to object verbs like *expect* and *want*, we assign the minimally different type `=t V`, which allows for both ECM as well as sentences like 90. Although it may seem less than maximally elegant to assign two types to intransitive subject control verbs (such as *hope* and *expect*), the very fact that there exist among the intransitive subject control verbs, some which allow for object control (and passivization with non-finite complements), and the rest which don't (and don't), seems to be a brute fact, underivable from anything else. Our complete lexicon is given in figure 16. Although our type system is not strong enough to premit us to

[37]This allows us now to discriminate between sentences i and ii

i. It rained.

ii. *To rain.

[38]We might just as well have given this lexical item a phonetically null exponent. However, its distribution coincides with the distribution of the word "that" to a fairly large degree, which is suggestive.

will::=perf +k +q s	have::=en perf	be::=ing prog
-s::perf=> +k +q s	-en::=>prog en	-ing::=>v ing
-ed::perf=> +k +q s	ϵ::=>prog perf	ϵ::=>v prog
to::=perf t	be::=pass v	ϵ::=>V +k =d +q v
that::=s t	-en::=>V pass	
arrive::=d v	devour::=d V	
	shave::=d V	
seem::=t v	expect::=t V	expect::=t =d v
	want::=t V	want::=t =d v
	hope::=s V	hope::=t =d v
persuade::=t =d V		
promise::=d =s V	promise::=d +k =t =d +q v	
ϵ::=>v =z v	it::z -k -q	
George::*d -k -q	the::=n *d -k -q	ointment::n
John::*d -k -q	every::=n *d -k -q	abbot::n
Mary::*d -k -q	some::=n *d -k -q	barber::n

Figure 16: A Grammar for English A-movement

derive the full range of behaviour of each word from a single type assignment, note that it is precisely the subject control verbs which require multiple types. Given that it is precisely subject control verbs that children have difficulty acquiring, and that successful theories of grammatical inference can be built around assumptions about the number of types assigned to words [3, 44], this fact is tantalizingly suggestive!

3 Summary

In this chapter, we have introduced the version of minimalism (minimalist grammars [22, 27, 29, 36, 64–66, 81–87]) we will take as our background theory in this dissertation. We have made quite meagre assumptions, which we have tried to justify during the course of this introduction. Perhaps the most important such is the assumption that grammatical operations are resource sensitive, and that the resources which drive them are structured in

a simple way (as a unary tree). This allows us to formulate a well-formedness condition we have called the principle of immediacy, from which we can derive both the ban on super-raising, as well as the minimal distance principle.

We have shown how all non-interface levels can be eliminated from syntax, and the mappings to form and meaning incrementally computed in the course of the derivation. We have presented a compositional semantics which takes advantage of the fact that our syntax allows expressions to be multiply connected to others. We have demonstrated that lexical items are to chains as the acorn is to the oak tree, and that our syntax is inherently successive cyclic, and shown how to extend our semantics to take advantage of this fact.

Hornstein treats reflexivization in English as an instance of control movement. Clearly, as we currently block such 'too-local' movement (in the sense of Grohmann [33]) by our principle of immediacy (together with our lexical type assignments), there is no necessary connection between treating control as mediated by movement, and treating reflexivization as a species of control.

I have said nothing about non-obligatory control, or obligatory control out of adjuncts. (This latter is related to the fact that I have said nothing about adjuncts.) Indisputably, our characterization of the competence of a native speaker of English will not be complete until the form and interpretation of the class of sentences referred to with the terms 'non-obligatory control' and 'adjunct control' is specified. Insofar as other approaches to control deal with more data than does this one, they are empirically more adequate.[39] As other approaches to control are at least twenty years more established than the movement approach, it would be surprising were this upstart not empirically deficient in comparison. We should not require that novel approaches to a phenomenon (much less a novel 'slicing of the pie') be as far reaching and as empirically adequate as the orthodoxy (see e.g. [26] for discussion). Nor should we be reluctant to entertain multiple (even incompatible) perspectives on a particular range of data, as this often highlights strengths and weaknesses of each (see [5] for an interesting comparison of CCG and minimalist approaches to scope).

Throughout this chapter, we have been agnostic about the contents of the positions from which movement occurs, choosing to write a 'λ' as something of a 'catch-all.' It is generally accepted, however, that, in many cases, the source position of movement needs to be structured, and in ways that reflect the object that thence moved. It is common, in fact, to regard what we have been representing as λ as a copy of the moved element. In the next chapter, we explore two perspectives on how this might be achieved, what Chomsky [13] calls (copying via) 'internal merge' and 'external merge.' These two

[39]Insofar as 'empirically more adequate' just means 'deals with more data'.

positions on copying can be fruitfully viewed in terms of *where* they take the structure copied to be. In particular, 'internal merge' takes copying to be of the derived tree, while 'external merge' takes copying to be of the derivation itself. We take this up in the next chapter.

This chapter's appendices are organized as follows. In appendix A a formal foundation for the syntactic theory appealed to in this chapter is provided. In appendix A.1, grammatical operations are defined over labelled binary ordered trees. Because the trees are binary, I have taken the liberty of defining ordered trees so as to take advantage of this additional structure. Appendix A.2 gives a precise characterization of the 'treeless' version of minimalist grammars used in § 2.1. (The control-specific operations are included here, as they are used later in that section.) The proof that minimalist grammars over trees are equivalent in weak generative capacity to multiple context-free grammars (MCFGs, [79]) and thus to these treeless variants is presented in [36, 64, 66], and will not be reproduced here. Next (A.3) comes the formal definition of the semantic operations appealed to in § 2.2, as well as a brief discussion of 'direct compositionality'. In appendix B, I set myself the problem of making the grammar do the work of selecting appropriate variables (so as to avoid accidental capture). This is similar to the problem of making substitutions explicit in the computer science literature (see e.g. [1]).

A Definitions

A.1 Minimalist Grammars on Trees

A.1.1 Trees

Given a structure $\tau = \langle N_\tau, \lhd_\tau \rangle$ where N_τ is a finite set and $\lhd_\tau \subseteq N_\tau \times N_\tau$, we say, for $r, s \in N_\tau$, that r is a parent of s (equivalently, s is a child of r) if $r \lhd_\tau s$, that r is a leaf if r has no children, and that r is a root if r has no parents. Nodes $r, s \in N_\tau$ are siblings if they share a parent. For nodes $r, s \in N_\tau$, we say that r dominates s, if $r \lhd_\tau^* s$, where $\lhd_\tau^*$ is the reflexive transitive closure of $\lhd_\tau$. τ is an unordered tree if there is exactly one root, every node has at most one parent, and the root dominates every node. Tree τ is binary if every parent has exactly two children. An asymmetric relation $R \subseteq N_\tau \times N_\tau$ orders binary τ if for any two nodes $r, s \in N_\tau$, r and s are related by R iff they are siblings.

Operations on Ordered Trees Given two binary trees ρ, σ (such that N_ρ and N_σ are disjoint) which are ordered by R and S respectively, we denote

by $[_T\ \rho\ \sigma]$ the binary tree τ ordered by T, where, denoting with r and s the roots of ρ and σ respectively,

1. $N_\tau = N_\rho \cup N_\sigma \cup \{t\}$, where t is some object not in N_ρ or N_σ
2. $\lhd_\tau = \lhd_\rho \cup \lhd_\sigma \cup \{\langle t, r\rangle, \langle t, s\rangle\}$
3. $T = R \cup S \cup \{\langle r, s\rangle\}$

Given a binary tree τ ordered by R, and $t \in N_\tau$, we define $t/N_\tau := \{r \in N_\tau : t \lhd_\tau^* r\}$ and $N_\tau/t := \{r \in N_\tau : \neg(t \lhd_\tau^* r)\}$ to be a partitioning of N_τ into nodes dominated by t and nodes not dominated by t, respectively. t/τ, the subtree of τ rooted at t, is $\langle t/N_\tau, \lhd_{t/\tau}\rangle$, where $\lhd_{t/\tau}$ is the restriction of $\lhd_\tau$ to t/N_τ. t/tau is ordered by the restriction of R to $N_{t/\tau}$. The result of replacing in τ the subtree rooted at t with a leaf $\ell \notin N_\tau$ is $\tau/t := \langle N_{\tau/t}, \lhd_{\tau/t}\rangle$, where $N_{\tau/t} := N_\tau/t \cup \{\ell\}$ and $r \lhd_{\tau/t} s$ iff either $r, s \in N_{\tau/t}$ and $r \lhd_\tau s$, or $s = \ell$ and $r \lhd_\tau t$. τ/t is ordered by $R_{\tau/t}$, where $rR_{\tau/t}s$ iff $r, s \in N_\tau$ and rRs, or $s = \ell$ and rRt.

Functions over Ordered Trees Let τ be a binary tree ordered by R. We define $head_R : N_\tau \to N_\tau$ and $yield_R : N_\tau \to N_\tau^*$ as follows.

$$head_R(t) := \begin{cases} t & \text{if } t \text{ is a leaf} \\ head_R(r) & \text{otherwise, where } t \lhd_\tau r, s \text{ and } rRs \end{cases}$$

$$yield_R(t) := \begin{cases} \langle t\rangle & \text{if } t \text{ is a leaf} \\ yield_R(r)^\frown yield_R(s) & \text{otherwise, where } t \lhd_\tau r, s \text{ and } rRs \end{cases}$$

Given a node $t \in N_\tau$, we denote by $proj_R(t)$ the set of nodes that have the same head as t with respect to R ($proj_R(t) := \{t' : head_R(t) = head_R(t')\}$). Note that $proj_R(\cdot)$ induces an equivalence relation over N_τ, where each block $proj_R(t)$ is totally ordered by $\lhd_\tau^*$. A node $t \in N_\tau$ is a *maximal projection* just in case t is the least element of $proj_R(t)$ with respect to $\lhd_\tau^*$ (i.e. the unique $r \in proj_R(t)$, such that for any $s \in proj_R(t)$, $r \lhd_\tau^* s$).

A.1.2 Labels

Let Σ be a finite alphabet. The set of labels for a minimalist grammar over Σ is determined by a finite set **sel** of selection feature types, and a disjoint

finite set **lic** of licensing feature types. The set **Syn** of syntactic features is given by

$$\mathbf{Syn} := \mathbf{selector} \cup \mathbf{selectee} \cup \mathbf{licensor} \cup \mathbf{licensee}$$

where

$$\begin{array}{ll} \mathbf{licensor} := \{\texttt{+f} : f \in \mathbf{lic}\} & \mathbf{selector} := \{\texttt{=f}, \texttt{=>f}, \texttt{f=>} \ : f \in \mathbf{sel}\} \\ \mathbf{licensee} := \{\texttt{-f} : f \in \mathbf{lic}\} \quad \text{and} & \mathbf{selectee} := \{\texttt{f} : f \in \mathbf{sel}\} \end{array}$$

Features $f, f' \in \mathbf{Syn}$ *match* just in case one of the following conditions obtain

1. for some $s \in \mathbf{sel}$, one of f and f' is `s` and the other is one of `=s`, `=>s`, or `s=>` ,
2. for some $l \in \mathbf{lic}$, one of f and f' is `-l` and the other is `+l`.

The set of labels is $\Sigma^* \times \mathbf{Syn}^*$. We denote with λ the label $\langle \epsilon, \epsilon \rangle$.

A.1.3 Expressions

Given a set L of labels, a minimalist expression (over L) is a five-tuple $\langle N, \lhd, \prec, <, \mu \rangle$ such that $\langle N, \lhd, \prec, < \rangle$ is a binary tree ordered by the two relations, $\prec$ and $<$ (linear precedence and projection, respectively), together with a partial function $\mu : N \to L$ which assigns labels to the leaves of $\langle N, \lhd \rangle$. An expression is simple if its underlying tree is a single node, and is complex otherwise. The head $hd(e)$ of an expression e is the $head_<$ of the root of its underlying tree. An expression e begins with feature f just in case the label of $hd(e)$ is $\langle \sigma, f\delta \rangle$, for $\sigma \in \Sigma^*$ and $\delta \in \mathbf{Syn}^*$. With respect to some $f \in \mathbf{Syn}$, an expression e is said to be complete just in case the only syntactic feature of e is f, and no subtree of e has any syntactic features. Given an expression $e = \langle N, \lhd, \prec, <, \mu \rangle$, we say "$e'$ is like e except that the label of the head of e' is ℓ" to describe the expression $e' = \langle N, \lhd, \prec, <, \mu' \rangle$, where

$$\mu'(t) := \begin{cases} \ell & \text{if } t = hd(e) \\ \mu(t) & \text{otherwise} \end{cases}$$

We take $Exp(L)$ to denote the set of all expressions over L.

The Linear Correspondence Axiom Kayne's [45] linear correspondence axiom (LCA) can be understood as demanding that $\prec$ be definable in terms of $<$

$$\forall t, t' \in N. \text{ if } t < t', \text{ then } t \prec t' \text{ iff } t \text{ is a leaf} \qquad \text{(LCA)}$$

The LCA as given above is admissible in the system of [85]. We adopt it explicitly here, as it allows for a slight simplification in our statement of the generating functions. Accordingly, we suppress reference to the relation $\prec$ throughout the following.

Operations on Expressions The operations on trees defined in § A.1.1 are extended over expressions in the obvious manner:

- $[_< \langle N_c, \lhd_c, <_c, \mu_c\rangle \ \langle N_d, \lhd_d, <_d, \mu_d\rangle \ ]$ is the expression consisting of the tree $[_< \langle N_c, \lhd_c, <_c\rangle \ \langle N_d, \lhd_d, <_d\rangle \ ]$ and the labeling function $\mu := \mu_c \cup \mu_d$.
- Given $e = \langle N_e, \lhd_e, <_e, \mu_e\rangle$, for any $t \in N_e$,
  - t/e is the expression consisting of the tree $t/\langle N_e, \lhd_e, <_e\rangle$, and the labeling function which is the restriction of μ to $N_{t/e}$
  - e/t is the expression consisting of the tree $\langle N_e, \lhd_e, <_e\rangle/t$ and the labeling function $\mu_{e/t}$, where

$$\mu_{e/t}(r) := \textbf{if } r = \ell \textbf{ then } \lambda \textbf{ else } \mu(r)$$

- Given an expression $e = \langle N_e, \lhd_e, <_e, \mu_e\rangle$, the yield of e, $Yield(e)$, is defined to be the concatenation of the first components of the labels of the leaves of its underlying tree, in the order given by the linear precedence relation $\prec$. That is,

$$Yield(e) := (\pi_1 \circ \mu_e)(yield_\prec(\langle N_e, \lhd_e, <_e\rangle))$$

  where π_1 is the first projection function, and $\pi_1 \circ \mu_e$ is extended over sequences of pairs in the obvious way.

A.1.4 Merge and Move

Merge The domain of the merge operation is the set of pairs $\langle e_0, e_1\rangle$, where e_0 begins with $f \in$ **selector**, and e_1 begins with matching $g \in$ **selectee**. For $e_0, e_1 \in Dom(merge)$, with $\langle \sigma_0, f\delta_0\rangle$ the label of the head of e_0 and $\langle \sigma_1, \mathtt{x}\delta_1\rangle$ the label of the head of e_1,

$$merge(e_0, e_1) := [_< e'_0 \ e'_1 \ ]$$

where e'_0 and e'_1 are just like e_0 and e_1, except that

if $f =$ `=x` then the head of e'_0 is $\langle \sigma_0, \delta_0\rangle$ and that of e'_1, $\langle \sigma_1, \delta_1\rangle$

if $f =$ `=>x` then the head of e'_0 is $\langle \sigma_1\sigma_0, \delta_0\rangle$ and that of e'_1, $\langle \epsilon, \delta_1\rangle$

if $f =$ `x=>` then the head of e'_0 is $\langle \epsilon, \delta_0\rangle$ and that of e'_1, $\langle \sigma_1\sigma_0, \delta_1\rangle$

Move An expression e is in the domain of the move operation just in case e begins with `+x` and there is exactly one maximal projection $t \in N_e$ such that the head of t begins with matching `-x`. For $e \in Dom(move)$, with $t \in N_e$ the unique maximal projection such that the head of t begins with the matching licensee feature,

$$move(e) := [_{<}\ e_0\ e_1\ ]$$

where e_0 and e_1 are just like e/t and t/e respectively, except that the first feature of each of these expressions has been deleted.

A.1.5 Minimalist Grammars

A minimalist grammar (MG) is a five-tuple $G = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \mathsf{c} \rangle$, where $\Sigma, \mathbf{lic}$, and $\mathbf{sel}$ are finite sets (the alphabet, licensing feature types, and selection feature types respectively) which together determine a set of labels L, Lex is a finite set of simple expressions over L, and $\mathsf{c} \in \mathbf{selectee}$ is the designated type of complete expressions.

An expression e is generated by a minimalist grammar G just in case $e \in CL^n(G)$ for some $n \in \mathbb{N}$, where

$$\begin{aligned} CL^0(G) :=& Lex \\ CL^{k+1}(G) :=& CL^k(G) \cup \{move(e) : e \in Dom(move) \cap CL^k(G)\} \\ & \cup \{merge(e_0, e_1) : \langle e_0, e_1 \rangle \in Dom(merge) \cap CL^k(G) \times CL^k(G)\} \end{aligned}$$

The string language of a minimalist grammar G is

$$L(G) := \{Yield(e) : e \in \bigcup_{n=0} CL^n(G) \text{ and } e \text{ is complete}\}$$

A.2 Minimalist Grammars without Trees

The trees of appendix A.1 provide for more distinctions between expressions than are strictly necessary to determine whether a string is generated by a minimalist grammar (as follows from a result of Michaelis [66]). Instead of trees, it has become standard to view minimalist expressions as sequences of categorized strings. Given a minimalist grammar $G = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \mathsf{c} \rangle$, where

1. Σ is a finite, non-empty set
2. $\mathbf{lic}, \mathbf{sel}$ are the licensing and selection feature types, respectively, which together determine a set **Syn** of syntactic features as follows.

(a) for $l \in \mathbf{lic}$, $\texttt{+l}, \texttt{-l} \in \mathbf{Syn}$

(b) for $s \in \mathbf{sel}$, $\texttt{=s}, \texttt{=>s}, \texttt{s=>}, \texttt{s}, \texttt{*s} \in \mathbf{Syn}$

(c) nothing else is in $\mathbf{Syn}$

3. $\texttt{c} \in \mathbf{selectee}$ is the designated type of complete expressions

4. *Lex* is a finite subset of $\{\epsilon\} \times \Sigma \times \{\epsilon\} \times \{::\} \times \mathbf{Syn}^*$

we define a minimalist expression to be a pair $\langle i, N \rangle$, where $i \in \mathcal{I} := \Sigma^* \times \Sigma^* \times \Sigma^* \times \{:, ::\} \times \mathbf{Syn}^*$, and $N \in \mathcal{N} := (\Sigma^* \times \mathbf{Syn}^+)^*$. Given an expression $\langle i, N \rangle$, where $N = n_1, \ldots, n_k$, we write $i, n_1, \ldots, n_k$. If $k = 0$, we write simply i. We set $E := \mathcal{I} \times \mathcal{N}$. The functions merge and move are partial functions from $E \times E \to E$ and $E \to E$ respectively. We present them here in an inference-rule format for convenience.

merge : $E \times E \to E$ is the union of the following five functions, for $s_i, t_i \in \Sigma^*$ (for $1 \leq i \leq 3$), $\cdot \in \{:, ::\}$, $f \in \mathbf{sel}$, $g \in \{\texttt{f}, \texttt{*f}\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$ satisfying

(IMM1) no element of α or β has $\texttt{*f}$ as its first feature

$$\frac{(s_1, s_2, s_3) :: \texttt{=f}\gamma \qquad (t_1, t_2, t_3) \cdot g, \beta}{(s_1, s_2, s_3 t_1 t_2 t_3) : \gamma, \beta} \text{merge1}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=f}\gamma, \alpha \qquad (t_1, t_2, t_3) \cdot g, \beta}{(t_1 t_2 t_3 s_1, s_2, s_3) : \gamma, \alpha\beta} \text{merge2}$$

$$\frac{(s_1, s_2, s_3) \cdot \texttt{=f}\gamma, \alpha \qquad (t_1, t_2, t_3) : g\delta, \beta}{(s_1, s_2, s_3) : \gamma, \alpha(t_1 t_2 t_3, \delta)\beta} \text{merge3}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{=>f}\gamma \qquad (t_1, t_2, t_3) \cdot g, \beta}{(s_1, t_2 s_2, s_3 t_1 t_3) : \gamma, \beta} \text{affixRaise}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{f=>}\gamma \qquad (t_1, t_2, t_3) \cdot g, \beta}{(s_1, \epsilon, s_3 t_1 t_2 s_2 t_3) : \gamma, \beta} \text{affixLower}$$

As the domains of merge1, merge2, merge3, affixRaise and affixLower are all pairwise disjoint, their union is a function.

move : $E \to E$ is the union of the following two functions, for $s_1, s_2, s_3, t \in \Sigma^*$, $f \in \mathbf{lic}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$ satisfying:

(IMM2) no element of α or β has $\texttt{-f}$ as its first licensee feature

$$\frac{(s_1, s_2, s_3) : \texttt{+f}\gamma, \alpha(t, \texttt{-f})\beta}{(t s_1, s_2, s_3) : \gamma, \alpha\beta} \text{move1}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+f}\gamma, \alpha(t, \texttt{-f}\delta)\beta}{(s_1, s_2, s_3) : \gamma, \alpha(t, \delta)\beta}\text{move2}$$

Again, as the domains of move1 and move2 are disjoint, their union is a function.

To deal with control we add the following three rules to our grammar, for $s_i, t, t_i \in \Sigma^*$ (for $1 \leq i \leq 3$), $\cdot \in \{:, ::\}$, $f \in \mathbf{sel}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$ satisfying

(IMM3) no element of α or β has $\texttt{*f}$ as its first feature

$$\frac{(s_1, s_2, s_3) \cdot \texttt{=f}\gamma, \alpha \qquad (t_1, t_2, t_3) : \texttt{*f}\delta, \beta}{(s_1, s_2, s_3) : \gamma, \alpha(t_1 t_2 t_3, \texttt{*f}\delta)\beta}\text{cmerge}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=f}\gamma, \alpha(t, \texttt{*f}\delta)\beta}{(s_1, s_2, s_3) : \gamma, \alpha(t, \delta)\beta}\text{cmove1}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=f}\gamma, \alpha(t, \texttt{*f}\delta)\beta}{(s_1, s_2, s_3) : \gamma, \alpha(t, \texttt{*f}\delta)\beta}\text{cmove2}$$

The conditions 'IMM' implement what we have called the 'principle of immediacy', which states that a feature of an expression must be checked as soon as it is in principle possible to do so. The condition IMM2 on the domain of the move operation is referred to in all other work on minimalist grammars as the SMC, which is intended to recall Chomsky's [18] 'shortest move' constraint. As the actual shortest move constraint is different from the SMC, I have renamed it here so as to avoid possible terminology-induced confusion.

We define $CL(G) := \bigcup_{n=0} CL^n(G)$ to be the closure of Lex under merge, move, and our three control operations. Then the string language generated by a minimalist grammar $G = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \texttt{c} \rangle$ is defined to be

$$L(G) := \{s_1 s_2 s_3 : (s_1, s_2, s_3) \cdot \texttt{c} \in CL(G) \text{ for } \cdot \in \{:, ::\}\}$$

A.3 A Semantics for MGs

Just as we associated a string (more generally, a term in a phonological algebra) with a syntactic derivation in appendix A.2, here we show how to associate a meaning (a term in a semantic algebra) with a derivation.

Here, we take our semantic alphabet M to contain the binary function symbols $\texttt{G}$ (interpreted as the geach combinator), $\texttt{F}$ (interpreted as function application) and for each $i \in \mathbb{N}$, the nullary function symbols $\mathbf{x_i}$ and λ_i. This is a constant across languages. Furthermore, each of the denotations of our

lexical items is a nullary function symbol (e.g. **abbot** is a nullary function symbol).

As in appendix A.2, we present the generating functions in inference-rule format, with arguments on top, and result on bottom. The functions below are logically independent of the cases of merge and move presented previously. A specification of the form-meaning pairing (or what we might more traditionally call 'assigning meanings to expressions') requires us to specify which meaning generating functions can be used in tandem with which string generating functions (in the case of 'Free Retrieval' we need an 'identity' string generating function). Thus, our 'actual' generating functions are pairings $\langle \sigma, \mu \rangle$ of string and meaning generating functions.

On the semantic side of things, a minimalist expression is a pair $\langle i, N \rangle$, where $i \in \mathcal{I} := T_M \times \{:, ::\} \times \mathbf{Syn}^*$, and $N \in \mathcal{N} := (T_M^* \times \mathbf{Syn}^+)^*$. Elements of T_M^* we write between angled brackets ($\langle$, $\rangle$), and given $t, t' \in T_M^*$, $t^\frown t'$ is the concatenation of t and t' in that order. Given an expression $\langle i, N \rangle$, where $N = n_1, \ldots, n_k$, we write $i, n_1, \ldots, n_k$. If $k = 0$, we write simply i. We set $E := \mathcal{I} \times \mathcal{N}$.

We have the following three binary generating functions (which on the string side are associated with merge). σ and τ are elements of the meaning term algebra T_M, $\bullet f \in \{\texttt{=f}, \texttt{=>f}, \texttt{f=>}\}$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.[40]

$$\frac{\sigma \cdot \bullet f\gamma, \alpha \qquad \tau \cdot g, \beta}{\sigma(\tau) : \gamma, \alpha\beta}\text{FA}$$

$$\frac{\sigma \cdot \bullet f\gamma, \alpha \qquad \tau \cdot g, \beta}{\tau(\sigma) : \gamma, \alpha\beta}\text{BA}$$

$$\frac{\sigma \cdot \bullet f\gamma, \alpha \qquad \tau \cdot g\delta, \beta}{\sigma(\mathbf{x_i}) : \gamma, \alpha(\langle \mathtt{G}(\tau, \lambda_i) \rangle, \delta)\beta}\text{store}$$

The following four unary generating functions are associated with move on the string side. $\sigma, \tau \in T_M$, $t \in T_M^*$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(\langle \tau \rangle, \texttt{-f})\beta}{\tau(\sigma) \cdot \gamma, \alpha\beta}\text{Retrieve1}$$

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(\langle \tau \rangle^\frown t, \texttt{-f}\delta)\beta}{\tau(\sigma) \cdot \gamma, \alpha(t, \delta)\beta}\text{Retrieve2}$$

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(\langle\rangle, \texttt{-f})\beta}{\sigma \cdot \gamma, \alpha\beta}\text{Ignore1}$$

[40] We indicate here the results of interpreting the terms we generate in the intended model. This amounts to basically cashing out the `F` combinator at each step.

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(t, \texttt{-f}\delta)\beta}{\sigma \cdot \gamma, \alpha(t, \delta)\beta}\text{Ignore2}$$

The functions below are associated with thir namesakes from appendix A.2, and are tailored specifically for the movement theory of control outlined in § 2.7. $\sigma, \tau \in T_M$, $t \in T_M^*$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.

$$\frac{\sigma \cdot \texttt{=f}\gamma, \alpha \qquad \tau \cdot \texttt{*f}\delta, \beta}{\sigma(\mathbf{x_i}) : \gamma, \alpha(\langle \mathbf{x_i}, \texttt{G}\tau\lambda_i \rangle, \texttt{*f}\delta)\beta}\text{cmerge}$$

$$\frac{\sigma : \texttt{=f}\gamma, \alpha(\langle \tau \rangle^\frown t, \texttt{*f}\delta)\beta}{\sigma(\tau) : \gamma, \alpha(t, \delta)\beta}\text{cmove1}$$

$$\frac{\sigma : \texttt{=f}\gamma, \alpha(\langle \tau \rangle^\frown t, \texttt{*f}\delta)\beta}{\sigma(\tau) : \gamma, \alpha(\langle \tau \rangle^\frown t, \texttt{*f}\delta)\beta}\text{cmove2}$$

The generating function below is intended to allow for non-feature-driven scope-taking, or, in other words, reconstruction into landing sites of successive cyclic movements. To achieve the intended effects, we need an identity function on the string side, which maps each expression to itself, to be paired with this one. $\sigma, \tau \in T_M$, $t \in T_M^*$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.

$$\frac{\sigma \cdot \gamma, \alpha(\langle \tau \rangle^\frown t, \delta)\beta}{\tau(\sigma) \cdot \gamma, \alpha(t, \delta)\beta}\text{FreeRetrieval}$$

B Eliminating Indices

The rule of β-conversion in the lambda calculus allows us to substitute a formula for a variable, as shown below:

$$(\lambda x.a)b := a\{b/x\} \qquad (\beta\text{-conversion})$$

We must be certain, in performing this substitution, that variables in b don't accidentally get 'captured' by binders in a.[41] Accordingly, one standardly assumes that the variables in a and b are distinct—the α-conversion rule allows us to uniformly rename all instances of a variable, which guarantees that we can always construct appropriate a' and b' with distinct variables. However, this renaming of variables during β-conversion is left at the meta-level, and thus is not part of the calculus proper.

A similar problem arises in transformational syntax, where traces and other objects are assumed to have indices, and the distribution of indices on

[41] This is the condition that all that matters for variables is whether they are bound (or bindable) by the same binder or not.

objects is severely restricted. Here, we need to worry about how the index assigned to a term is decided. While it is easy for the linguist to assign the desired indices to structures, our utterances come out 'correctly indexed', and thus our theory needs to account for this. In the GB theory [15] indices were assumed to be freely generated, with filters (or constraints) ruling out all but a few be-indexed structures. Rogers [77] shows that free indexation increases the complexity of GB to the point where the emptiness problem becomes undecidable. Thus, we'd like some way of only generating correctly indexed structures to begin with.

In our current system, we don't have traces as syntactic objects. Moreover, we also don't have variables that we can refer to, to check whether an index has been already used (the variables in our notation (and the indices on them) are merely names for functions). For us, the problem of assigning the correct index is the problem of deciding which of the infinite number of functions $\mathbf{x_i}$ we select, and must be computed on the fly, as we build up our model-theoretic interpretation.[42]

In our case, we want to avoid putting together two independently constructed objects that have the same variables. How are we to do this, if we can't see which variables are in each object? *Because our binders shadow the expressions into which they bind*, we can avoid accidental variable capture by telling the binders to look at different indices in a systematic way. Given two assignments, g and h, we can put them together without losing any information by forming the new assignment $g \bowtie h := \langle g_0, h_0, g_1, h_1, \ldots \rangle$, which is the result of interleaving g and h. Now, given sets of assignments H_1 and H_2, we can combine them to form $H = H_1 \bowtie H_2$, which is the pairwise interleaving of assignments in H_1 with those in H_2. If λ_3 and λ_{17} are the binders shadowing H_1 and H_2 respectively, then λ_6 will have the same effect on H that λ_3 does on H_1, and λ_{35} will have the same effect on H had by λ_{17} on H_2.

B.1 Model-Theoretic Glory (cont'd)

The signatures of our meaning algebras contain the following six function symbols: nullary `x`, unary `E` and `O`, and binary `F`, `R` and λ. In the intended

[42]The issue is somewhat more nuanced than I have presented it here. Although if we do our computations over the model-theoretic objects, we can clearly not see indices, do we really think that the language user computes over these (infinitely large) objects? Certainly a computer program (which is still the best mental model we have) would compute over the formula itself, where the indices are visible. However, the question needs to be raised as to *how* this computation is effected. I am giving an explicit account thereof here.

model, $\mathtt{x}$ is $\mathbf{x_0}$. $\textsc{e}$ and $\textsc{o}$ are functions of type $[G \to E] \to G \to E$ such that

$$\textsc{e}(f)(g) = f(^{\vee}g) \qquad \textsc{o}(f)(g) = f(^{\wedge}g)$$

where $(^{\vee}g)(i) = g(2i)$ and $(^{\wedge}g)(i) = g(2i+1)$. Note that when f is a variable $\mathbf{x_i}$,

$$\textsc{e}(\mathbf{x_i})(g) = g_{2i} \qquad \textsc{o}(\mathbf{x_i})(g) = g_{2i+1}$$

We can think of $^{\vee}$ and $^{\wedge}$ as functions over G that take just the even and odd values respectively of their argument assignments, as schematized below.

$$^{\vee}g = \langle g_0, g_2, g_4, \ldots\rangle \qquad ^{\wedge}g = \langle g_1, g_3, g_5, \ldots\rangle$$

$\mathtt{F}$ is interpreted as function application. $\mathtt{R}$ is a function of type $[G \to \beta \to \gamma] \to [G \to \beta] \to G \to \gamma$ such that

$$\mathtt{R}ABg = (A(^{\vee}g))(B(^{\wedge}g))$$

Intuitively, $\mathtt{R}$ interprets assignment functions as encoding two others, and then decodes and passes the encoded assignments down into its arguments. $\lambda : [G \to E] \to [G \to T] \to [G \to E] \to G \to T$ such that[43]

$$\lambda(f)(H)(f')(g) = \mathbf{true} \text{ iff } \exists h \in H.\ f'(g) = f(h) \text{ and either } g = h \text{ or } \exists!j.\ g_j \neq h_j \wedge g_j = f(g) \wedge h_j = f(h))$$

Informally, λ takes an object f to abstract over, a formula H to bind it in,and returns a function that takes another object f', and yields the first formula with the first object substituted by the second. In particular, $\lambda(\mathbf{x_i}) = \lambda_i$.

B.2 A Semantics for MGs with Variable Management

On the semantic side of things, a minimalist expression is a pair $\langle i, N\rangle$, where $i \in \mathcal{I} := T_M \times \{:, ::\} \times \mathbf{Syn}^*$, and $N \in \mathcal{N} := ((T_M \times T_M) \times \mathbf{Syn}^+)^*$. Sequences over $T_M \times T_M$ we write between angled brackets ($\langle$, $\rangle$). Given an elements $f, \tau \in T_M$, we write $E(\langle f, \tau\rangle)$ for $\langle \textsc{e}(f), \tau\rangle$ (similarly, $O(\langle f, \tau\rangle)$ is $\langle \textsc{o}(f), \tau\rangle$). E and O are extended over sequences in the usual way. Given an expression $\langle i, N\rangle$, where $N = n_1, \ldots, n_k$, we write $i, n_1, \ldots, n_k$. If $k = 0$, we write simply i. We set $E := \mathcal{I} \times \mathcal{N}$.

[43]This is a monstrous type. Logical operators like λ, $\forall$, $\exists$, and even $\wedge$, $\vee$ and $\neg$ are standardly not given an explicit denotation, their meaning contribution being left implicit and in the meta-language (a notable exception is Church [19], who assigns explicit denotations to all but λ). Once everything is made fully explicit, the type we in fact do assign to λ emerges.

We have the following three binary generating functions (which on the string side are associated with merge). σ and τ are elements of the meaning term algebra T_M, $\bullet f \in \{\texttt{=f}, \texttt{=>f}, \texttt{f=>}\}$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.

$$\frac{\sigma \cdot \bullet f\gamma, \alpha \qquad \tau \cdot g, \beta}{\sigma(\tau) : \gamma, \alpha\beta}\text{FA}$$

$$\frac{\sigma \cdot \bullet f\gamma, \alpha \qquad \tau \cdot g, \beta}{\tau(\sigma) : \gamma, \alpha\beta}\text{BA}$$

$$\frac{\sigma \cdot \bullet f\gamma, \alpha \qquad \tau \cdot g\delta, \beta}{\texttt{R}(\sigma)(\texttt{x}) : \gamma, E(\alpha)(\langle \textsc{o}\texttt{x}, \tau\rangle, \delta)O(\beta)}\text{store}$$

The following four unary generating functions are associated with move on the string side. $\sigma, \tau \in T_M$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.[44]

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(\langle f, \tau\rangle, \texttt{-f})\beta}{\tau(\lambda(f)(\sigma)) \cdot \gamma, \alpha\beta}\text{Retrieve1}$$

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(\langle f, \tau\rangle, \texttt{-f}\delta)\beta}{\tau(\lambda(f)(\sigma)) \cdot \gamma, \alpha(\langle\rangle, \delta)\beta}\text{Retrieve2}$$

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(\langle\rangle, \texttt{-f})\beta}{\sigma \cdot \gamma, \alpha\beta}\text{Ignore1}$$

$$\frac{\sigma \cdot \texttt{+f}\gamma, \alpha(\langle f, \tau\rangle, \texttt{-f}\delta)\beta}{\sigma \cdot \gamma, \alpha(\langle f, \tau\rangle, \delta)\beta}\text{Ignore2}$$

The functions below are associated with their namesakes from appendix A.2, and are tailored specifically for the movement theory of control outlined in § 2.7. $\sigma, \tau \in T_M$, $t \in T_M^*$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.

$$\frac{\sigma \cdot \texttt{=f}\gamma, \alpha \qquad \tau \cdot \texttt{*f}\delta, \beta}{\texttt{R}(\sigma)(\texttt{x}) : \gamma, E(\alpha)(\langle \textsc{o}\texttt{x}, \tau\rangle, \texttt{*f}\delta)O(\beta)}\text{cmerge}$$

$$\frac{\sigma : \texttt{=f}\gamma, \alpha(\langle f, \tau\rangle, \texttt{*f}\delta)\beta}{\sigma(f) : \gamma, \alpha(\langle f, \tau\rangle, \delta)\beta}\text{cmove1}$$

$$\frac{\sigma : \texttt{=f}\gamma, \alpha(\langle f, \tau\rangle, \texttt{*f}\delta)\beta}{\sigma(f) : \gamma, \alpha(\langle f, \tau\rangle, \texttt{*f}\delta)\beta}\text{cmove2}$$

The generating function below is intended to allow for non-feature-driven scope-taking, or, in other words, reconstruction into landing sites of successive cyclic movements. To achieve the intended effects, we need an identity

[44]The notation here again gives the interpretation of the generated term in the intended model. As a term, the result of Retrieval is $\texttt{F}(\tau, \lambda(f, \sigma))$, which evaluates to the object which is written.

function on the string side, which maps each expression to itself, to be paired with this one. $\sigma, \tau \in T_M$, $\cdot \in \{:, ::\}$, $\gamma, \delta \in \mathbf{Syn}^+$, and $\alpha, \beta \in \mathcal{N}$.

$$\frac{\sigma \cdot \gamma, \alpha(\langle f, \tau \rangle, \delta)\beta}{\tau(\lambda(f)(\sigma)) \cdot \gamma, \alpha\beta}\text{FreeRetrieval}$$

Our expressions must take as their first argument an assignment function (given the proliferation of the combinator R). Accordingly, verbs (like *shave*) need to denote functions that take (and then throw away) an assignment function. Moreover, we will derive meaning terms like the following

$$\mathtt{R}(\mathbf{sleep})(\mathbf{x_i})$$

which, upon being supplied with an assignment function g reduces in the manner below, where the argument to the predicate is of type E, not of type $G \to E$.

$$\begin{aligned}
&\mathtt{R}(\mathbf{sleep})(\mathbf{x_i})(g)\\
&= (\mathbf{sleep}(^\vee g))(\mathbf{x_i}(^\wedge g))\\
&= \mathbf{sleep}(\mathbf{x_i}(^\wedge g))\\
&= \mathbf{sleep}((^\wedge g)(i))\\
&= \mathbf{sleep}(g(2i+1))\\
&= \mathbf{sleep}(g_{2i+1})
\end{aligned}$$

We therefore need to re-assign types to lexical expressions in our grammar. We do this systematically as per figure 17 (Pn is an n-place predicate, CN is a common noun, GQ is a generalized quantifier, and Det is a determiner). Note that we are forced to assign different types to one place predicates (P1s)

$$\begin{aligned}
&\text{P0} &&: G \to T\\
&\text{P1} &&: G \to E \to T\\
&\text{P2} &&: G \to E \to E \to T\\
&\vdots\\
&\text{CN} &&: [G \to E] \to G \to T\\
&\text{GQ} &&: [[G \to E] \to G \to T] \to G \to T\\
&\text{Det} &&: [[G \to E] \to G \to T] \to [[G \to E] \to G \to T] \to G \to T
\end{aligned}$$

Figure 17: The semantic type of expressions of natural language

and common nouns (CNs). These two grammatical categories (`n` and `V`) are usually treated as denoting in the same domain ($e \to t$), which fact makes puzzling their quite different syntactic distributions—why, in language after

language, do common nouns differ syntactically from verbs, although they denote the very same objects (sets of entities)? Having derived a semantic difference between predicates and nouns, we are presented with an opportunity to try and explain their observed differences based on this motivated formal one. In this case, the formal difference allows us to maintain a strong connection between semantic type and syntactic type: the relation between syntactic and semantic type is one-to-one.

We work through an example. The expression *every barber will shave an abbot* has the derivation below. We continue to assume for simplicity that only verbs, determiners, and nouns have non-trivial semantic contributions to make.

1. $\langle$merge1, FA$\rangle$(an::=n *d -k -q, abbot::n)

$$(\epsilon, \text{an}, \text{abbot}) : \texttt{*d -k -q}$$

$$\mathbf{some}(\mathbf{abbot}) : \texttt{*d -k -q}$$

2. $\langle$merge3, store$\rangle$(shave::=d V, 1)

$$(\epsilon, \text{shave}, \epsilon) : \texttt{V},\ (\text{an abbot}, \texttt{-k -q})$$

$$\texttt{R}(\mathbf{shave})(\texttt{x}) : \texttt{V},\ (\langle \textsc{O}\texttt{x}, \mathbf{some}(\mathbf{abbot})\rangle, \texttt{-k -q})$$

3. $\langle$affixRaise, FA$\rangle$(ϵ::=>V +k =d +q v, 2)

$$(\epsilon, \text{shave}, \epsilon) : \texttt{+k =d +q v},\ (\text{an abbot}, \texttt{-k -q})$$

$$\texttt{R}(\mathbf{shave})(\texttt{x}) : \texttt{+k =d +q v},\ (\langle \textsc{O}\texttt{x}, \mathbf{some}(\mathbf{abbot})\rangle, \texttt{-k -q})$$

4. $\langle$move2, Ignore2$\rangle$(3)

$$(\epsilon, \text{shave}, \epsilon) : \texttt{=d +q v},\ (\text{an abbot}, \texttt{-q})$$

$$\texttt{R}(\mathbf{shave})(\texttt{x}) : \texttt{=d +q v},\ (\langle \textsc{O}\texttt{x}, \mathbf{some}(\mathbf{abbot})\rangle, \texttt{-q})$$

5. $\langle$merge1, FA$\rangle$(every::=n *d -k -q, barber::n)

$$(\epsilon, \text{every}, \text{barber}) : \texttt{*d -k -q}$$

$$\mathbf{every}(\mathbf{barber}) : \texttt{*d -k -q}$$

6. $\langle$merge3, store$\rangle$(4, 5)

$$(\epsilon, \text{shave}, \epsilon) : \texttt{+q v},\ (\text{an abbot}, \texttt{-q}),\ (\text{every barber}, \texttt{-k -q})$$

$$\texttt{R}(\texttt{R}(\mathbf{shave})(\texttt{x}))(\texttt{x}) : \texttt{+q v},\ (\langle \textsc{EOx}, \mathbf{some}(\mathbf{abbot})\rangle, \texttt{-q}),\ (\langle \textsc{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \texttt{-k -q})$$

7. $\langle$move1, Retrieve1$\rangle$(6)

$$(\text{an abbot}, \text{shave}, \epsilon) : \texttt{v},\ (\text{every barber}, \texttt{-k -q})$$

$$\mathbf{some}(\mathbf{abbot})(\lambda(\textsc{EOx})(\texttt{R}(\texttt{R}(\mathbf{shave})(\texttt{x}))(\texttt{x}))) : \texttt{v},\ (\langle \textsc{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \texttt{-k -q})$$

8. $\langle$affixRaise, FA$\rangle$(ϵ::=>v prog, 7)

$$(\epsilon, \text{shave}, \text{an abbot}) : \texttt{prog},\ (\text{every barber}, \texttt{-k -q})$$

$$\mathbf{some}(\mathbf{abbot})(\lambda(\textsc{EOx})(\texttt{R}(\texttt{R}(\mathbf{shave})(\texttt{x}))(\texttt{x}))) : \texttt{prog},\ (\langle \textsc{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \texttt{-k -q})$$

9. $\langle$affixRaise, FA$\rangle$(ϵ::=>prog perf, 8)

$$(\epsilon, \text{shave}, \text{an abbot}) : \texttt{perf},\ (\text{every barber}, \texttt{-k -q})$$

$$\mathbf{some}(\mathbf{abbot})(\lambda(\textsc{EOx})(\texttt{R}(\texttt{R}(\mathbf{shave})(\texttt{x}))(\texttt{x}))) : \texttt{perf},\ (\langle \textsc{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \texttt{-k -q})$$

10. $\langle$merge1, FA$\rangle$(will::=perf +k +q s, 9)

$$(\epsilon, \text{will}, \text{shave an abbot}) : \texttt{+k +q s},\ (\text{every barber}, \texttt{-k -q})$$

$$\mathbf{some}(\mathbf{abbot})(\lambda(\textsc{EOx})(\texttt{R}(\texttt{R}(\mathbf{shave})(\texttt{x}))(\texttt{x}))) : \texttt{+k +q s},\ (\langle \textsc{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \texttt{-k -q})$$

11. $\langle$move2, Ignore2$\rangle$(10)

$$(\epsilon, \text{will}, \text{shave an abbot}) : \texttt{+q s},\ (\text{every barber}, \texttt{-q})$$

$$\mathbf{some}(\mathbf{abbot})(\lambda(\textsc{EOx})(\texttt{R}(\texttt{R}(\mathbf{shave})(\texttt{x}))(\texttt{x}))) : \texttt{+q s},\ (\langle \textsc{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \texttt{-q})$$

12. $\langle$move1, Retrieve1$\rangle$(11)

$$\text{(every barber, will, shave an abbot)} : \mathtt{s}$$

$$\mathbf{every}(\mathbf{barber})(\lambda(\mathrm{O}\mathtt{x})(\mathbf{some}(\mathbf{abbot})(\lambda(\mathrm{EO}\mathtt{x})(\mathtt{R}(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(\mathtt{x}))))) : \mathtt{s}$$

An assignment function g belongs to the set denoted by 12 just in case for every f such that $f(g) \in \mathbf{barber}$, g belongs to the set denoted by

$$(\lambda(\mathrm{O}\mathtt{x})(\mathbf{some}(\mathbf{abbot})(\lambda(\mathrm{EO}\mathtt{x})(\mathtt{R}(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(\mathtt{x}))))(f)$$

This happens just in case there is some assignment function g' which differs from g on at most the index i, such that $f(g) = \mathrm{O}\mathtt{x}(g')$, $g_i = \mathrm{O}\mathtt{x}(g)$, $g'_i = \mathrm{O}\mathtt{x}(g')$, and g' belongs to the set denoted by

$$\mathbf{some}(\mathbf{abbot})(\lambda(\mathrm{EO}\mathtt{x})(\mathtt{R}(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(\mathtt{x})))$$

Because $g_i = \mathrm{O}\mathtt{x}(g) = \mathtt{x}(^\wedge g) = (^\wedge g)(0) = g(2 \cdot 0 + 1) = g_1$ and $g'_i = \mathrm{O}\mathtt{x}(g') = g'_1$, since g differs from g' if at all then only at i, either $i = 1$ or $g = g'$. Accordingly, $g \approx_1 g'$. The new assignment g' belongs to the above set just in case there is some e such that $e(g') \in \mathbf{abbot}$, and g' belongs to the set denoted by

$$(\lambda(\mathrm{EO}\mathtt{x})(\mathtt{R}(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(\mathtt{x})))(e)$$

This in turn obtains iff there is another assignment function h differing from g' on at most the index j, such that $e(g') = \mathrm{EO}\mathtt{x}(h)$, $g'_j = \mathrm{EO}\mathtt{x}(g')$, $h_j = \mathrm{EO}\mathtt{x}(h)$, and h belongs to the set denoted by

$$\mathtt{R}(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(\mathtt{x})$$

Again, $g' \approx_2 h$, and therefore $f(g) = g'_1 = h_1$. Also, $e(g') = \mathrm{EO}\mathtt{x}(h) = \mathtt{x}(^\wedge{}^\vee h) = h_2$. Finally, h belongs to $\mathtt{R}(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(\mathtt{x})$ just in case

$$\begin{aligned}
&\mathtt{R}(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(\mathtt{x})(h)\\
=&(\mathtt{R}(\mathbf{shave})(\mathtt{x}))(^\vee h)(\mathtt{x}(^\wedge h))\\
=&(\mathbf{shave}(^\vee{}^\vee h))((\mathtt{x})(^\wedge{}^\vee h))(\mathtt{x}(^\wedge h))\\
=&\mathbf{shave}(\mathtt{x}(^\wedge{}^\vee h))(\mathtt{x}(^\wedge h))\\
=&\mathbf{shave}((^\wedge{}^\vee h)(0))((^\wedge h)(0))\\
=&\mathbf{shave}((^\vee h)(1))(h(1))\\
=&\mathbf{shave}(h(2))(h_1)\\
=&\mathbf{shave}(h_2)(h_1)
\end{aligned}$$

Therefore, *every barber will shave an abbot* is true with respect to an assignment g, just in case for every f such that $f(g) \in \mathbf{barber}$, there is some g', and some e such that $e(g') \in \mathbf{abbot}$, and $\mathbf{shave}(e(g'))(f(g))$.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] S. R. Anderson. *A-Morphous Morphology*. Cambridge University Press, 1992.

[3] D. Angluin. Inference of reversible languages. *Journal of the Association for Computing Machinery*, 29:741–765, 1982.

[4] M. Baker. *Incorporation: a theory of grammatical function changing*. MIT Press, Cambridge, Massachusetts, 1988.

[5] J. Blaszczak and H.-M. Gärtner. Intonational phrasing, discontinuity, and the scope of negation. *Syntax*, 8(1):1–22, 2005.

[6] C. Boeckx and S. Stjepanović. Head-ing toward PF. *Linguistic Inquiry*, 32(2):345–355, 2001.

[7] J. Bos. Predicate logic unplugged. In P. Dekker and M. Stokhof, editors, *Proceedings of the Tenth Amsterdam Colloquium*, pages 133–143, Amsterdam, 1996.

[8] M. Brody. *Lexico-Logical Form: A Radically Minimalist Theory*. MIT Press, Cambridge, Massachusetts, 1995.

[9] L. Burzio. *Italian syntax: A Government-Binding approach*. D. Reidel, Dordrecht, 1986.

[10] J. L. Bybee. *Morphology: A study of the relation between meaning and form*. Benjamins, Philadelphia, 1985.

[11] N. Chomsky. Minimalist inquiries: The framework. In R. Martin, D. Michaels, and J. Uriagereka, editors, *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik*, pages 89–155. MIT Press, Cambridge, Massachusetts, 2000.

[12] N. Chomsky. Three factors in language design. *Linguistic Inquiry*, 36 (1):1–22, 2005.

[13] N. Chomsky. On phases. In R. Freidin, C. P. Otero, and M. L. Zubizarreta, editors, *Foundational Issues in Linguistic Theory*, pages 133–166. MIT Press, Cambridge, Massachusetts, 2008.

[14] N. Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

[15] N. Chomsky. *Lectures on Government and Binding*. Foris, Dordrecht, 1981.

[16] N. Chomsky. A minimalist program for linguistic theory. In Hale and Keyser [34].

[17] N. Chomsky. Bare phrase structure. In G. Webelhuth, editor, *Government and Binding Theory and the Minimalist Program*, pages 383–439. MIT Press, Cambridge, Massachusetts, 1995.

[18] N. Chomsky. *The Minimalist Program*. MIT Press, Cambridge, Massachusetts, 1995.

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

[20] R. Cooper. *Quantification and Syntactic Theory*. D. Reidel, Dordrecht, 1983.

[21] A. Copestake, D. Flickinger, C. Pollard, and I. A. Sag. Minimal recursion semantics: An introduction. *Research on Language and Computation*, 3 (4):281–332, 2005.

[22] T. Cornell. Derivational and representational views of minimalist transformational grammar. In A. Lecomte, F. Lamarche, and G. Perrier, editors, *Logical Aspects of Computational Linguistics*, volume 1582 of *Lecture Notes in Computer Science*, pages 92–111, Berlin Heidelberg, 1999. Springer-Verlag.

[23] P. de Groote, G. F. Morrill, and C. Retoré, editors. *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence*, Berlin, 2001. Springer Verlag.

[24] D. Embick and R. Noyer. Movement operations after syntax. *Linguistic Inquiry*, 32(4):555–595, 2001.

[25] J. Engelfriet. Top-down tree transducers with regular look-ahead. *Mathematical Systems Theory*, 10:289–303, 1977.

[26] P. K. Feyerabend. How to be a good empiricist—a plea for tolerance in matters epistemological. In P. H. Nidditch, editor, *The Philosophy of Science*, Oxford Readings in Philosophy, chapter 1, pages 12–39. Oxford University Press, 1968.

[27] W. Frey and H.-M. Gärtner. Scrambling and adjunction in minimalist grammars. In G. Jäger, P. Monachesi, G. Penn, and S. Wintner, editors, *Proceedings of Formal Grammar 2002*, pages 41–52, 2002.

[28] Y. Fyodorov, Y. Winter, and N. Francez. Order-based inference in natural logic. *Logic Journal of the IGPL*, 11(4):385–416, 2003.

[29] H.-M. Gärtner and J. Michaelis. A note on the complexity of constraint interaction: Locality conditions and minimalist grammars. In P. Blache, E. Stabler, J. Busquets, and R. Moot, editors, *Logical Aspects of Computational Linguistics*, volume 3492 of *Lecture Notes in Computer Science*, pages 114–130. Springer, Berlin, 2005.

[30] G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge, MA, 1985.

[31] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[32] J. H. Greenberg, editor. *Universals of Language*. MIT Press, Cambridge, Massachusetts, 1963.

[33] K. K. Grohmann. *Prolific Peripheries: A Radical View From The Left*. PhD thesis, University of Maryland, 2000.

[34] K. Hale and S. J. Keyser, editors. *The View from Building 20*. MIT Press, Cambridge, Massachusetts, 1993.

[35] M. Halle and A. Marantz. Distributed morphology and the pieces of inflection. In Hale and Keyser [34], pages 111–176.

[36] H. Harkema. A characterization of minimalist grammars. In de Groote et al. [23].

[37] I. Heim and A. Kratzer. *Semantics in Generative Grammar*. Blackwell Publishers, 1998.

[38] J. Hintikka. No scope for scope? *Linguistics and Philosophy*, 20:515–544, 1997.

[39] C. F. Hockett. Two models of grammatical description. *Word*, 10:210–231, 1954.

[40] N. Hornstein. *Move! A Minimalist Theory of Construal*. Blackwell, 2001.

[41] N. Hornstein. *Logical Form: From GB to Minimalism*. Blackwell, Cambridge, Massachusetts, 1995.

[42] N. Hornstein. Movement and control. *Linguistic Inquiry*, 30(1):69–96, 1999.

[43] G. Jäger, P. Monachesi, G. Penn, J. Rogers, and S. Wintner, editors. *Proceedings of the 10th conference on Formal Grammar and the 9th Meeting on Mathematics of Language*, Edinburgh, 2005.

[44] M. Kanazawa. *Learnable Classes of Categorial Grammars*. CSLI Publications, Stanford University., 1998.

[45] R. Kayne. *The Antisymmetry of Syntax*. MIT Press, Cambridge, Massachusetts, 1994.

[46] E. L. Keenan. Beyond the Frege boundary. *Linguistics and Philosophy*, 15:199–221, 1992.

[47] W. R. Keller. Nested cooper storage: The proper treatment of quantification in ordinary noun phrases. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, number 35 in Studies in Linguistics and Philosophy, pages 432–447. D. Reidel, Dordrecht, 1988.

[48] G. M. Kobele. *Generating Copies: An investigation into structural identity in language and grammar*. PhD thesis, University of California, Los Angeles, 2006.

[49] G. M. Kobele and J. Michaelis. Two type 0-variants of minimalist grammars. In Jäger et al. [43].

[50] G. M. Kobele, C. Retoré, and S. Salvati. An automata theoretic approach to minimalism. In J. Rogers and S. Kepser, editors, *Proceedings of the Workshop Model-Theoretic Syntax at 10; ESSLLI '07*, Dublin, 2007.

[51] M. Koizumi. *Phrase Structure In Minimalist Syntax*. PhD thesis, Massachusetts Institute of Technology, 1995.

[52] H. Koopman and D. Sportiche. The position of subjects. *Lingua*, 85: 211–258, 1991.

[53] A. Kratzer. Severing the external argument from its verb. In J. Rooryck and L. Zaring, editors, *Phrase Structure and the Lexicon*, pages 109–137. Kluwer, Dordrecht, 1996.

[54] G. Kreisel and J.-L. Krivine. *Elements of Mathematical Logic (Model Theory)*. North-Holland, Amsterdam, 1967.

[55] S. Kripke. *Naming and Necessity*. Harvard University Press, 1980.

[56] R. K. Larson. *Promise* and the theory of control. *Linguistic Inquiry*, 22 (1):103–139, 1991.

[57] H. Lasnik. Chains of arguments. In S. D. Epstein and N. Hornstein, editors, *Working Minimalism*, number 32 in Current Studies in Linguistics, chapter 8, pages 189–215. MIT Press, Cambridge, Massachusetts, 1999.

[58] J. Lidz and W. J. Idsardi. Chains and phono-logical form. In A. Dimitriadis, H. Lee, C. Moisset, and A. Williams, editors, *University of Pennsylvania Working Papers in Linguistics*, volume 8, pages 109–125, 1998.

[59] A. Mahajan. Eliminating head movement. In *The 23rd Generative Linguistics in the Old World Colloquium*, 2000.

[60] M. R. Manzini and A. Roussou. A minimalist theory of A-movement and control. *Lingua*, 110(6):409–447, 2000.

[61] A. Marantz. *On the Nature of Grammatical Relations*. MIT Press, Cambridge, Massachusetts, 1984.

[62] O. Matushansky. Going through a phase. In M. McGinnis and N. Richards, editors, *Perspectives on Phases*, number 49 in MIT Working Papers in Linguistics, Cambridge, Massachusetts, 2005.

[63] O. Matushansky. Head movement in linguistic theory. *Linguistic Inquiry*, 37(1):69–109, 2006.

[64] J. Michaelis. Transforming linear context-free rewriting systems into minimalist grammars. In de Groote et al. [23].

[65] J. Michaelis. An additional observation on strict derivational minimalism. In Jäger et al. [43].

[66] J. Michaelis. Derivational minimalism is mildly context-sensitive. In M. Moortgat, editor, *Logical Aspects of Computational Linguistics, (LACL '98)*, volume 2014 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, Berlin, Heidelberg, Germany, 1998.

[67] R. Montague. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, New Haven, 1974. edited and with an introduction by R. H. Thomason.

[68] M. Moortgat. Categorial type logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*. Elsevier, Amsterdam, 1996.

[69] N. Nasu. *Aspects of the syntax of A-movement: A study of English infinitival constructions and related phenomena*. PhD thesis, University of Essex, 2002.

[70] J. H. O'Neil III. *Means of Control: Deriving the Properties of PRO in the Minimalist Program*. PhD thesis, Harvard, 1997.

[71] P. M. Postal. *On Raising: One Rule of English Grammar and its Theoretical Implications*. MIT Press, Cambridge, Massachusetts, 1974.

[72] L. Pylkkänen. *Introducing Arguments*. PhD thesis, Massachusetts Institute of Technology, 2002.

[73] G. Restall. *An Introduction to Substructural Logics*. Routledge, 2000.

[74] C. Retoré and E. P. Stabler. Resource logics and minimalist grammars. *Research on Language and Computation*, 2(1):3–25, 2004.

[75] L. Rizzi. *Relativized Minimality*. MIT Press, Cambridge, Massachusetts, 1990.

[76] R. H. Robins. In defense of WP. *Transactions of the Philological Society*, pages 116–144, 1959. Reprinted in *Transactions of the Philological Society* 99(2):171–200.

[77] J. Rogers. *A Descriptive Approach to Language-Theoretic Complexity*. CSLI Publications, 1998.

[78] P. S. Rosenbaum. *The grammar of English predicate complement constructions*. MIT Press, Cambridge, Massachusetts, 1967.

[79] H. Seki, T. Matsumura, M. Fujii, and T. Kasami. On multiple context-free grammars. *Theoretical Computer Science*, 88:191–229, 1991.

[80] S. M. Shieber. Unifying synchronous tree-adjoining grammars and tree transducers via bimorphisms. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL-2006)*, pages 377–384, Trento, 2006.

[81] E. P. Stabler. Minimalist grammars and recognition. In C. Rohrer, A. Rossdeutscher, and H. Kamp, editors, *Linguistic Form and its Computation*, pages 327–352. CSLI Publications, 2001.

[82] E. P. Stabler. Recognizing head movement. In de Groote et al. [23], pages 254–260.

[83] E. P. Stabler. Comparing 3 perspectives on head movement. In A. Mahajan, editor, *Syntax at Sunset 3: Head Movement and Syntactic Theory*, volume 10 of *UCLA/Potsdam Working Papers in Linguistics*, pages 178–198, 2003.

[84] E. P. Stabler. Sidewards without copying. In P. Monachesi, G. Penn, G. Satta, and S. Wintner, editors, *Proceedings of the 11th conference on Formal Grammar*, pages 133–146. CSLI, 2006.

[85] E. P. Stabler. Derivational minimalism. In C. Retoré, editor, *Logical Aspects of Computational Linguistics*, volume 1328 of *Lecture Notes in Computer Science*, pages 68–95. Springer-Verlag, Berlin, 1997.

[86] E. P. Stabler. Remnant movement and complexity. In G. Bouma, E. Hinrichs, G.-J. M. Kruijff, and R. T. Oehrle, editors, *Constraints and Resources in Natural Language Syntax and Semantics*, chapter 16, pages 299–326. CSLI Publications, Stanford, CA, 1999.

[87] E. P. Stabler and E. L. Keenan. Structural similarity within and among languages. *Theoretical Computer Science*, 293:345–363, 2003.

[88] M. Steedman. *The Syntactic Process*. MIT Press, 2000.

[89] G. T. Stump. *Inflectional Morphology: A Theory of Paradigm Structure*. Cambridge University Press, 2001.

[90] A. Tarski. The concept of truth in formalized languages. In J. Corcoran, editor, *Logic, Semantics, Metamathematics*, chapter 8, pages 152–278. Hackett Publishing Company, Indianapolis, 1983.

[91] L. Travis. *Parameters and effects of word order variation.* PhD thesis, Massachussets Institute of Technology, Cambridge, Massachusetts, 1984.

[92] J. van Benthem. Meaning: Intrpretation and inference. *Synthese*, 73: 451–470, 1987.

[93] C. Wartena. *Storage Structures and Conditions on Movement in Natural Language Syntax.* PhD thesis, Universität Potsdam, 1999.

[94] A. Zamansky, N. Francez, and Y. Winter. A 'Natural Logic' inference system using the Lambek calculus. *Journal of Logic, Language and Information*, 15(3):273–295, 2006.