University of California
Los Angeles

Generating Copies: An investigation into structural identity in language and grammar

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Linguistics

by

Gregory Michael Kobele

2006

© Copyright by
Gregory Michael Kobele
2006

The dissertation of Gregory Michael Kobele is approved.

Edward L. Keenan

Marcus Kracht

Charles E. Taylor

Colin Wilson

Edward P. Stabler, Committee Chair

University of California, Los Angeles

2006

To my brother, Steve, whom I love and admire.

Table of Contents

1 Introduction 1

1 Structure of the Thesis 2

A–1 Mathematical Preliminaries 5

2 Syntax and Semantics in Minimalist Grammars 14

1 Introducing Minimalism 15

1.1 Syntactic Structures 16

1.2 Features 18

1.3 Head Movement 24

1.4 Phrasal Movement 31

1.5 Interim Summary 49

2 Introducing Phases 50

2.1 To PF Without Trees 55

2.2 Direct Compositionality 61

2.3 Semantics in Chains 63

2.4 Model-Theoretic Glory 66

2.5 Quantifiers and Scope 72

2.6 Raising and Passive 81

2.7 Control 85

2.8 Reflections on Control 95

3 Summary 106

B–1 Definitions 110
B–1.1 Minimalist Grammars on Trees 110
B–1.2 Minimalist Grammars without Trees 116
B–1.3 A Semantics for MGs 119
B–2 Eliminating Indices 121
B–2.1 Model-Theoretic Glory (cont'd) 123
B–2.2 A Semantics for MGs with Variable Management 124

3 Copying in Grammar 132
1 The Copy Theory of Movement 132
1.1 Where Does Copying Take Place? 137
1.2 Internal Merge 138
1.3 External Merge 147
2 The Pronunciation of Copy-Chains 160
2.1 Pronounce Highest 161
2.2 Local Nondeterminism 170
2.3 Pronouncing Multiple Copies 180
3 Summary 183
C–1 Two ways to copy 185
C–1.1 Copying the Derived Tree 186
C–1.2 Copying the Derivation Tree 188
C–1.3 The Complexity of Derivational Copying 193
C–2 Pronunciation of Copies 198

C–2.1 Spelling Out Only One Chain Link 199

C–2.2 Spelling Out Multiple Chain Links 207

4 Copying in Language 211

1 Yoruba 212

1.1 Simple Sentences 214

1.2 Serial Verbs 217

1.3 Relative Clauses 227

2 Summary 245

D–1 On the complexity of natural language 247

D–1.1 The hypothesis of the mild context-sensitivity of natural languages 248

D–1.2 The structure of a challenge 249

D–1.3 Copying (of copies)* in Yoruba 255

5 Conclusions 261

List of Figures

1.1 $\sigma(t_1, \ldots, t_n)$ as a tree 8

2.1 The English auxiliary system (I) 25

2.2 Head movement of auxiliaries in English 26

2.3 T-to-C movement 27

2.4 The English auxiliary system (II) 28

2.5 An intermediate stage in the derivation of 2.27 33

2.6 The English auxiliary system (III) 35

2.7 Specifying the associations between sounds and meanings 53

2.8 Directly interpreting derivations 54

2.9 Syntactically indistinguishable expressions 55

2.10 Two accounts of the structure of the non-finite TP 61

2.11 A model for the subject-wide scope reading of sentence 2.43 . . . 64

2.12 A model for the subject-narrow scope reading of sentence 2.43 . . 64

2.13 Modes of Semantic Combination (I) 74

2.14 Modes of Semantic Combination (II) 75

2.15 Modes of Semantic Combination (III) 77

2.16 A Grammar for English A-movement 107

2.17 The semantic type of expressions of natural language 127

3.1 Explicit versus implicit representations of copies 139

3.2 A Morphological Component 177

3.3 A grammar for the language a^{2^n} 186

3.4 Computing m-command in the derivation tree 195

4.1 Yoruba (I) 215

4.2 Yoruba (II) 216

4.3 The derived structure for sentence 4.13 220

4.4 The derived structure for sentence 4.14 223

4.5 Copyable Constituents 233

4.6 Copying at different points in the derivation 237

4.7 A grammar for a fragment of Yoruba 244

4.8 Case stacking in Old Georgian 253

Acknowledgments

This thesis owes its very existence to my mentor, committee chair, and fellow master-brewer Ed Stabler, whom I will never be able to thank enough for his influence on my intellectual development. Thanks also to the other members of my committee, Ed Keenan, Marcus Kracht, Chuck Taylor, and Colin Wilson, who have each contributed in substantive ways to my development at UCLA. I have been here too long, and the contributions run too deep and varied to list them all.

I have benefited from talks with nearly all of our faculty and graduate students, although I must give a special shout out to Jason Riggle, my office mate, friend, co-author, and traveling companion, who left too soon, leaving a yawning chasm behind. Jeff Heinz quickly took advantage of this, settling himself in unabashedly behind Jason's old desk. My gratitude to both of them, for challenging me intellectually, and for their friendship. Also notable in the sheer magnitude of their awesomeness are (in no particular order) Adam Anderson, Julia Berger-Morales, Leston Buell, Howard Chen, Travis Collier, Eric Fiala, Jesús Gaytán, Hans-Martin Gärtner, Dieter Gunkel, John Hale, Yoosook Lee, Jens Michaelis, Mike Pan, Katya Pertsova, Elisa Pigeron, Kathryn Roberts, Manola Salustri, Shabnam Shademan, Anna d'Souza, Luca Storto, Harold Torrence, Willemijn Vermaat, Sarah van Wagenen, Steve Wedig, Pierre-Yves Yanni, and many more.

Special thanks are due to Selassie Ahorlu and Damola Osinulu, without whom this thesis, and my linguistic education, would have been far poorer.

Writing this thesis took a lot of time. It was good that I took off eight hours a week for Wing Tzun. Thanks to Sifu Michael Casey, and all of my EBMAS friends for intense training, and great fun.

Many of the above mentioned individuals have significant others who are also quite cool; they will have to content themselves with this most indirect of shouts out.

Finally, I am forever grateful to my loving family, in me all of whom can be seen reflected.

Vita

1978	Born, Upland, California
1996	Graduated, Diamond Bar High School, Diamond Bar, California.
2001	B.A., Linguistics & Computer Science, and Philosophy, *cum laude*, UCLA.

Publications

Gregory M. Kobele. Features Moving Madly: A Formal Perspective on Feature Movement in the Minimalist Program. *Research on Language and Computation*, 3(4):391–410. 2005.

Gregory M. Kobele Formalizing Mirror Theory. *Grammars*, 5(3):177–221. 2002

Abstract of the Dissertation

Generating Copies: An investigation into structural identity in language and grammar

by

Gregory Michael Kobele

Doctor of Philosophy in Linguistics

University of California, Los Angeles, 2006

Professor Edward P. Stabler, Chair

In this dissertation I provide a directly compositional semantics for minimalist grammars, which allows us to view the derivation as the only relevant syntactic structure, thereby eliminating all non-interface levels, and obtaining a system similar in this respect to categorial grammar. I give an explicit account of a fragment of English consisting of passivization, raising, control, and expletive-*it*, which accounts for quantifier scope ambiguities. The system is quite minimal; there are no trans-derivational economy conditions, no preferences for merge over move, no numerations, no lexical sub-arrays. Instead, all operations are feature driven, and there is a single economy condition, the *Principle of Immediacy*, which simply requires that features be checked as soon as the appropriate configuration arises.

I add copy movement to the minimalist grammar system. I implement copying in two ways. Once with multiple dominance, treating copies as being derived only once, and once with synchronous derivation, treating each copy as having been

derived. Both approaches are strongly equivalent, and generate only languages in **P**. Our semantics extends immediately to minimalist grammars with copying.

I turn next to the West African language Yoruba, which has constructions characterized by overt copying of VPs. As Yoruba also has serial verbs, there is no principled upper bound on the size of the copied VP. I give an explicit account of a fragment of Yoruba consisting of serialization and relative clauses, both over predicates (the so-called relativized predicate) and over nouns. Copying in Yoruba relativized predicates is not a straightforward matter, with the copy relation sometimes rendered opaque by other processes. However, our minimalist grammars with copying account elegantly for the range of copy possibilities, and assign natural structures to these copies, which no known mildly context sensitive formalism is capable of doing.

CHAPTER 1

Introduction

My goal in this thesis is to investigate copy movement in the minimalist paradigm. I ask and answer such questions as

- Is the copy-theory of movement non-compositional (as has been suggested by Cormack and Smith [47])?
- Can the choice of which copy to pronounce be made in a way compatible with a cyclic approach to spellout?
- More generally, can the copy-theory of movement be reconciled with phase-theory?
- How would a theory of copying based on external merge work?
- What is the difference between copying via internal or external merge? Is one more complex than the other? How much more? Why?
- Does the copy-theory of movement provide us with a tractable formalism which is able to assign "natural" structures to sentences with copies [139]?
- Does the copy-theory of movement increase the expressive power of our theory? By how much? Is such an increase warranted on empirical grounds?

In order to do this I have to provide a theory in enough detail to allow for a serious investigation of these questions. The architectural assumptions I adopt

are minimal, but often particular analytical choices need to be made that are orthogonal and irrelevant to the ultimate points I want to discuss (such as, is control movement, is there successive cyclic movement, is there an Agr projection between the subject and the object, is feature checking symmetric, is there covert movement, etc). When such choices present themselves, I will adopt the simplest option, mentioning alternatives in the literature. The investigation of copying conducted herein is largely independent of these particular choices.

1 Structure of the Thesis

Chapter 2 introduces the particular version of minimalism used throughout this thesis. All assumptions about the architecture of the grammar are made fully explicit, and motivated. This is done by analyzing successively more and more complex fragments of English, which, at the end of the first section, includes passivization, raising, and expletive constructions. In the second section, the concept of a 'phase' is introduced, and it is shown how the formalism we have developed lends itself naturally to a strong theory of phases, whereby *every* head introduces a phase. Furthermore, the concept of successive cyclicity is discussed, and we see that our formalism is also strongly successive cyclic. Next, a compositional semantics for our formalism is provided, one that is both compatible with our strong theory of phases, and which makes sense of the intuition that chains, not chain links, are the units of interpretation. We show how our fragment supports the semantics we have just developed, and extend it further to deal with control and quantifier scope ambiguities.

Chapter 3 introduces the copy theory of movement. Two approaches to copying are distinguished, copying the derived tree and copying the derivation tree,

which we identify with the distinction between internal and external merge. We show how to extend our formalism so as to allow for each of these approaches of copying, and discuss the relative and absolute complexity of these two approaches. The semantics developed in chapter 2 is seen to carry over directly to both approaches, thereby proving that copying is not inherently non-compositional. We formally reconstruct a variety of sophisticated linearization strategies in our system, rendering them compatible with our strong theory of phases. Finally, we extend our fragment of English with *there*-insertion, showing the relation between long-distance agreement and covert movement.

Chapter 4 justifies the increase in expressive power due to the addition of a copy operation in our grammar. Looking at the relativized predicate construction in Yoruba, which allows for copies of unbounded size, we are able to probe into the mechanics of the human copier, something we are unable to do when focussing on syntactically simple copies. Interspersed amongst the full copies in Yoruba, we find surface discontinuous, 'opaque' ones. Our syntactic copy mechanism, which copies points in the derivation, provides us with just the right range of copying opportunities, which are independently motivated, given our fully compositional treatment of object sharing in serial verbs. Our fragment of Yoruba also includes relative clauses, which are given a raising analysis, and a natural conjunctive semantics. Finally, I argue against the hypothesis that natural languages are mildly context sensitive (MCS). Current MCS formalisms are unable to assign natural structures to sentences with copies, and the natural treatment given here to Yoruba, which allows copies to contain copies, is not describable by these formalisms at all.

Chapter 5 provides a brief synopsis of the thesis, taking a step back and reflecting on the significance of the results obtained.

Appendices

These appendices make formally precise the mechanisms and ideas introduced in the main chapters. As such, they require a bit of mathematical sophistication, and may be skipped over on first reading. I have tried to make the notation consistent across the main chapters and the appendices.

A–1 Mathematical Preliminaries

Sets, Relations, and Functions

$\mathbb{N}$ is the set $\{0, 1, 2, \ldots\}$ of natural numbers. The set with no elements is denoted $\emptyset$. Given two sets A and B, A is a subset of B ($A \subseteq B$) just in case every element $a \in A$ is also in B. The usual operations of union, intersection, difference, cross-product, and powerset are defined below[1]

$$A \cup B := \{c : c \in A \text{ or } c \in B\} \qquad \text{(union)}$$

$$A \cap B := \{c : c \in A \text{ and } c \in B\} \qquad \text{(intersection)}$$

$$A - B := \{c : c \in A \text{ and } c \notin B\} \qquad \text{(difference)}$$

$$A \times B := \{\langle a, b\rangle : a \in A \text{ and } b \in B\} \qquad \text{(cross-product)}$$

$$\mathbf{2}^A := \{C : C \subseteq A\} \qquad \text{(powerset)}$$

Given a family of sets $\mathcal{B} \subseteq \mathbf{2}^A$, for some A, the union of $\mathcal{B}$, written $\bigcup \mathcal{B}$, is the set which contains an element a iff a is a member of some $B \in \mathcal{B}$. The intersection of $\mathcal{B}$, $\bigcap \mathcal{B}$, is defined dually. $\mathcal{B} \subseteq \mathbf{2}^A$ is a partition of A if and only if (iff) the elements of $\mathcal{B}$ are non-empty and pairwise disjoint (i.e. for B, B' distinct

[1] The object $\langle a, b\rangle$ is the ordered pair whose first projection is a and whose second projection is b. We may take $\langle a, b\rangle$ as an abbreviation for the set $\{a, \{a, b\}\}$.

elements of $\mathcal{B}$, $B \cap B' = \emptyset$) and $\bigcup \mathcal{B} = A$. If $\mathcal{B}$ is a partition, its elements are called blocks.

Relations

A binary relation on sets A and B is a subset of $A \times B$. If R is a binary relation and $\langle a, b\rangle \in R$ we sometimes write aRb. Given $R \subseteq A \times B$ and $C \subseteq A$, the restriction of R to C is the relation $R \restriction C := \{\langle a, b\rangle : a \in C \text{ and } aRb\}$. The inverse of a relation R is denoted R^{-1}, and is defined such that aRb iff $bR^{-1}a$. The identity relation over a set A is $\mathtt{id}_A := \{\langle a, a\rangle : a \in A\}$. A relation $R \subseteq A \times A$ may have some of the following properties, for every $a, b, c \in A$

$$aRa \qquad \text{(reflexivity)}$$

$$aRb \text{ implies } bRa \qquad \text{(symmetry)}$$

$$aRb \text{ and } bRc \text{ implies } aRc \qquad \text{(transitivity)}$$

$$aRb \text{ implies not } bRa \text{ unless } a = b \qquad \text{(anti-symmetry)}$$

$$aRb \text{ implies not } bRa \qquad \text{(asymmetry)}$$

An equivalence relation is one which is reflexive, symmetric and transitive. An equivalence relation over a set A partitions A into sets of elements mutually related to each other. A partial order is a relation which is reflexive, anti-symmetric, and transitive. A total order $R \subseteq A \times A$ is a partial order where for every two elements $a, b \in A$, either aRb or bRa. Given a relation $R \subseteq A \times A$, its transitive closure is defined to be $R^+ := \bigcup\{R^{n+1} : n \in \mathbb{N}\}$, where

$$R^0 := \mathtt{id}_A$$

$$R^{n+1} := \{\langle a, c\rangle : aRb \text{ and } bR^n c\}$$

The reflexive transitive closure of a relation $R \subseteq A \times A$ is written R^*, and is $R^+ \cup \mathtt{id}_A$.

Functions

A function f from A (its domain) to B (its codomain), written $f : A \to B$, is a relation $f \subseteq A \times B$ such that if a pair $\langle a, b\rangle$ is in f, there is no other pair $\langle a, b'\rangle \in f$, where $b \neq b'$. In this case we write $f(a) = b$. f is total if for every $a \in A$ there is some b such that $f(a) = b$, it is partial otherwise. The set of all total functions from A to B is denoted with $[A \to B]$. Given $f : A \to B$ and $g : B \to C$, their composition $g \circ f : A \to C$ exists and is defined by $(g \circ f)(a) := g(f(a))$. We can view any relation $R \subset A \times B$ as a function $f_R : A \to \mathbf{2}^B$, where $f_R(a) := \{b : aRb\}$, and a function $f : A \to B$ extends to a function $\mathbf{2}^f : \mathbf{2}^A \to \mathbf{2}^B$, where $\mathbf{2}^f(X) := \{f(a) : a \in X\}$. As is standard, I write f_R as R and $\mathbf{2}^f$ as f, when no confusion will arise.

I use the notation $\lambda x \in A.\phi(x)$ to denote the function that takes elements $a \in A$ and returns $\phi(a)$. For example, $\lambda x \in \mathbb{N}.x + 3$ denotes the function f such that $f(x) = x + 3$. When the domain of f is understood from context, I do not write it explicitly.

Sequences, Words and Languages

Given a set A, a sequence of length $n \in \mathbb{N}$ over A is a function $\alpha : \mathbf{n} \to A$, where $\mathbf{n} := \{0, \ldots, n-1\}$. The unique sequence of length 0 is called the empty sequence, and is written ϵ. Given sequences α and β of length $m \in \mathbb{N}$ and $n \in \mathbb{N}$ respectively, their concatenation $\alpha^\frown\beta$ is of length $m + n$ and is defined by

$$\alpha^\frown\beta(i) := \textbf{if } i < m \textbf{ then } \alpha(i) \textbf{ else } \beta(i - m)$$

We often write the concatenation symbol as $\cdot$, or leave it out entirely, and represent the concatenation of α and β by simple juxtaposition.

We write a sequence α of length n as $\langle a_0, \ldots, a_{n-1}\rangle$, where $a_i = \alpha(i)$. We

denote with A^* the set of all finite sequences over A, and with A^+ the set of all non-empty such sequences. A function $f : A \to B$ extends to a function $f^* : A^* \to B^*$ by setting $f^*(\alpha) := f \circ \alpha$.

An alphabet is a non-empty finite set Σ (of words). A sentence $w \in \Sigma^*$ is a finite sequence of words, and a language $L \subseteq \Sigma^*$ is a (possibly infinite) set of sentences. Given two languages $S, T \subseteq \Sigma^*$, we write $S \cdot T$ (or ST) to denote the set $\{\sigma\tau : \sigma \in S \text{ and } \tau \in T\}$ of concatenations of sequences from S with sequences from T. We set $S^0 := \{\epsilon\}$, and $S^{n+1} := S \cdot S^n$.

Terms and Trees

A *ranked set* is a pair consisting of a set Σ of function symbols and a function **arity** : $\Sigma \to \mathbb{N}$ assigning to each $\sigma \in \Sigma$ its arity, or rank. $\Sigma_n \subseteq \Sigma$ is the set of n-ary function symbols. If $\sigma \in \Sigma_n$, I will sometimes indicate this by writing $\sigma^{(n)}$ when referring to σ. The set of *ground terms* over Σ is the smallest set T_Σ such that

1. $\Sigma_0 \subseteq T_\Sigma$
2. if $\sigma^{(n)} \in \Sigma$ and $t_1, \ldots, t_n \in T_\Sigma$ then $\sigma(t_1, \ldots, t_n) \in T_\Sigma$

A term $\sigma(t_1, \ldots, t_n)$ represents the tree whose root is labelled with σ and which has subtrees $t_1, \ldots, t_n$ in that order (figure 1.1). A *context* is a tree with holes.

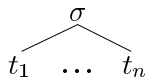


Figure 1.1: $\sigma(t_1, \ldots, t_n)$ as a tree

Given a set X of names of holes, we denote with $T_\Sigma(X)$ the set of terms $T_{\Sigma \cup X}$

over the ranked set $\Sigma \cup X$, where elements of Σ have their usual rank, and elements of X have rank 0. When $X = \emptyset$, $T_\Sigma(X) = T_\Sigma$. We let $X = \{x_1, x_2, \ldots\}$ be a denumerable set of variables, and $X_n = \{x_1, \ldots, x_n\}$ the set of the first n variables. Given a term $t \in T_\Sigma(X_n)$, and terms $s_1, \ldots, s_n \in T_\Sigma(X_m)$, $t[s_1, \ldots, s_n]$ is the term in $T_\Sigma(X_m)$ where each x_i in t is replaced by s_i. Formally, we define

1. $x_i[s_1, \ldots, s_n] := s_i$, for $1 \leq i \leq n$
2. $\sigma(t_1, \ldots, t_k)[s_1, \ldots, s_n] := \sigma(t_1[s_1, \ldots, s_n], \ldots, t_k[s_1, \ldots, s_n])$

A context $C \in T_\Sigma(X_n)$ is *linear* just in case no $x \in X_n$ occurs more than once in C. It is *complete* just in case every $x \in X_n$ occurs at least once in C.

Tree Transducers

Of the myriad possible functions from terms over one ranked set Σ to another Δ, some preserve more of the structure of the terms than do others. This idea has been made precise in terms of tree transducers of various sorts, which can deform terms in some ways, but not in others. We will be primarily concerned with macro tree transducers (MTT), which are a generalization of top-down tree transducers ($\downarrow TT$), which are themselves a generalization of homomorphisms.[2]

Tree Homomorphisms

A tree homomorphism $h : T_\Sigma \to T_\Delta$ is given by a function $\hat{h} : \Sigma \to T_\Delta(X)$ which assigns n-ary function symbols in Σ to n-ary contexts over Δ. We define h as

[2] [44, 65] are good overviews of tree automata, and tree transducers. [53] introduces regular look-ahead, and macro tree transducers are studied in [54].

follows.

$$h(\sigma^{(0)}) = \hat{h}(\sigma)$$
$$h(\sigma^{(n)}(t_1, \ldots, t_n) = \hat{h}(\sigma)[h(t_1), \ldots, h(t_n)]$$

A homomorphism h is linear or complete if the range of its kernel $\hat{h}$ is.

Top-down Tree Transducers

Intuitively, a top-down tree transducer is a homomorphism with a limited memory, which allows it to map the very same function symbol in Σ to different contexts in $T_\Delta(X)$ depending on the contents of its memory. We formalize the notion of memory as a finite set of states. Formally, a top-down tree transducer is a quintuple $M = \langle Q, \Sigma, \Delta, \rightarrow, q_0 \rangle$, where Q is a finite set of states all of rank one, $q_0 \in Q$ is the initial state, Σ and Δ are ranked sets of input and output symbols respectively, and $\rightarrow$ is a set of productions of the form

$$q(\sigma^{(k)}(x_1, \ldots, x_k)) \rightarrow C[q_1(x_1), \ldots, q_k(x_k)]$$

where $q, q_1, \ldots, q_k \in Q$, $\sigma \in \Sigma$, and $C \in T_\Delta(X_k)$. A homomorphism h is the limiting case of a top-down transducer where $Q = \{q_0\}$. M is deterministic iff no two distinct productions have the same left-hand side, total iff for every $q \in Q$ and $\sigma^{(k)} \in \Sigma$ there is exactly one production with left-hand side $q(\sigma(x_1, \ldots, x_k))$, and linear iff every context in the right-hand side of a production is. Given terms $t, t' \in T_{\Sigma \cup \Delta \cup Q}(X)$, t derives t' in one step ($t \vdash t'$) iff there is some production $q(\sigma^{(k)}(x_1, \ldots, x_k)) \rightarrow C[q_1(x_1), \ldots, q_k(x_k)]$, a context $D \in T_{\Sigma \cup \Delta \cup Q}(X_1)$, and terms $t_1, \ldots, t_k \in T_\Sigma$ such that

$$t = D[q(\sigma(t_1, \ldots, t_k)] \text{ and } t' = D[C[q_1(t_1), \ldots, q_k(t_k)]]$$

Macro Tree Transducers

A macro tree transducer further generalizes the top-down tree transducer by allowing states of any rank. The intuition is that the extra arguments to the state beyond the first represent the results of earlier transduction, and may be incorporated into the derived structure at any time. Formally, a macro tree transducer is a quintuple $M = \langle Q, \Sigma, \Delta, \rightarrow, q_0 \rangle$, where Σ and Δ are, as before, the input and output alphabets, but now elements of Q can be of any rank. The start state, $q_0 \in Q$ is of rank one. Productions in $\rightarrow$ are much the same as before, except that now there may be additional trees 'carried along'.

$$q^{(n+1)}(\sigma^{(k)}(x_1, \ldots, x_k), y_1, \ldots, y_n) \rightarrow t$$

The right hand side term, t, is a tree built up with output symbols, the variables $y_1, \ldots, y_n$, and contexts like $q'^{(r)}(x_i, t_1, \ldots, t_r)$, where $1 \leq i \leq k$, and $t_1, \ldots, t_r$ are trees built up in the same way as t.

An example of a relation computable macro tree transducer but not by a top-down tree transducer is the relation associating terms over $\Sigma = \{f^{(2)}, a^{(0)}, b^{(0)}\}$ with their yields expressed as terms over $\Delta = \{a^{(1)}, b^{(1)}, \epsilon^{(0)}\}$. For example, $f(a, f(b, a))$ would be associated with $a(b(a(\epsilon)))$. Let $M = \langle \{q_0^{(1)}, q^{(2)}\}, \Sigma, \Delta, \rightarrow, q_0 \rangle$, where $\rightarrow$ contains

$$q_0(x) \rightarrow q(x, \epsilon) \tag{1.1}$$

$$q(f(x_1, x_2), y) \rightarrow q(x_1, q(x_2, y)) \tag{1.2}$$

$$q(a, y) \rightarrow a(y) \tag{1.3}$$

$$q(b, y) \rightarrow b(y) \tag{1.4}$$

On input $f(a, f(b, a))$, M does the following.

$$\begin{aligned} q_0(f(a, f(b, a))) &\to q(f(a, f(b, a)), \epsilon) && \text{(by 1.1)} \\ &\to q(a, q(f(b, a), \epsilon)) && \text{(by 1.2)} \\ &\to q(a, q(b, q(a, \epsilon))) && \text{(by 1.2)} \\ &\to q(a, q(b, a(\epsilon))) && \text{(by 1.3)} \\ &\to q(a, b(a(\epsilon))) && \text{(by 1.4)} \\ &\to a(b(a(\epsilon))) && \text{(by 1.3)} \end{aligned}$$

Transducers with Regular Look-ahead

Finally, we can add to a transducer the ability to query an oracle about the subtree about to be transduced. We may imagine the transducer perched at the top of the tree, basing its next move on the results of its query about what lies below. We allow productions to have the form

$$q(\sigma(x_1, \ldots, x_k), y_1, \ldots, y_n) \to t,\ \langle p_1, \ldots, p_k \rangle$$

where $p_1, \ldots, p_k$ are the results of querying the oracle about trees $x_1, \ldots, x_k$ respectively. A transducer with look-ahead is deterministic iff there are no two productions with the same left-hand sides *and the same answers to the queries* but different right-hand sides. It is total iff for every symbol $\sigma^{(k)} \in \Sigma$, every $q^{(n+1)} \in Q$, and every sequence $p_1, \ldots, p_k$ of query answers there is a production which can be applied.

We represent our oracles as total deterministic bottom-up tree automata, and call transducers with such oracles *transducers with regular look-ahead*. A total deterministic bottom-up tree automaton is given by a triple $M = \langle P, \Sigma, \leftarrow \rangle$, where P is a finite set of states of rank one (the query answers), Σ is a ranked set

(the input alphabet), and $\leftarrow$ is a finite set of productions such that for $\sigma^{(n)} \in \Sigma$, and $p, p_1, \ldots, p_n \in P$ there is exactly one production of the following form in $\leftarrow$.

$$p(\sigma(x_1, \ldots, x_n)) \leftarrow \sigma(p_1(x_1), \ldots, p_n(x_n))$$

For example, taking $\Sigma = \{f^{(2)}, a^{(0)}, b^{(0)}\}$, the following bottom-up tree automaton tells us whether a term $t \in T_\Sigma$ has an even or an odd number of a-leaves. $M = \langle\{e, o\}, \Sigma, \leftarrow\rangle$, where $\leftarrow$ consists of

$$e(f(x_1, x_2)) \leftarrow f(e(x_1), e(x_2)) \tag{1.5}$$

$$e(f(x_1, x_2)) \leftarrow f(o(x_1), o(x_2)) \tag{1.6}$$

$$o(f(x_1, x_2)) \leftarrow f(o(x_1), e(x_2)) \tag{1.7}$$

$$o(f(x_1, x_2)) \leftarrow f(e(x_1), o(x_2)) \tag{1.8}$$

$$o(a) \leftarrow a \tag{1.9}$$

$$e(b) \leftarrow b \tag{1.10}$$

On input $f(a, f(b, a))$, M proceeds as per the following

$$\begin{aligned}
f(a, f(b, a)) &\rightarrow f(a, f(b, o(a))) && \text{(by 1.9)} \\
&\rightarrow f(a, f(e(b), o(a))) && \text{(by 1.10)} \\
&\rightarrow f(a, o(f(b, a))) && \text{(by 1.8)} \\
&\rightarrow f(o(a), o(f(b, a))) && \text{(by 1.9)} \\
&\rightarrow e(f(a, f(b, a))) && \text{(by 1.6)}
\end{aligned}$$

CHAPTER 2

Syntax and Semantics in Minimalist Grammars

This chapter presents a grammar for a fragment of English A-movement. 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 control 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

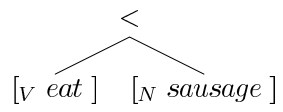
will suggest new ways to understand other aspects of other constructions. The fragments identified here abstract away from various complexities unearthed over fifty years of investigation into English and other languages. As such, there remain a great number of questions about how these myriad complexities 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

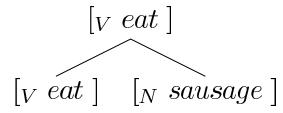
We start with the intuitive picture that words (or something like them) are the building blocks of sentences. Linguistic expressions (non-atomic ones, at least) can be broken up into subexpressions, which can then be recombined to form other expressions. We take as a basic syntactic operation one that 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 [38], 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 [90], 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 ')')).[1] 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.

(2.1) *John will devour.

(2.2) John will devour the ointment.

Something is amiss with sentence 2.1. 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

[1] 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 [90], 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:

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

eat, whose semantic object needn't be overt in the sentences in which it appears.

(2.3) John will eat.

(2.4) 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:[2]

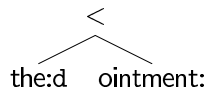
*Kwasi bedi.
Kwasi FUT.eat
"Kwasi will eat."

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

[2]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.

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 terms, the expression with the `=f` feature is the head, and the one with the `f` its dependent.[3] There are non-trivial correlations between the order of heads and dependents of various categories within a language [67]. 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

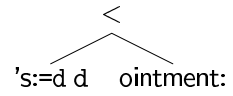


[3]In the idiom of the day we are assuming that all syntactic features are 'uninterpretable' in the sense that the only formal feature which is allowed to be present in a complete sentence is the categorial feature of the head of that sentence. The conception of grammar that naturally follows from this is of a *resource-sensitive* system [66, 143], which perspective is present also in the categorial type-logic community [125, 144].

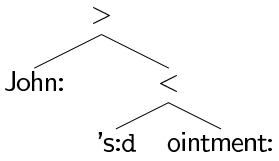
In English, if there are two dependents, the dependents flank the head. One example is the subject-verb-object word order of English, assuming that both subject and object are dependents of the verb. Another example is given by the Saxon genitive construction (2.5), assuming that the marker *'s* is the head of the construction, a two-argument version of *the*.

(2.5) 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 (2.5) in two steps. First, we merge *'s* and *ointment* to get



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

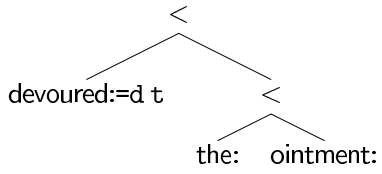


Note that it is crucial here that the syntactic features of an expression are checked in a paricular order—otherwise we couldn't distinguish syntactically between *ointment's John* and the above. We call the expression first merged with a lexical item its *complement*, and later merged expressions are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

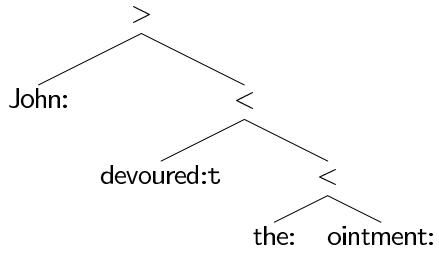
To derive a simple transitive sentence like (2.6),

(2.6) 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

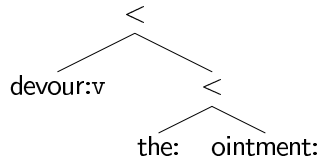


Now consider the following sentences

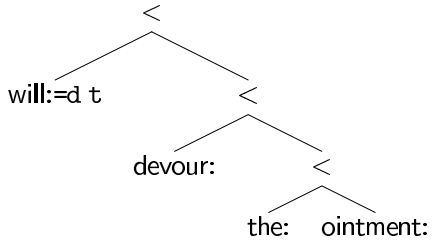
(2.7) John will devour the ointment

(2.8) John is devouring the ointment

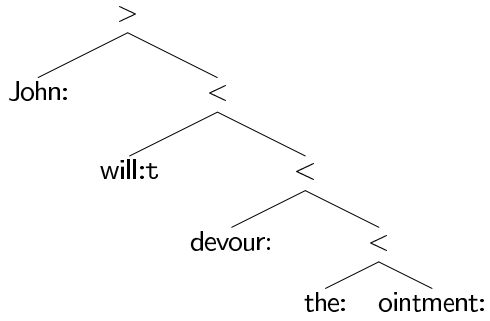
To generate the first of these (2.7) we need to add to our lexicon the expressions will::=v =d t and devour::=d v. Then we can merge *devour* with *the ointment*



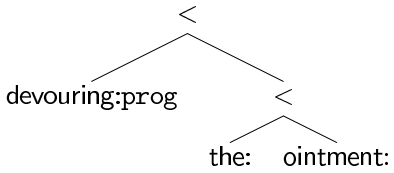
and then *will* with *devour the ointment*



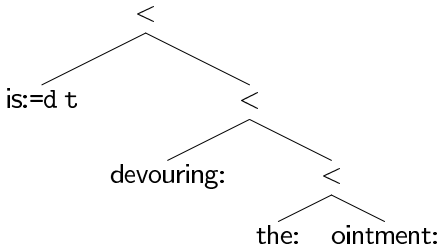
and finally *will devour the ointment* with *John* to get



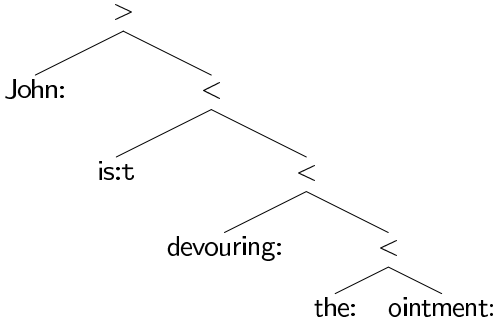
To generate sentence 2.8, we need to extend our lexicon again with the expressions is::=prog =d t and devouring::=d prog. Now we simply merge *devouring* with *the ointment*



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

(2.9) John devours the ointment

(2.10) John devoured the ointment

(2.11) John will devour the ointment

(2.12) John has devoured the ointment

(2.13) John had devoured the ointment

(2.14) John will have devoured the ointment

(2.15) John is devouring the ointment

(2.16) John was devouring the ointment

(2.17) John will be devouring the ointment

(2.18) John has been devouring the ointment

(2.19) John had been devouring the ointment

(2.20) John will have been devouring the ointment

To describe these twelve sentences, we might postulate the existence of the lexical items in figure 2.1. 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

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 2.1: The English auxiliary system (I)

just have one more, which is predictable given the comment above). An elegant description of the English auxiliary system, going back to [33], 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 [33] 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 [7, 167] to talk about movement of heads to other heads (figure 2.2).[4]

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

[4]Head movement does not commit us to a morphemic morphological theory (i.e. Item and Process or Item and Arrangement [77]). It is fully compatible with a Word and Paradigm approach [4, 146, 165]—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).

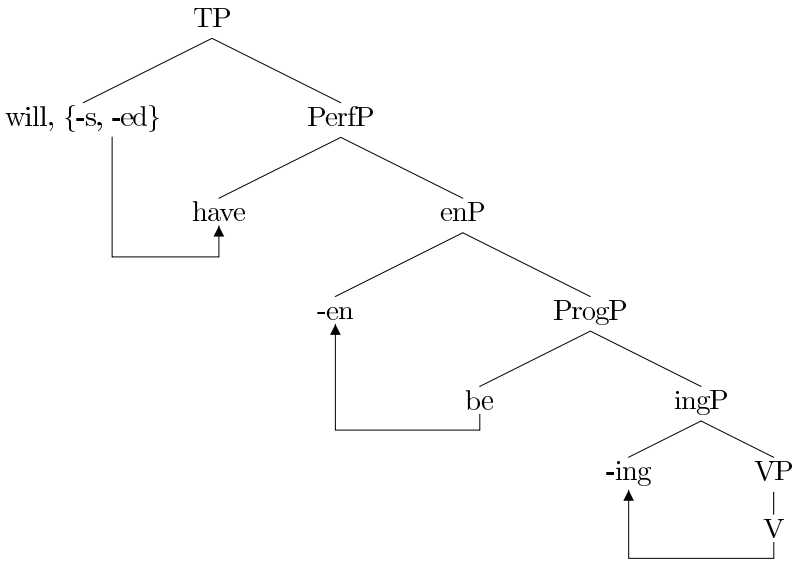


Figure 2.2: Head movement of auxiliaries in English

are defined over). Some have proposed to eliminate head movement from linguistic theory alltogether [113]. I mean to take no position on this issue here (but see [158] 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 [18, 29, 118]. 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 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 2.3). So it seems that we cannot leave

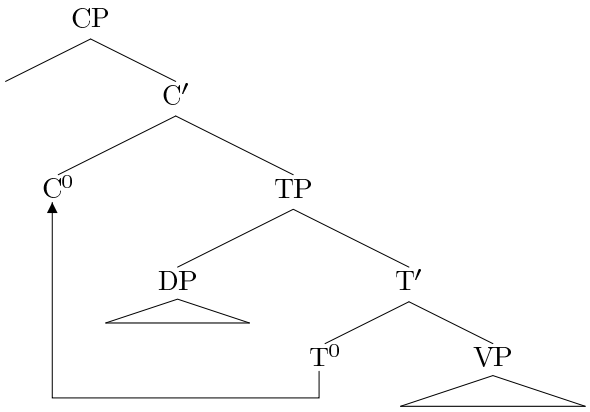


Figure 2.3: T-to-C movement

head 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 [157, 160]).

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.[5] 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

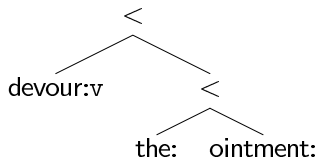
[5]Note that we are assuming that affixes are all structurally higher than the bases to which they attach. This is by no means a necessary assumption, but it not only seems to hold given our assumptions thus far about English, but also seems to be relatable to a cross-linguistic tendency for more 'abstract' meanings to be expressed via bound morphemes more frequently than more 'contentful' meanings [24]. 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.

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 2.4).[6] [7]

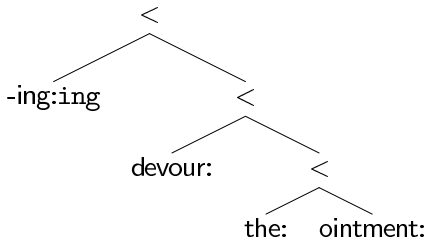
will::=perf =d t	have::=en perf	be::=ing prog	devour::=d v
-s::perf=> =d t	-en::=>prog en	-ing::=>v ing	
-ed::perf=> =d t	ϵ::=>prog perf	ϵ::=>v prog	

Figure 2.4: 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*



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



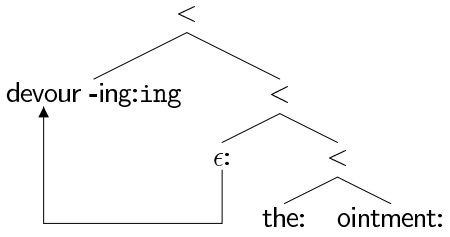
[6] Although now we generate terminal sequences like

John be -s kiss -ing Mary

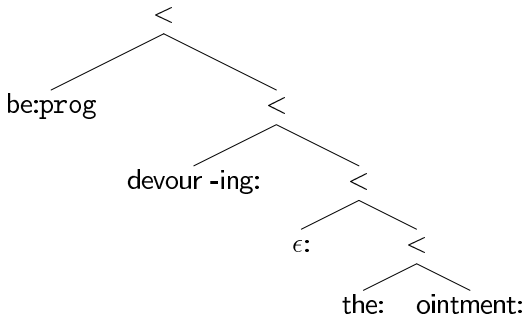
which is yet another step removed from the data. Thus, the apparent elegance of this approach needs to be evaluated in conjunction with the modifications necessary to the morphological module implicitly appealed to.

[7] The 'empty' lexical items in figure 2.4 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 2.1, which requires five entries for each verb in English).

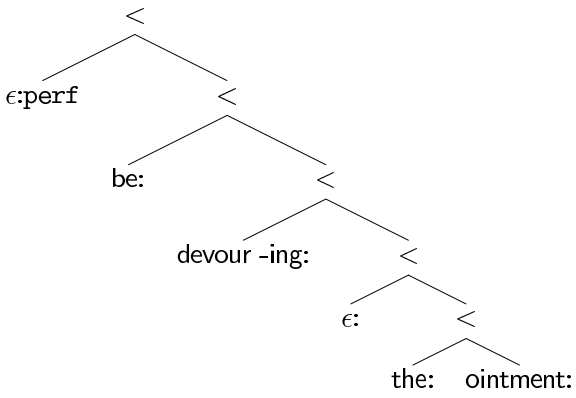
and the morphological properties of the merged affix trigger morphological readjustment—the head *devour* raises to the head *-ing*:



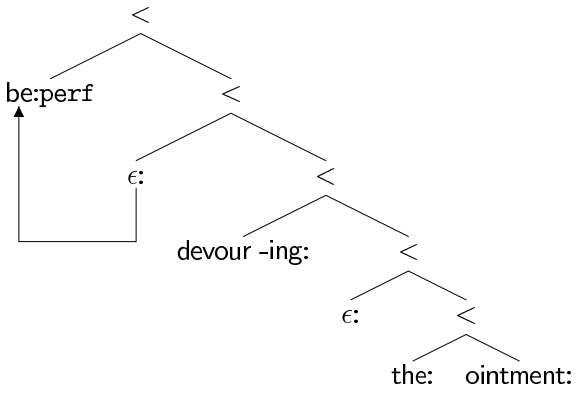
and *be* with *devouring the ointment*



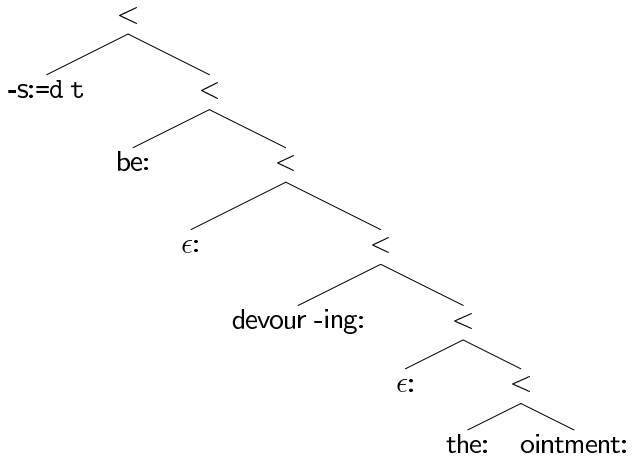
and ϵ::`=>prog perf` with *be devouring the ointment*



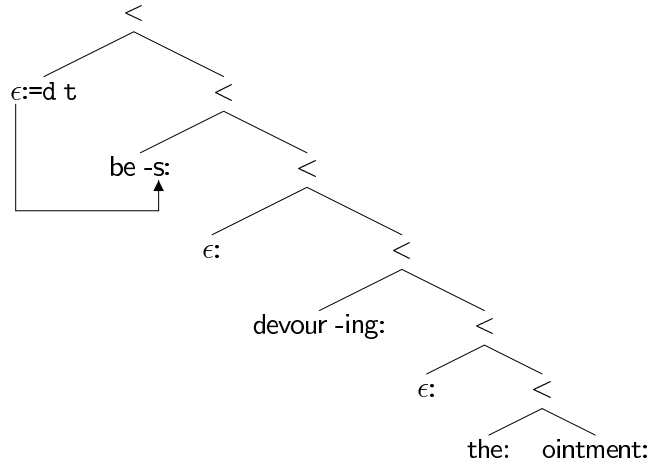
and again, the morphological properties of the merged affix trigger raising of the head *be* to the higher head:



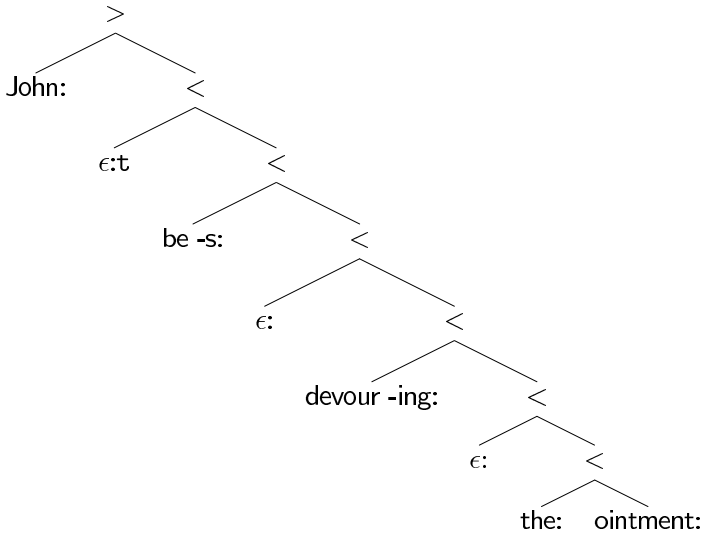
and *-s* with the above phrase



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 [118], 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 [52, 72]), and serves to illustrate some interesting properties of the copy mechanism introduced in chapter 4.

1.4 Phrasal Movement

Consider the following sentences.

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

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

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

(2.24) *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,[8] 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 [98, 115, 140], this is not as straightforward as our current theory would seem to predict.[9]

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

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

(2.27) There seemed to appear a pirate ship.

(2.28) *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

[8] 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.

[9] We defer a treatment of *there*-insertion to § 2.2.1 in chapter 3, when we investigate long-distance agreement.

accounting for these 'long-distance' dependencies in a simple and revealing way.[10] 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 [64] 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 [169] 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 2.5). We thus need to specify how the subsequent

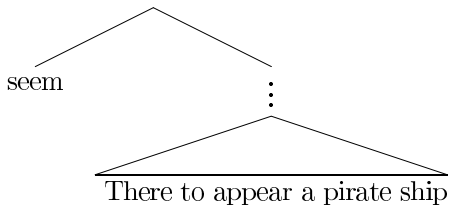


Figure 2.5: An intermediate stage in the derivation of 2.27

[10] 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.

'movement' of the subject occurs. As the movement seems to be to particular positions only, and of the subject only (cf (2.29) and (2.30)),[11] we conclude that movement is not free, i.e. there are restrictions governing it.

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

(2.30) *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.[12] 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 (2.31).

(2.31) *(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 [37] 'Extension Condition'). We adopt this here. We adopt also a general principle of 'immediacy', which (informally) requires that an expression move as soon as

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

[12] Relaxing this assumption (as done in [159]) 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.

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 [145] 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.[13] 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 [97] 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 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 2.6.

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 2.6: The English auxiliary system (III)

Then the sentences

(2.32) John has arrived.

[13] Constraints on movement can be incorporated into the domain of the move operation. Different constraints result in different generative power [63, 93, 120]. 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.

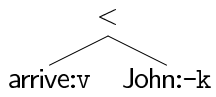
(2.33) John seems to have arrived.

(2.34) John seems to seem to have arrived.

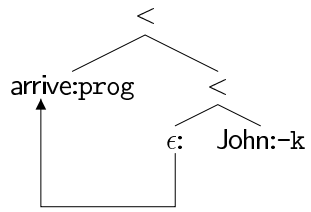
⋮

can be derived using the lexical items John::d -k, arrive::=d v, seem::=t v and the auxiliaries from figure 2.6.

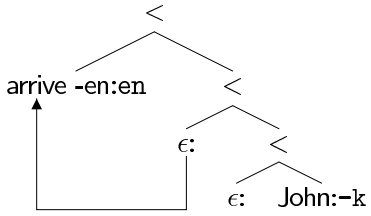
We derive the sentence *John has arrived* by first merging *arrive* and *John*,



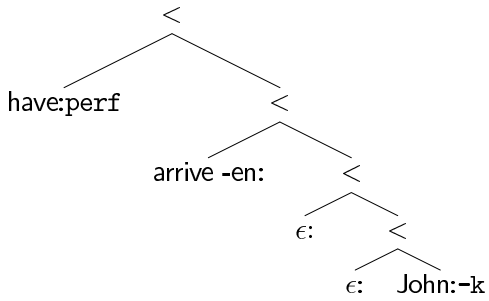
merging ϵ::=>v prog with the above expression (with the concommittant morphological readjustment)



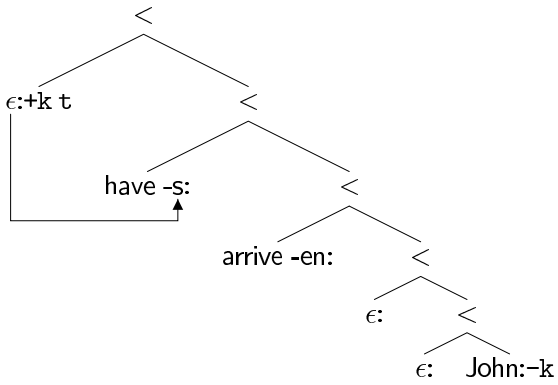
merging *-en* with the above (and resolving the morphological requirements of the merged affix)



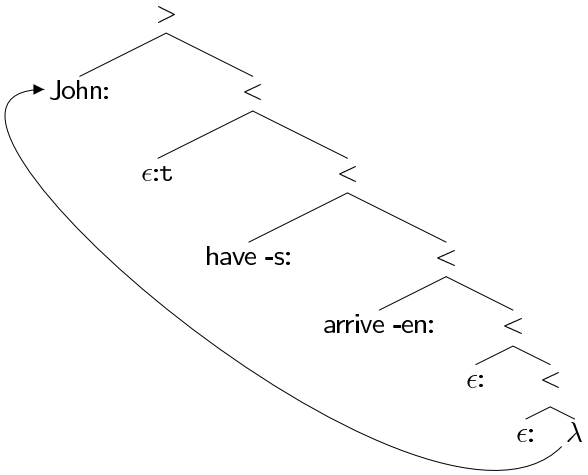
merging *have* with the above



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

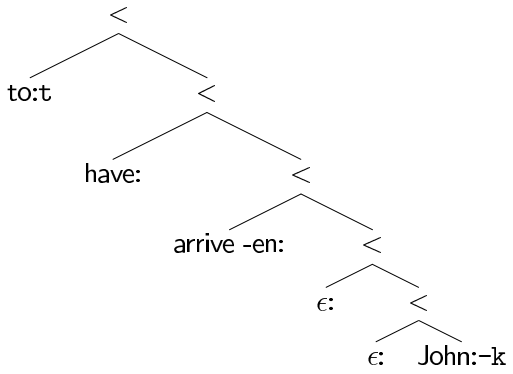


and then finally moving *John* in the above expression.[14]

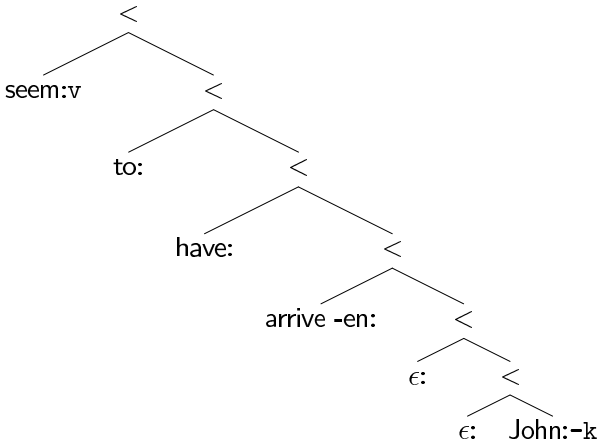


The sentence 'John seems to have arrived' can be derived by merging *to* with the expression *have arrived John* derived above

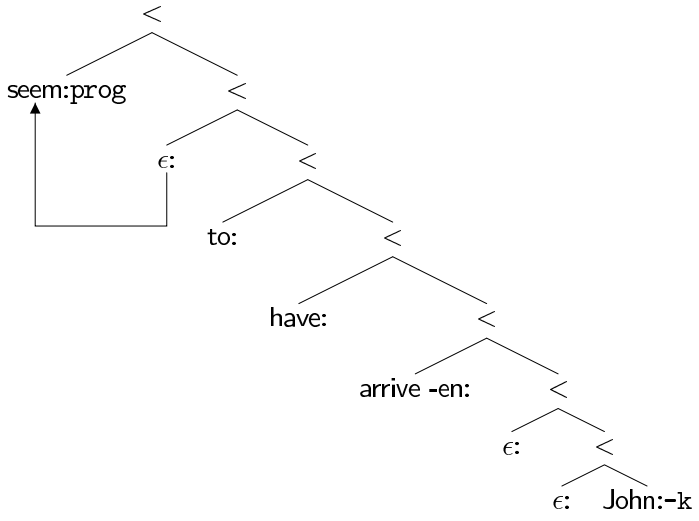
[14]For convenience, we represent the source of phrasal movements with a λ.



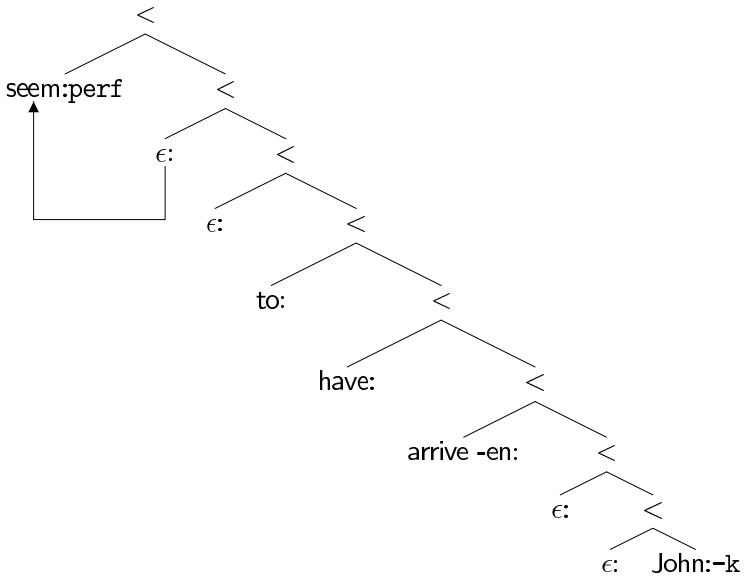
merging *seem* with the above



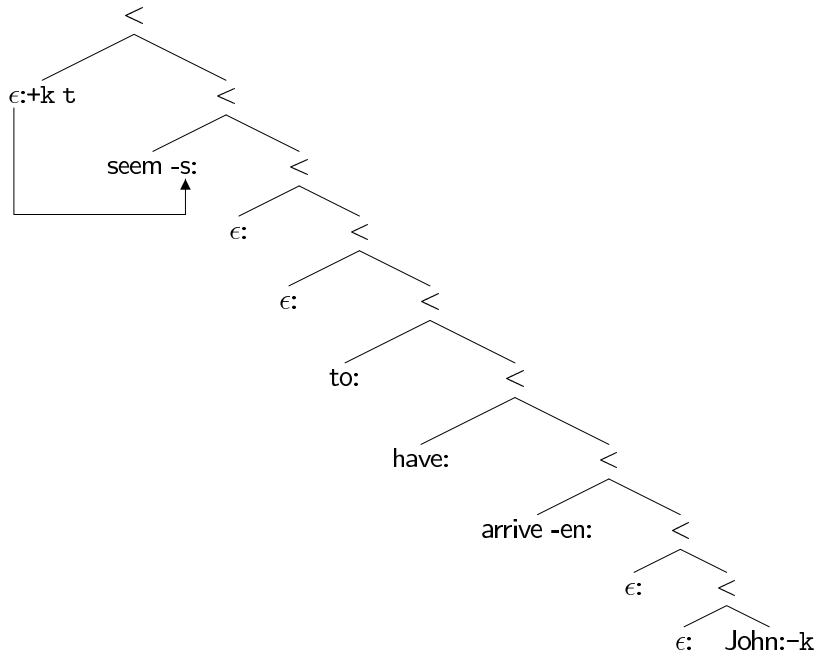
merging ϵ::=>v prog with the above expression (and raising the head of its complement)



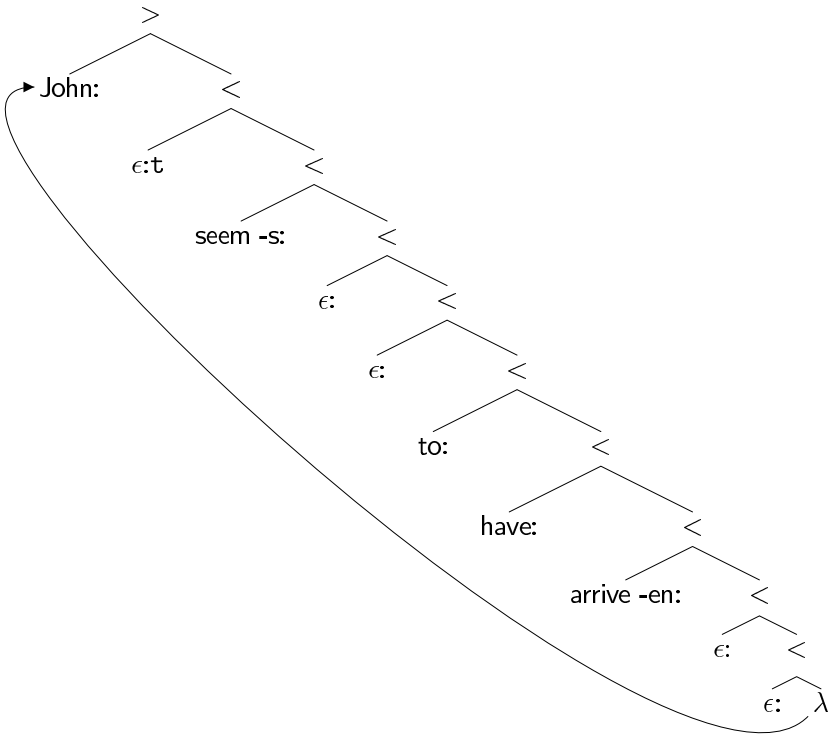
merging ϵ::=>prog perf with the expression above (and raising)



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 [129] 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.

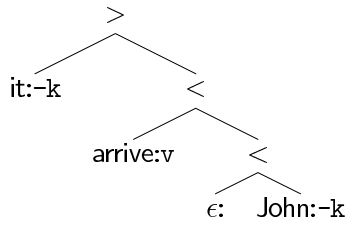
Within the context of our treatment of the raising to subject construction, our principle of immediacy allows for a simple account of the impossibility of so-called super-raising in English (as in 2.37 below). As is well known, raising sentences (such as 2.33) have non-raised counterparts (such as 2.35) in which the subject position of the matrix clause is filled by an expletive *it*.

(2.35) It seems John has arrived.

(2.36) *It seems John to have arrived.

(2.37) *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`.[15] A derivation for the super-raising sentence in 2.37 would go through an intermediate form such as the below (merging ϵ::`=>v =expl v` and then *it* with *arrive John* from above)



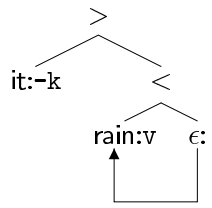
which will run afoul of our principle of immediacy as soon as a tense head is merged (in the matrix clause in 2.36, and in the embedded clause in 2.37). The grammatical sentence 2.35 is derived by merging the expletive into the structure after the lower clause subject has checked its `-k` feature (i.e. once *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 2.38 below.

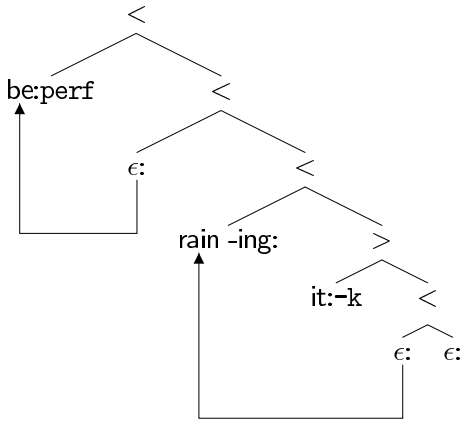
(2.38) It is raining.

[15]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 12).

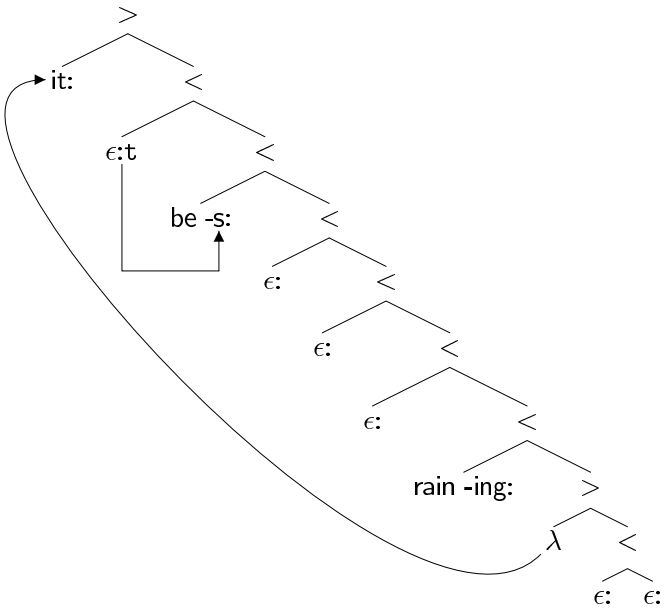
Including the expression rain::v into our lexicon, we derive the sentence *it is raining* in seven steps. We first merge ϵ::=>v =expl v with *rain*, and then continue by merging the resulting expression with *it*.



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.[16]



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.

(2.39) John will have been devouring the ointment.

(2.40) The ointment will have been being devoured.

(2.41) George expects the ointment to be devoured.

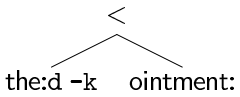
(2.42) The ointment is expected to be devoured.

[16]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).

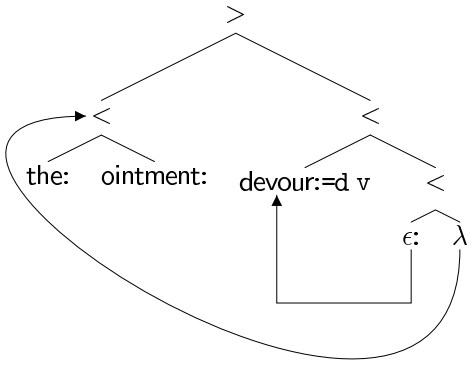
The pair 2.39 and 2.40 suggest to us that the external argument (here, the agent *John*) can be dissociated from the verb, and that, in that case, the internal argument moves to the canonical subject position to check its `-k` feature. In other words, the possibility of checking a `-k` feature internal to the verb phrase is dependent upon whether the external arugment is present. This is essentially Burzio's generalization [22]:

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

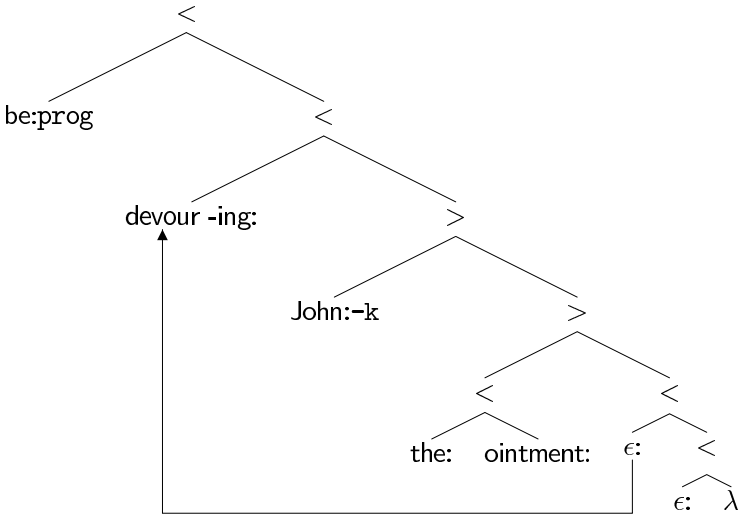
In a basic transitive sentence like 2.39, our principle of immediacy forces us to check the `-k` feature of the object before the subject is introduced (a position familiar from [94, 97]). We assign to the lexeme "devour" the type `=d V`, and recast Burzio's generalization in our system with the lexical item ϵ::`=>V +k =d v`. Then 2.39 can be derived by first merging *the* and *ointment*.



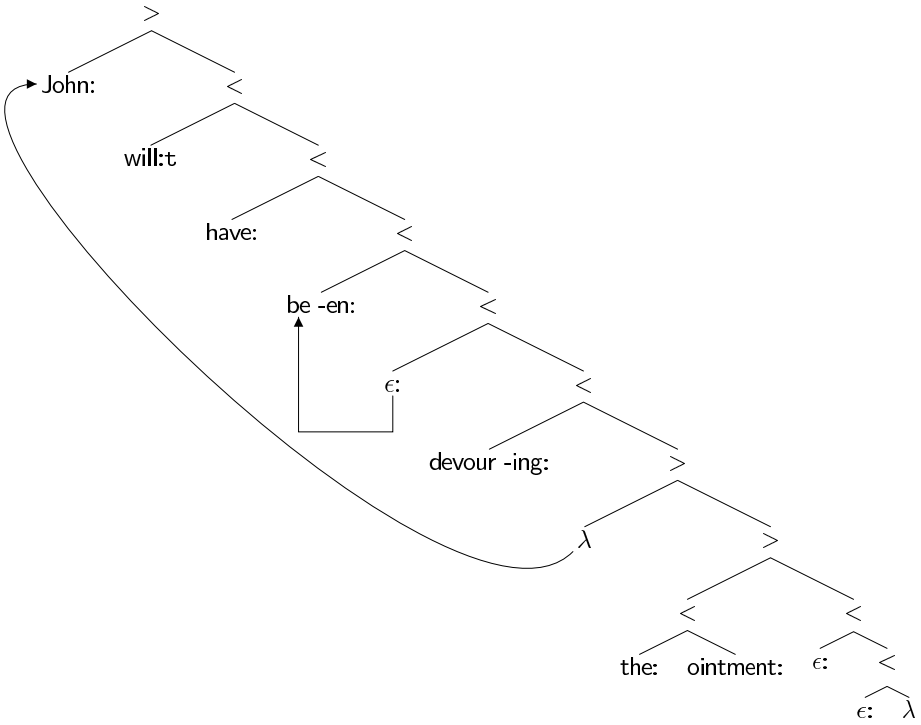
Then *devour* is merged with *the ointment*, and then ϵ::`=>V +k =d v` with the resulting expression (with the appropriate morphological readjustments), followed by movement of *the ointment*.



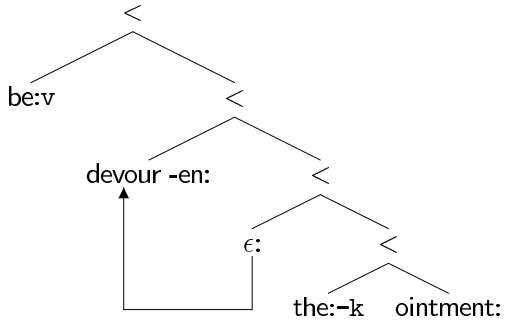
We continue by merging the expression above with *John*, and then merging *-ing* with the result, and then *be* with the result of that.



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.

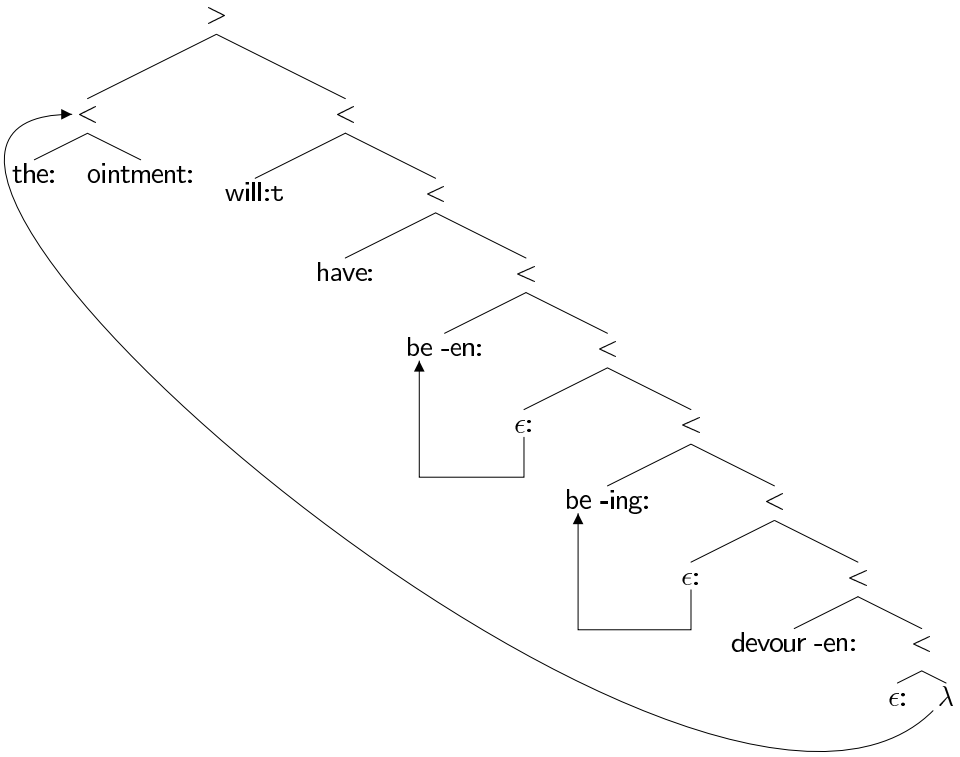


Sentence 2.40, with its passive 'be' and its associated perfective participial morphology, can be nicely accomodated with our existing head movement operation. We add the expressions `be::=pass v` and `-en::=>V pass` to our lexicon. Sentence 2.40 can then be derived by first merging passive *-en* with the expression *devour the ointment* derived above followed by merging passive *be* with *devoured the ointment*.

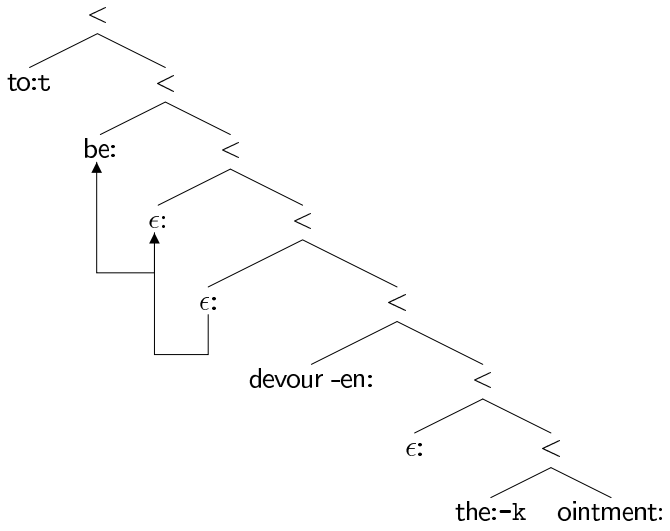


We then successively merge *-ing*, *be*, *-en*, *have* and *will* (performing the necessary morphological operations when appropriate). Finally, we move *the ointment* in

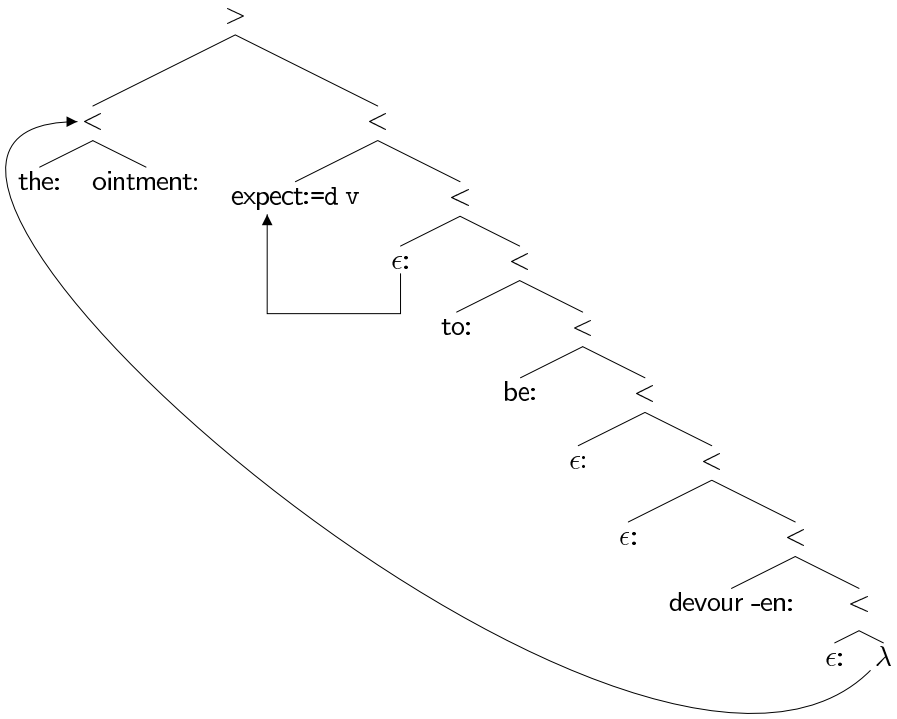
the resulting expression.



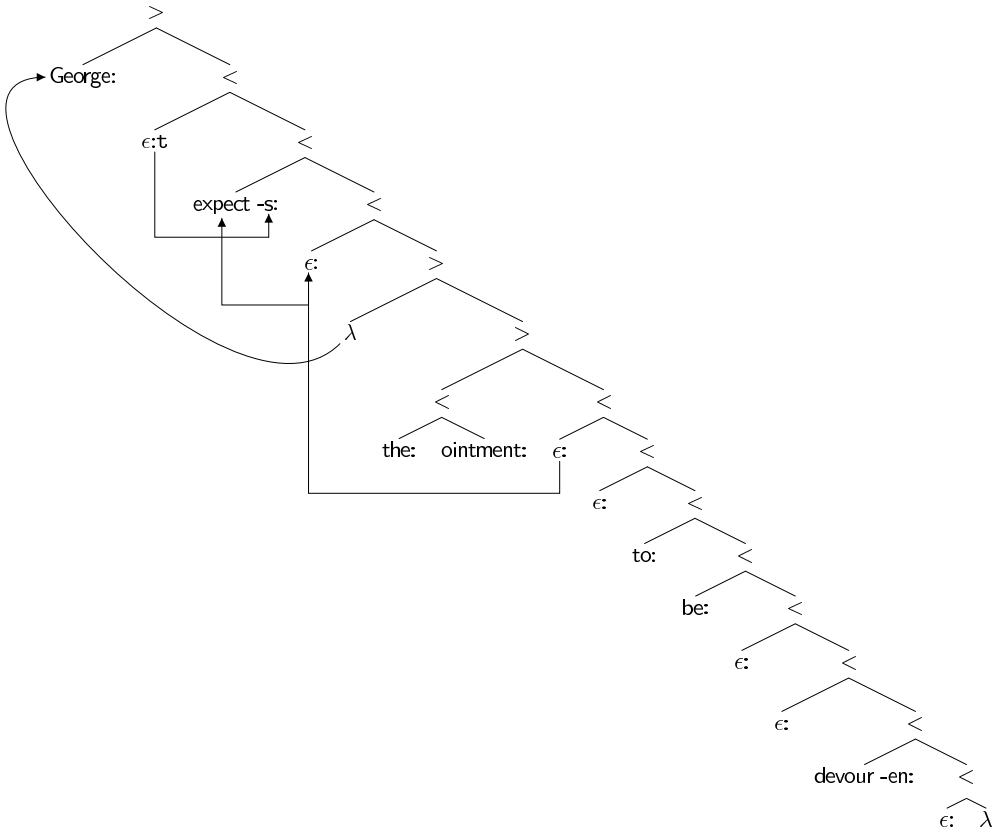
To derive sentences 2.41 and 2.42 we need only add the lexical item expect::=t V. Note that, descriptively, we can have raising to object, and then passivization, and then more raising without any additional stipulations—raising and passive feed each other. Formally, object to subject movement in passives is motivated by the same thing that motivates raising in raising constructions. Sentence 2.41 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 [137]).



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

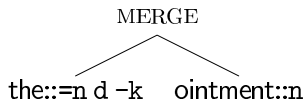
We have not discussed many architectural options that are or have been hot topics in the minimalist tradition. Nothing has been said about numerations, lexical subarrays, preferences for merge over move, trans-derivational economy constraints, covert movement, sidewards movement, adjunction, scrambling, the EPP, or Agree.[17] 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

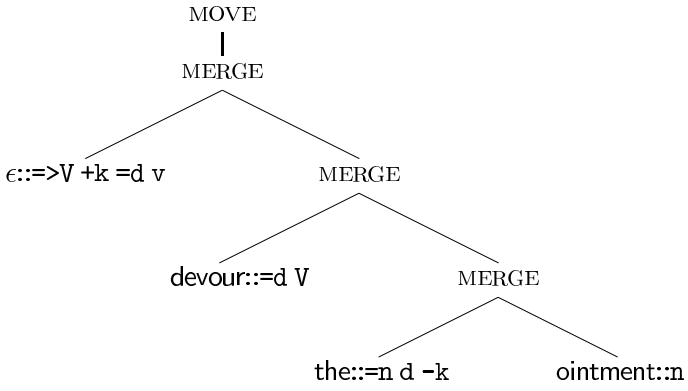
Recall that our (immediate) goal is to specify the associations between the pronounced form of a sentence on the one hand and the range of interpretations that it may have on the other. The way we have approached this task of defining a relation is by defining a set of abstract objects, together with 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

[17]Some of these have been explored within the context of the assumptions made here elsewhere. In particular, Stabler [159, 160] discusses covert and sidewards movement, and Frey and Gärtner [59] discuss adjunction and scrambling. Covert movement and Agree are discussed and formalized in chapter 3.

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:



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 a tree

The set of possible derivations defined above includes those that do not stand for any well-formed expression (e.g. $\textsc{move}(the)$). Derivations that do stand for well-formed expressions we will call successful, or convergent. Those that do not we will say have crashed.[18] 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 16).

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 2.7.[19] The procedural metaphor dominating much 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

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

[19]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.

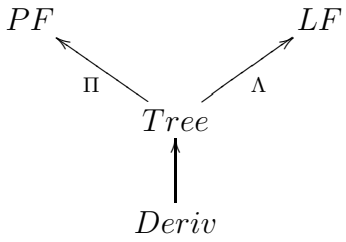


Figure 2.7: Specifying the associations between sounds and meanings

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 2.7 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 [31] 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 [117], given the possibility of iterated adjunction (of, say, relative clauses, adjectives, or adverbs) and of iterated raising constructions, the window of 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). 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 2.8), 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[...] ([163], pp *xi*)

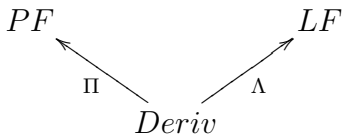


Figure 2.8: Directly interpreting derivations

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 [74] 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" ([21], 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.

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 2.9. These expressions are indistin-

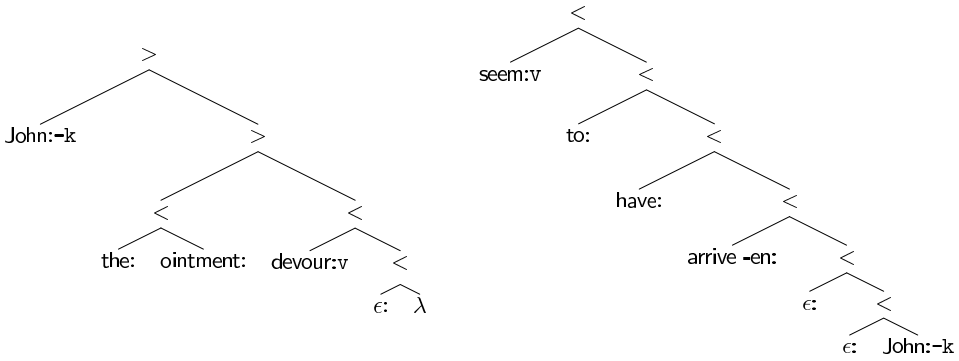


Figure 2.9: Syntactically indistinguishable expressions

guishable 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

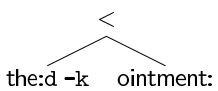
tree t whose heads have a licensee feature `-x`, and remove these constituents from t (in case a constituent whose head bears a licensee feature itself containts another constituent with the same property, we remove this latter constituent from the former and put it into the list). For each sub-tree 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`.

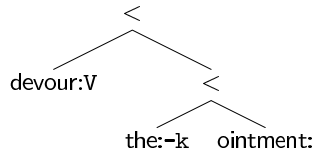
We work through a derivation of the sentence "John devoured the ointment," showing at each step both the derived tree, as well as its abbreviation according to our convention. The crucial fact of note is that the operations of merge and move can be *directly defined* over these reduced expressions, rendering the derived tree unnecessary for the computation of a PF legible representation. Precise definitions of the generating functions are given in appendix B–1.2.

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



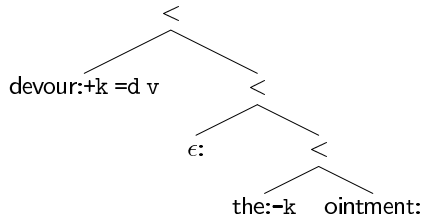
$(\epsilon,$ the, ointment$)$: d -k

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



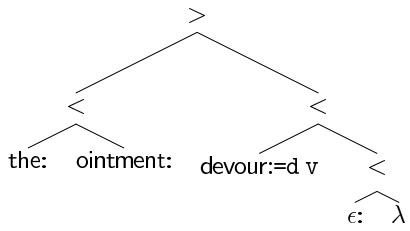
$(\epsilon,$ devour, $\epsilon)$: V, (the ointment, -k)

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



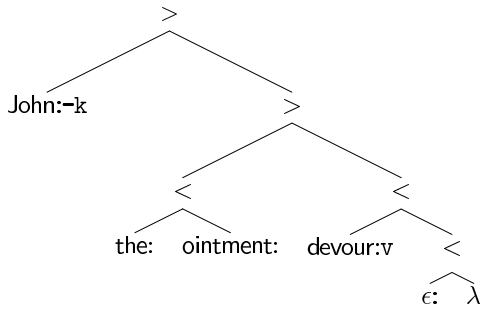
$(\epsilon,$ devour, $\epsilon)$: +k =d v, (the ointment, -k)

Move applies to the expression thus derived, to get



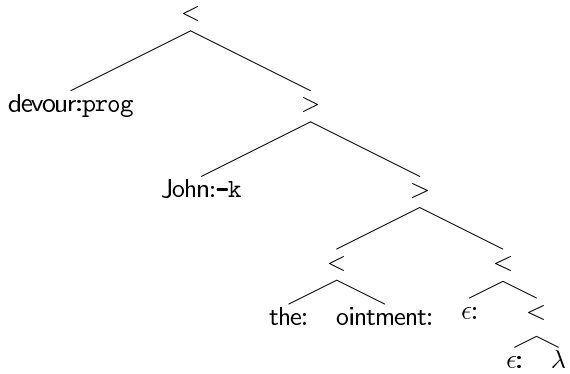
(the ointment, devour, ϵ) : =d v

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



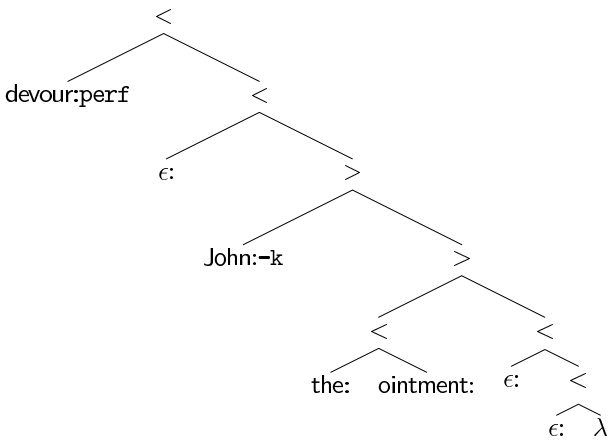
(the ointment, devour, ϵ) : v, (John, -k)

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



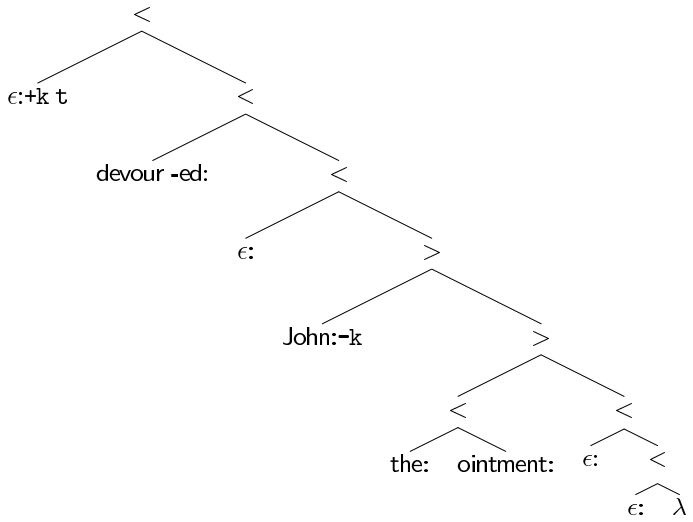
(ϵ, devour, the ointment) : prog, (John, -k)

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



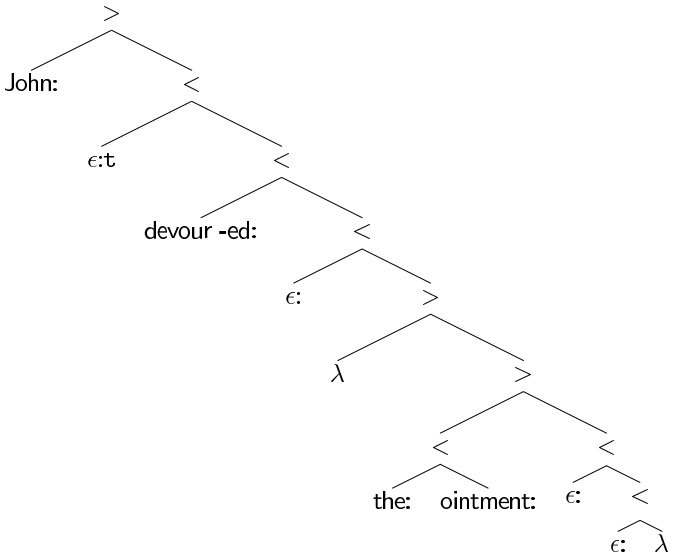
(ϵ, devour, the ointment) : perf, (John, -k)

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.[20] 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 2.10, 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 2.10.

$$(\epsilon,\ \mathsf{to},\ \mathsf{have\ arrive\ \text{-}en}) : \mathtt{t},\ (\mathsf{John},\ \mathtt{-k})$$

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 move-

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

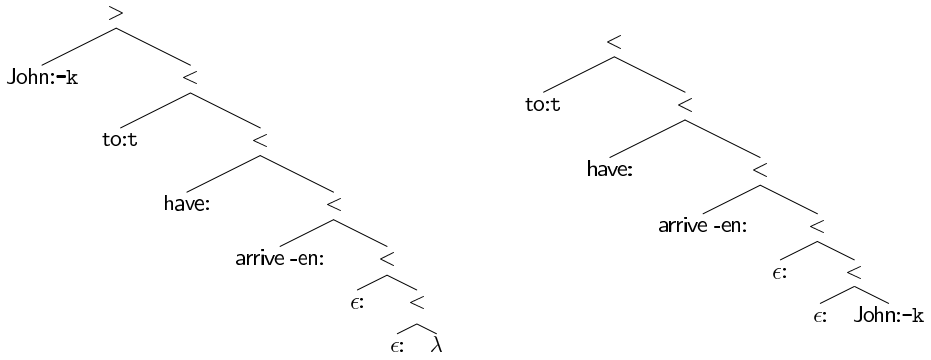


Figure 2.10: Two accounts of the structure of the non-finite TP

ment, 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[21] define the rules of inference for the 'language of thought'. (See [60, 168, 170] 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 [45], 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).

[21] With respect to the closure properties of the relevant transformations.

2.3 Semantics in Chains

Sentences like 2.43 are commonly thought to be compatible with two seemingly different states of affairs.

(2.43) 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 2.44.

(2.44) 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 2.44 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 2.43 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 2.11. If, on the other hand, we imagine 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

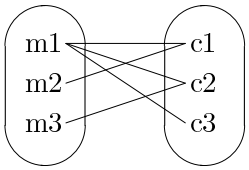


Figure 2.11: A model for the subject-wide scope reading of sentence 2.43

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 2.12.

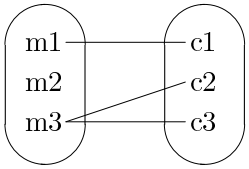
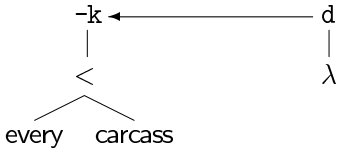


Figure 2.12: A model for the subject-narrow scope reading of sentence 2.43

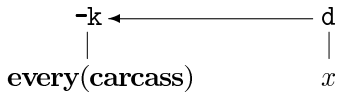
Quantified noun phrases (QNPs) are usefully thought of as making two general meaning contributions to the clauses in which they appear. The first is the role they play in the clause (i.e. is *exactly one maggot* the devourer or the devouree). The second is their logical priority with respect to other elements (as Hintikka [75] calls it). For example, the two readings of sentence 2.43 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 [156]).



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 [166]

(and thus sentences denote sets of satisfying assignments, as in [99]). 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 [79], 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 [78, 80] 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.

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

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

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

$$\text{for each } i \in \mathbb{N},$$

$$\lambda_i : [G \to T] \to [G \to E] \to G \to T$$

$$\lambda_i(H)(f)(g) = \textbf{true} \text{ iff there is some } h \in H \text{ such that}$$

$$h \approx_i g \text{ and } f(g) = h_i$$

$$\mathtt{G} : [\alpha \to \gamma] \to [\beta \to \alpha] \to \beta \to \gamma$$

$$\mathtt{G}xyz = x(yz)$$

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 $=$, &, and $\neg$, 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. $(t_1 = t_2) \in \mathbf{Term_{Bool}}$, 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 $(\mathsf{x_0} = \mathsf{0''}) \in \mathbf{Term_{Bool}}$, which we can calculate as follows.

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

That is, $[\![(\mathsf{x_0} = \mathsf{0''})]\!]_{\mathcal{M}}$ is the set of all assignments which assign the value $\sigma(\sigma(0))$ to the index 0. The denotation of $(\forall \mathsf{x_0})(\mathsf{x_0} = \mathsf{0''}) \in \mathbf{Term_{Bool}}$ is given as a function of the denotation of this subformula.

$$\begin{aligned} g \in [\![(\forall \mathsf{x_0})(\mathsf{x_0} = \mathsf{0''})]\!]_{\mathcal{M}} \text{ iff for every } f \in [G \to E] & \text{ there is an } h \in [\![(\mathsf{x_0} = \mathsf{0''})]\!]_{\mathcal{M}} \\ \text{such that } h \approx_0 g \text{ and } & f(g) = h_0 \\ \text{iff } f(g) = \sigma(\sigma(0)) \text{ for every } & f \in [G \to E] \end{aligned}$$

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

Application and Abstraction

Now we add another type to our language, **Num** $\rightarrow$ **Bool**, along with the symbol $\curlywedge$. The set of terms of type τ is expanded as follows.

8. $(\curlywedge x)\phi \in \mathbf{Term}_{\mathbf{Num}\rightarrow\mathbf{Bool}}$ if $\phi \in \mathbf{Term}_{\mathbf{Bool}}$ and $x \in \mathbf{Var}$
9. $(\alpha(t)) \in \mathbf{Term}_{\mathbf{Bool}}$ if $\alpha \in \mathbf{Term}_{\mathbf{Num}\rightarrow\mathbf{Bool}}$ and $t \in \mathbf{Term}_{\mathbf{Num}}$

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

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

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

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

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

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 [92]), 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.[22] 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.

[22]Others ([45]) 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.

The Basics

We begin by considering simple intransitive sentences like 2.45 below.

(2.45) 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 [74] 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 possible mode of semantic combination associated with an instance of merger (see figure 2.13). 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 2.45 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 2.13), 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 2.13: Modes of Semantic Combination (I)

Quantifiers in Object Positions

Turning now to transitive sentences like 2.46 with quantificational DPs in the object position, we run into familiar problems.

(2.46) 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 [91], 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 [124], 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 [45]. 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 2.14).

		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 2.14: Modes of Semantic Combination (II)

The calculation of the interpretation of sentence 2.46 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 2.15)). 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 2.15), yielding

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

Our final modes of semantic combination are as schematized in figure 2.15 (for precision see appendix B–1.3). The $\mathcal{Q}$ in the figure represents the stored meaning of the moving constituent.

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

	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)
$[\![move(\alpha)]\!] \rightarrow [\![\alpha]\!]$	$store(\alpha)$	(Id)
$[\![move(\alpha)]\!] \rightarrow \mathcal{Q}([\![\alpha]\!])$	$store(\alpha) - \mathcal{Q}$	(Retrieve)

Figure 2.15: Modes of Semantic Combination (III)

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 2.47 below, in which a QNP in a lower clause checks its `-q` feature in the higher clause.

(2.47) *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 2.47 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 2.48 below, we need to at least allow all DPs, quantificational or not, to bear a `-q` feature.

(2.48) John devoured George.

Again, we need to ensure that sentences like the ungrammatical below (2.49) are not generated.[23]

(2.49) *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 [160]) 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.[24] 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. (We investigate covert movement in chapter 3.) 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 [78, 80] proposes to do away with the standard quantifier raising (QR) operation (an operation usually conceived of as being different 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

[23]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.

[24]As in sentences like

1. 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*.

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.

(2.50) 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:[25]

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)

[25]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 [19] and Minimal Recursion Semantics [46], among others). Here, treating with Cooper [45] the semantic scope of DPs as (partially) independent of the syntactic derivation, our derivation tree *is* an underspecified semantic representation.

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 2.50 is calculated from the derivation above in the following manner (the notation α; $\boxed{\beta}$ represents 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 2.50 in the following way. Note that we may retrieve the stored function either in step 9 or 10.[26]

[26]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–2 (viewing object level substitutions as instructions to systematically permute assignment functions).

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 raising. For example, there is a reading of sentence 2.51 (2.51.ii) according to which the object is logically prior to the raised subject.

(2.51) Something seems to be devouring everyone.

1. (it seems that) there is a particular entity (Grendel, say), such that that entity is devouring everyone.
2. (it seems that) for each person, there is some entity or other that is devouring him.

The existence of unraised equivalents to 2.51 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.[27] Our useful functions (e.g. **every** and λ_i) are given the obvious reinterpretations.

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

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

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 2.51.

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

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 between feature driven and successive cyclic movements. If we wanted to allow for

[27] Treating possible worlds in this manner (as just a richer set of truth values) entails that names are "rigid designators" [100] as they have no world parameter.

the possibility of QNPs to take scope in intermediate, successive cyclic positions (which has been argued against in the case of A-movement [106, 129, 137]),[28] we could simply allow retrieval to apply freely throughout the derivation, making our proposal even more similar to Cooper's [45] original one.[29] 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.

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

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.[30] 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.'[31] 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 2.52 below is roughly synonymous with (the subject narrow scope reading of) the active 2.53 with an existentially quantified

[28]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 B–1.2).

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

[30]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.

[31]We will come back to this in chapter 3.

subject.

(2.52) Everyone was devoured.

(2.53) 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 assign to the passive voice head the same denotation as we assign to the QNP *something*. The lack of an 'inverse scope' reading in 2.52 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 2.52 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 2.54 below.

(2.54) 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 2.54, 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.

(2.55) John seemed to shave an abbot.

(2.56) John wanted to shave an abbot.

(2.57) George expected John to shave an abbot.

(2.58) 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 2.55 and 2.57 entail the passivized lower-clause versions 2.59 and 2.61 below, 2.56 and 2.58 do not.

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

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

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

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

Furthermore, expletive subjects are permitted in the raising clauses 2.63 and 2.65 but not in 2.64 or 2.66.

(2.63) It seemed to be raining.

(2.64) *It wanted to be raining.

(2.65) George expected it to be raining.

(2.66) *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 [78, 80, 111, 114, 134], 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.[32] 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 (2.67), we decide to allow for asymmetric feature checking.

(2.67) 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.

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

[32] 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.

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.

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

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 2.68 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.

(2.68) 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`.[33] 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

[33]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.

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) [148], 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)

   $$(\epsilon,\ \mathsf{a},\ \mathsf{abbot}) : \mathtt{*d\ -k\ -q}$$

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

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

   $$(\epsilon,\ \mathsf{shave},\ \epsilon) : \mathtt{V},\ (\mathsf{a\ abbot},\ \mathtt{-k\ -q})$$

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

3. merge(ϵ::=>V +k =d +q v, 2)

   $$(\epsilon,\ \mathsf{shave},\ \epsilon) : \mathtt{+k\ =d\ +q\ v},\ (\mathsf{a\ abbot},\ \mathtt{-k\ -q})$$

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

4. move(3)

   $$(\epsilon,\ \mathsf{shave},\ \epsilon) : \mathtt{=d\ +q\ v},\ (\mathsf{a\ abbot},\ \mathtt{-q})$$

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

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)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1})$, $\boxed{\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$

$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

23. merge(ϵ::=>v prog, 22)

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

(every barber, −k −q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1})$, $\boxed{\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$

$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

24. merge(ϵ::=>prog perf, 23)

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

(every barber, -k -q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1})$, $\boxed{\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$

$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

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)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1})$, $\boxed{\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$

$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

26. move(25)

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

(every barber, -q)

$\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1})$, $\boxed{\mathtt{G}(\mathbf{every}(\mathbf{barber}))(\lambda_1)}$

$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

27. move(26)

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

$\mathbf{every}(\mathbf{barber})(\lambda_1(\mathbf{promise}(\mathbf{g})(H)(\mathbf{x_1}))$

$H = \mathbf{some}(\mathtt{G}(E))(\mathbf{persuade}(\mathbf{some}(\mathbf{abbot})(\lambda_0(\mathbf{shave}(\mathbf{x_0})(\mathbf{x_1})))(\mathbf{x_1}))$

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 [104]). 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. 2.69, 2.70), they allow for passivization when their clausal complement is finite (cf. 2.71, 2.72).

(2.69) George promised John to arrive on time.

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

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

(2.72) 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 B–1.2). 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 2.69 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 2.69), we now overgenerate wildly, predicting the existence of control through finite clauses.

1. *to arrive every barber*

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

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

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

   $$(\epsilon, \text{promise}, \epsilon) : \texttt{=t V},\ (\text{George}, \texttt{-k -q})$$

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

3. merge(2, 1)

   $$(\text{to arrive}, \text{promise}, \epsilon) : \texttt{V},\ (\text{George}, \texttt{-k -q}),\ (\text{every barber}, \texttt{*d -k -q})$$

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

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 2.74 alongside the grammatical 2.73.

(2.73) Every barber wanted to arrive.

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

(2.74) *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 2.75).

(2.75) *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 2.76).

(2.76) *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 2.78.

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

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

(2.78) 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)

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

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)

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

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.[34]

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 2.79 and 2.80 on the one hand, and 2.81 and 2.82 on the other.

(2.79) John promised George to shave an abbot.

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

(2.81) John persuaded George to shave an abbot.

(2.82) 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.

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

[34]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 B–1.2.

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

(2.85) George hoped to shave an abbot.

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

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

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

As we were able to account for the distinction between 2.79 and 2.80 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 2.83 and 2.84. 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`.[35] As some expressions select for clausal complements, irrespective of their tensedness, we express that finite clauses are also clauses with the lexical item[36]

```
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*

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

1. It rained.
2. *To rain.

[36]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.

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 2.85 and 2.87, and correctly rules out 2.86. 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 2.88 while still correctly ruling out an ECM variant of 2.87). 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 2.88. 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 2.16. Although our type system is not strong enough to premit us to 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 [5, 88], this fact is tantalizingly suggestive!

3 Summary

In this chapter, we have introduced the version of minimalism (minimalist grammars [48, 59, 63, 73, 119–121, 156–162]) 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

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 2.16: A Grammar for English A-movement

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 [69]) 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.[37] 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

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

e.g. [56] 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 [16] 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 [32] calls (copying via) 'internal merge' and 'external merge.' These two 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.

Appendices

This chapter's appendices are organized as follows. In appendix B–1 a formal foundation for the syntactic theory appealed to in this chapter is provided. In appendix B–1.1, 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 B–1.2 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, [151]) and thus to these treeless variants is presented in [73, 119, 121], and will not be reproduced here. Next (B–1.3) 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–2, 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]).

B–1 Definitions

B–1.1 Minimalist Grammars on Trees

B–1.1.1 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 $s \in proj_R(t)$, $r \lhd_\tau^* s$).

B–1.1.2 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

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

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 $\langle \epsilon, \epsilon \rangle$.

B–1.1.3 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 $\mu : N \rightarrow L$ 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 [90] 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 [160]. 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 § B–1.1.1 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.

B–1.1.4 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.

B–1.1.5 Minimalist Grammars

A minimalist grammar (MG) is a five-tuple $G = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \mathtt{c} \rangle$, where Σ, **lic**, and **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 $\mathtt{c} \in$ **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

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

B–1.2 Minimalist Grammars without Trees

The trees of appendix B–1.1 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 [121]). Instead of trees, it has become standard to view minimalist expressions as sequences of categorized strings. Given a minimalist grammar $G = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \mathtt{c} \rangle$, 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 $l \in \mathbf{lic}$, $\mathtt{+l}, \mathtt{-l} \in \mathbf{Syn}$
   (b) for $s \in \mathbf{sel}$, $\mathtt{=s}, \mathtt{=>s}, \mathtt{s=>}, \mathtt{s}, \mathtt{*s} \in \mathbf{Syn}$
   (c) nothing else is in **Syn**
3. $\mathtt{c} \in \mathbf{selectee}$ 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 $i \in I := \Sigma^* \times \Sigma^* \times \Sigma^* \times \{:, ::\} \times \mathbf{Syn}^*$, and $N \in N := (\Sigma^* \times \mathbf{Syn}^+)^*$. 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 $E := I \times N$. 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 $\alpha, \beta \in N$ satisfying

(IMM1) no element of α or β has `*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 $\alpha, \beta \in N$ satisfying:

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

$$\frac{(s_1, s_2, s_3) : \texttt{+f}\gamma, \alpha(t, \texttt{-f})\beta}{(ts_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1}$$

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

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 $\alpha, \beta \in N$ satisfying

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

$$\frac{(s_1, s_2, s_3) \cdot \texttt{=f}\gamma, \alpha \qquad (t_1, t_2, t_3) : \texttt{*f}\delta, \beta}{(s_1, s_2, s_3) : \gamma, \alpha(t_1t_2t_3, \texttt{*f}\delta)\beta}\text{cmerge}$$

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

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

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 [39] '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 $G = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \mathtt{c} \rangle$ is defined to be

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

B–1.3 A Semantics for MGs

Just as we associated a string (more generally, a term in a phonological algebra) with a syntactic derivation in appendix B–1.2, 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 `G` (interpreted as the geach combinator), `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 B–1.2, 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 $i \in I := T_M \times \{:, ::\} \times \mathbf{Syn}^*$, and $N \in N := (T_M^* \times \mathbf{Syn}^+)^*$. 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 $E := I \times N$.

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 $\alpha, \beta \in N$.[38]

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

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

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

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 $\alpha, \beta \in N$.

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

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

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

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

The functions below are associated with thir namesakes from appendix B–1.2, 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 $\alpha, \beta \in N$.

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

[38] 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.

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

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

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 $\alpha, \beta \in N$.

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

B–2 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.[39] 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 objects

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

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 [36] indices were assumed to be freely generated, with filters (or constraints) ruling out all but a few be-indexed structures. Rogers [147] 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.[40]

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

[40]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.

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–2.1 Model-Theoretic Glory (cont'd)

The signatures of our meaning algebras contain the following six function symbols: nullary $\mathtt{x}$, unary $\textsc{e}$ and $\textsc{o}$, and binary $\mathtt{F}$, $\mathtt{R}$ and λ. In the intended 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[41]

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

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–2.2 A Semantics for MGs with Variable Management

On the semantic side of things, a minimalist expression is a pair $\langle i, N\rangle$, where $i \in I := T_M \times \{:, ::\} \times \mathbf{Syn}^*$, and $N \in N := ((T_M \times T_M) \times \mathbf{Syn}^+)^*$. 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 $E := I \times N$.

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 $\alpha, \beta \in N$.

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

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

[41] 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 [41], who assigns explicit denotations to all but λ). Once everything is made fully explicit, the type we in fact do assign to λ emerges.

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

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 $\alpha, \beta \in N$.[42]

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

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

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

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

The functions below are associated with their namesakes from appendix B–1.2, 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 $\alpha, \beta \in N$.

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

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

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

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

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

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 $\alpha, \beta \in N$.

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

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 2.17 (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) 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

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

Figure 2.17: The semantic type of expressions of natural language

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)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(ϵ, shave, an abbot) : prog, (every barber, -k -q)

$\mathbf{some}(\mathbf{abbot})(\lambda(\text{EOx})(\text{R}(\text{R}(\mathbf{shave})(\text{x}))(\text{x}))) : \text{prog}$,

$(\langle \text{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \text{-k -q})$

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

(ϵ, shave, an abbot) : perf, (every barber, -k -q)

$\mathbf{some}(\mathbf{abbot})(\lambda(\text{EOx})(\text{R}(\text{R}(\mathbf{shave})(\text{x}))(\text{x}))) : \text{perf}$,

$(\langle \text{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \text{-k -q})$

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

(ϵ, will, shave an abbot) : +k +q s, (every barber, -k -q)

$\mathbf{some}(\mathbf{abbot})(\lambda(\text{EOx})(\text{R}(\text{R}(\mathbf{shave})(\text{x}))(\text{x}))) : \text{+k +q s}$,

$(\langle \text{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \text{-k -q})$

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

(ϵ, will, shave an abbot) : +q s, (every barber, -q)

$\mathbf{some}(\mathbf{abbot})(\lambda(\text{EOx})(\text{R}(\text{R}(\mathbf{shave})(\text{x}))(\text{x}))) : \text{+q s}$,

$(\langle \text{Ox}, \mathbf{every}(\mathbf{barber})\rangle, \text{-q})$

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

(every barber, will, shave an abbot) : s

$\mathbf{every}(\mathbf{barber})(\lambda(\text{Ox})(\mathbf{some}(\mathbf{abbot})(\lambda(\text{EOx})(\text{R}(\text{R}(\mathbf{shave})(\text{x}))(\text{x}))))) : \text{s}$

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

$$(\lambda(\textsc{o}\mathtt{x})(\textbf{some}(\textbf{abbot})(\lambda(\textsc{eo}\mathtt{x})(\mathtt{R}(\mathtt{R}(\textbf{shave})(\mathtt{x}))(\mathtt{x})))))(f)$$

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

$$\textbf{some}(\textbf{abbot})(\lambda(\textsc{eo}\mathtt{x})(\mathtt{R}(\mathtt{R}(\textbf{shave})(\mathtt{x}))(\mathtt{x})))$$

Because $g_i = \textsc{o}\mathtt{x}(g) = \mathtt{x}(^{\wedge}g) = (^{\wedge}g)(0) = g(2 \cdot 0 + 1) = g_1$ and $g'_i = \textsc{o}\mathtt{x}(g') = g'_1$, 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 $e(g') \in \textbf{abbot}$, and g' belongs to the set denoted by

$$(\lambda(\textsc{eo}\mathtt{x})(\mathtt{R}(\mathtt{R}(\textbf{shave})(\mathtt{x}))(\mathtt{x})))(e)$$

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

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

Again, $g' \approx_2 h$, and therefore $f(g) = g'_1 = h_1$. Also, $e(g') = \textsc{eo}\mathtt{x}(h) = \mathtt{x}(^{\wedge\vee}h) =$

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 $\mathbf{shave}(e(g'))(f(g))$.

CHAPTER 3

Copying in Grammar

The goal of this chapter is to show how minimalist grammars can be extended with a structure copying mechanism. There are a number of logically possible ways of doing this, however our aim is to select one that changes minimally the structure of the formalism, in particular, its compatibility with phases, as discussed in chapter 2. Before we can do this, however, we need to get clear first on what the structure is that we are copying. I argue that the question of whether copying is internal or external merge is best understood as a debate about whether copies are derived, or not. I then show how each of these perspectives can be implemented in minimalist grammars, and (in appendix C–1.3) that these implementations are (strongly) equivalent. I then discuss the question of how copies are to be pronounced. I go through a variety of proposals, and show how they can be implemented in our system. I conclude with examining how extra-syntactic filters which crash otherwise syntacticly well-formed derivations can be incorporated into our system, thereby allowing for some of the functionality of Nunes's [131] Optimality Theoretic approach to pronunciation of chains. We begin by briefly rehashing the original motivation for the copy theory of movement.

1 The Copy Theory of Movement

Chomsky [39] argues that in order for the sentence in 3.1 to be interpretable,

(3.1) (guess) [[$_{wh}$ in which house] John lived t]

it needs to be converted into one of the following forms.

(3.2) 1. [which x, x a house] John lived [in x]

2. [which x] John lived [in [x house]]

There are two obvious ways of effecting this transformation. The first, reconstruction, posits movement of the relevant fragments of the PP back into its source position. The other option, the copy theory of movement, posits that what have been called traces are really copies of the moved material, and that a better approximation to the structure of a sentence like 3.1 is as in 3.3.

(3.3) (guess) [[$_{wh}$ in which house] John lived [$_{wh}$ in which house]]

Some operation particular to the LF component converts the phrases marked wh to either 3.4a or 3.4b.

(3.4) 1. [which house] [$_{wh}$ in t]

2. [which] [$_{wh}$ in [t house]]

A general process by means of which no 'redundant' information may remain in a representation usable by the CI system ensures that the duplicated material (among the higher and lower copies) is deleted. The only interpretable outcomes are assumed to be those which, in the higher copy, the phrase marked wh is deleted, and in the lower copy the other phrase is deleted, as sketched below.

[[which house] ~~[$_{wh}$ in t]~~ John lived ~~[which house]~~ [$_{wh}$ in t]]

The copying analysis of pied-piping is intended to extend naturally to other, more theoretically problematic phenomena, such as provided by the distribution

of anaphora, pronouns, and referring expressions. In previous incarnations of the principles and parameters theory of which minimalism is the most recent, the distribution of anaphora, pronouns, and referring expressions was governed by principles A, B, and C of the Binding Theory, respectively, which made reference to a level of representation (Surface Structure) now thought to be superfluous. Chomsky presents examples, and sketches proposals which suggest that not only is reference to this defunct level of representation eliminable, but that in so doing we are left with a more descriptively adequate theory. As an example, consider the sentence below.

(3.5) John wondered [which picture of himself] [Bill took t]

Chomsky observes that this sentence has two obvious loci of semantic indeterminacy; the reference of the anaphor may be identical with either the matrix or embedded subject, and the embedded predicate "take a picture" may be interpreted either literally (as in "abscond with a picture") or idiomatically (as in "photograph"). Instead of the four logically possible readings, only two are actually available—the idiomatic interpretation of the embedded predicate is available only if "Bill" is the antecedent of the anaphor. This fact is puzzling under the previous theory, according to which the antecedent of the anaphor is determined independently of and prior to the resolution of the interpretation of the embedded predicate. Under the copy theory outlined above, however, there is a natural way to link these together. The wh-phrase is, at LF, converted to one of the following representations.

(3.6) 1. [which] [t picture of himself]

2. [which picture of himself] t

The sentence 3.5 has then the following two possible structural descriptions at LF.

(3.7) 1. John wondered [which] ~~[*t* picture of himself]~~ [Bill took ~~[which]~~ [*t* picture of himself]]

2. John wondered [which picture of himself] ~~*t*~~ [Bill took ~~[which picture of himself]~~ *t*]

3.7a has the anaphor in the appropriate configuration for "Bill" to serve as its antecedent, and the one in 3.7b for "John." When coupled with the theory of idiom interpretation standard in the Principles and Parameters tradition, namely, that all pieces of the idiom must be structurally adjacent for the idiomatic interpretation to obtain, only 3.7a is predicted to allow for the idiomatic interpretation of "take a picture," thus deriving the correlation between the choice of antecedent for the reflexive and the availability of the idiomatic interpretation of the embedded predicate.

1.0.1 Visible Reflexes of Copying

The widespread acceptance of the copy-theory of movement has led to the discovery and popularization of cases where copying and deletion pave the way for an alternative analysis of the phonological form of sentences. Fanselow and Čavar [55] analyze discontinuous nominal phrases (as in 3.8a below) as stemming from deletion of parts of copies (as schematized in 3.8c). Compare 3.8a with the continuous (and synonymous) 3.8b.

(3.8) 1. Was hast du für Filme gesehen?
What have you for movies seen
"What kind of movies have you seen?"

2. Was für Filme hast du gesehen?

3. [Was ~~für Filme~~] hast du [~~was~~ für Filme] gesehen

Nunes [131] argues that the copy theory of movement provides us with a neat analysis of the Germanic wh-copy strategy for question formation, exemplified in 3.9 below, which exists alongside the more familiar wh-movement strategy (3.10).

(3.9) Mit wem glaubst du mit wem Hans spricht?
with whom believe you with whom Hans speaks
"Who do you believe Hans is talking to?"

(3.10) Mit wem glaubst du dass Hans spricht?

A standard assumption in the principles and parameters tradition is that long distance dependencies (such as obtain between a wh-word and its base position) are mediated via local movement steps. Representing source positions of movement as copies, the standardly hypothesized structure for 3.9 is as in 3.11.

(3.11) [$_{CP}$ [$_{PP}$ Mit wem] [$_{IP}$ glaubst du [$_{CP}$ [$_{PP}$ mit wem] [$_{IP}$ Hans [$_{PP}$ mit wem] spricht]]]]

Assuming traces are actually full copies of expressions allows for a simple account of these phenomena. In chapter 4 we delve more deeply into the question of whether a syntactic copying mechanism is warranted. For now, we simply assume that one such is. The rest of this chapter is devoted to carefully considering what exactly 'syntactic copying' is, and how it might be reasonably effected in a grammatical formalism.

1.1 Where Does Copying Take Place?

Although we have been speaking loosely of traces as being 'structurally identical' to their antecedents, a moment's reflection should reveal that it is not clear exactly what this should be taken to mean. There are two aspects of structure that might be relevant to us here—the first is the derived structure, what Chomsky [39] calls the structural description of an expression, and the second is what we might call the derivational structure, the derivational process itself. Clearly, 'copying' the derivational structure guarantees identity of derived structure.[1] However, the question of what it means to copy the derived structure without copying the derivational structure needs some additional thought. If, as seems reasonable, we view the derivation of a sentence as an abstract description of the process of parsing it (or better: as what the process of parsing it and the process of generating it have in common),[2] then copying just the derived structure would entail that, in recognizing a particular sentence, the processor would parse *just one* copy, and would re-recognize the other copy as a single Gestalt. Chomsky [32] discusses two interpretations of the copy theory of movement. According to the interpretation he prefers, which he refers to as internal merge, there is only one object, which appears in the derived tree at multiple positions. According to the interpreta-

[1] This holds only if the structure building operations are functions, as they are in our formalism. If to each derivation there corresponded possibly multiple derived structures, then copying derivational structure would allow for 'mismatches' between the derived structures of the two 'copies.' This isn't an empirically decidable matter, however, unless there are independent grounds for the determination of whether a particular structure building operation is one operation, or two, as any relation can be viewed as the union of functions.

[2] I take this to be the natural, 'levels', interpretation of the competence–performance distinction in generative linguistic theory. (For discussion of this perspective in cognitive science see e.g. [116, 141].) It is not obvious, however, that this is the interpretation that Chomsky [28, 31, 33, 34, 35] intends. According to Fodor [58], Chomsky conceives of linguistic competence as propositional in nature, and of performance as consisting of mechanisms that "put what you know into action." I do not understand this interpretation well enough to be able to say what the distinction between copying the derivation tree or the derived tree might amount to in it.

tion he refers to as external merge, copies are assembled as they are needed, and merged at various points of the derivation. These two interpretations seem best made sense of if we identify them with the two perspectives on what structure is copied as discussed above. Accordingly, the internal merge interpretation is copying at the level of the derived tree, and the external merge interpretation is copying at the level of the derivation tree. We discuss each of these two interpretations of the copy theory of movement further in the next sections. We first ask the question of how we can implement an operation of structure copying (§ 1.2 and § 1.3) before concerning ourselves with how the structures now generated are to be dealt with in terms of pronunciation and interpretation (§ 2).

Whichever approach we adopt, we need to make sure that the syntactic features that drive the derivation don't get duplicated—in our system, this would run afoul of the principle of immediacy immediately. Nunes [131] has suggested that allowing feature duplication might give a principled way to decide which copy to pronounce (see § 2.2 for a different account), but the changes to the grammar that must be instituted so as to allow us to distinguish between derivations that crash because formal features are unchecked, and those that do not crash despite having potentially the same unchecked formal features are left unstated.

As a final constraint on our developing theory, we want our theory of copying to be compatible with a theory of phases, which postulates the incremental destruction of the very structure that we want to be copied.

1.2 Internal Merge

Our reconstruction of the 'internal merge' approach to structure copying takes the structure copied to be the derived tree. Orthogonal to the afore mentioned distinction between copying as internal and external merge commented on by

Chomsky, there are different approaches to the implementation of the internal merge approach to copying, which center around whether copies should be explictly or implicitly (via structure sharing—see [61] for an excellent presentation of minimalism in these terms) represented in our derived structures (see figure 3.1).[3] The implicit representation camp uses multiple dominance structures to allow for multiple copies of the same expression to be represented by additional edges. In the limit, this allows for a substantial savings in ink. Note that there is

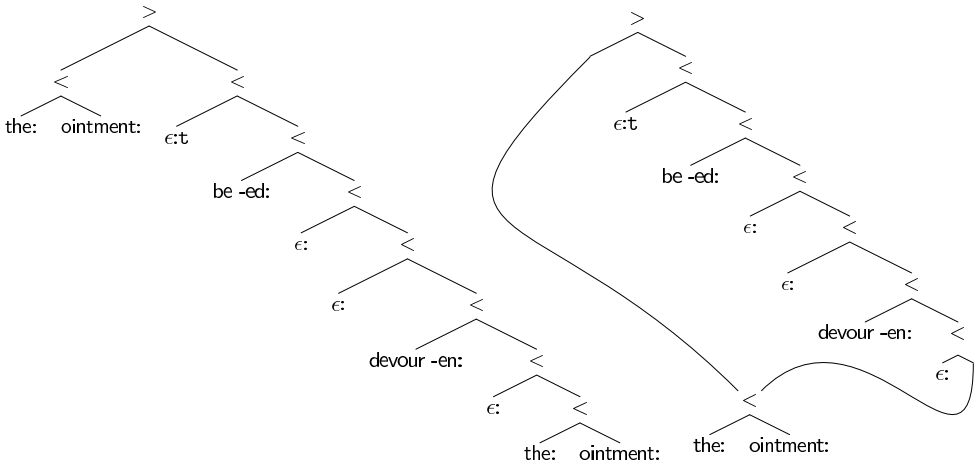


Figure 3.1: Explicit versus implicit representations of copies

a question as to what counts as 'the same expression.' Minimalist multiple dominance approaches take all and only common structure amongst members of the same chain to be so compressed. Thus multiple dominance approaches explicitly

[3]Unless we are realists about our notation, it serves merely as a vehicle within which to couch our thoughts, and equivalent notations allow for expression of the same ideas. This is not to deny that some notations may be much more concise, or better suited for the task at hand, than others (just think of multiplication by two in binary, versus decimal, notation). But this is another matter entirely.

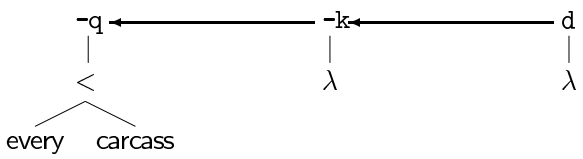
As for the realists among us, I suspect that the distinction between levels of description is being blurred. It is extremely useful to be able to specify which function is being computed independently of the many possible algorithms which effect its computation. This is the level at which the program of generative linguistics has been conducted since at least [33]. At lower levels, when we worry about how the relation we have formally specified at the higher levels could feasably be computed, space optimization strategies such as multiple domination may be relevant. (For a proposal about the relationship between recursive definitions of functions, algorithms, and their implementations see [126, 127].)

represent which subtrees are related via movement (i.e. which belong to the same chain). It is this fact, not the 'structure sharing', that has proven so useful.

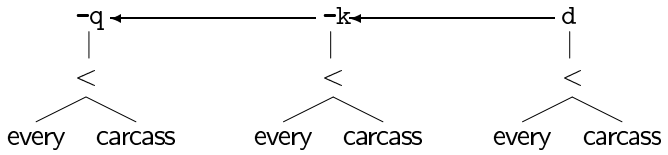
Any theory of copying structure is going to need to deal with the potential problem of unwanted featural duplication. Note that the shared structure representation of copies neutralizes the distinction between a number of different (but equivalent) explicit representations. One natural way to circumvent the problem of featural duplication with an explicit representation of copies is to simply not copy features. With this approach, conditions on movement (like our principle of immediacy) may be carried over from the non-copy theory unchanged. Another option, if we take the copied structure to include the features, is to reformulate our feature checking operations so as to apply to all chain members whenever any one of them enters into the appropriate configuration. There is a host of tedious bookkeeping, such as deciding which chain link will be copied in the move operation. We might wish to revise our principle of immediacy as well, so as to require only that no two expressions *from different chains* may have the same first licensee feature.

Given the obvious simplicity of the structure sharing representation over other, equivalent, representations, it seems almost perverse to use any of these others. We will show how this view of copying can be reconciled with our phase-based elimination of syntactic levels of representation shortly. First, let's get a feel for how it should work.

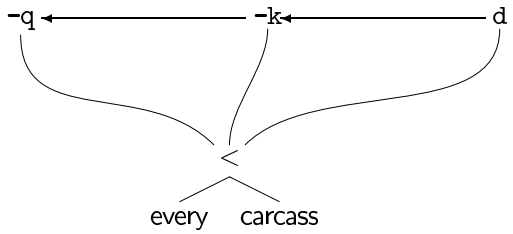
Recall from chapter 2 the 'chain'-like representation of phrases, which made explicit the interface effects of each chain position.



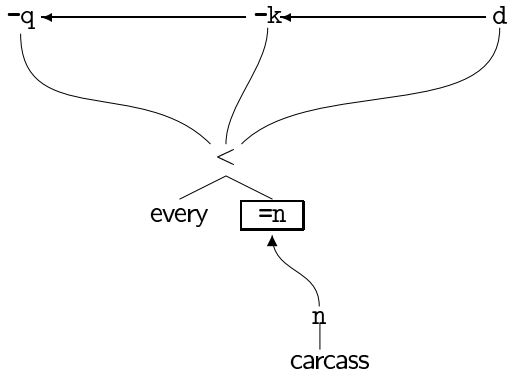
As we are now assuming that a copy of an expression appears in each of its chain positions, the above chain should be rewritten so as to look something like the following.



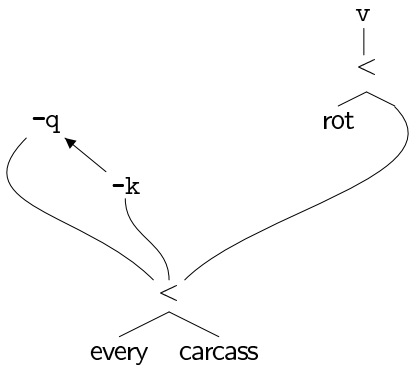
Representing the copies implicitly, by sharing identical structure, we obtain the below.



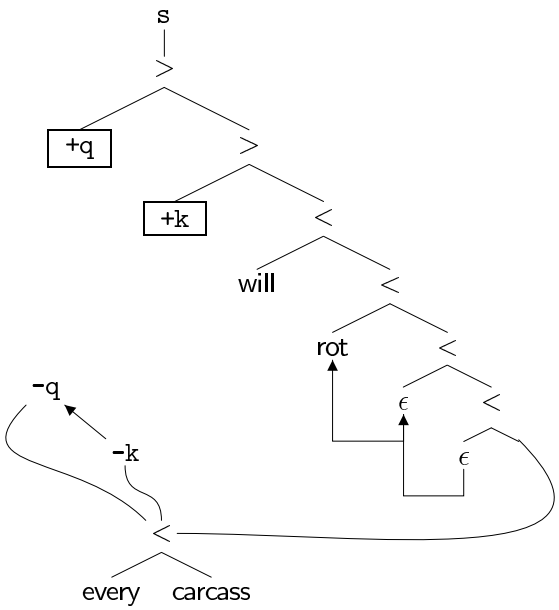
Here's how a derivation of *every carcass will rot* looks with this representation scheme. We begin by merging *every* and *carcass*.



We next merge *every carcass* and *rot* together. Note the unattached rest of the chain of *every carcass* in the below.

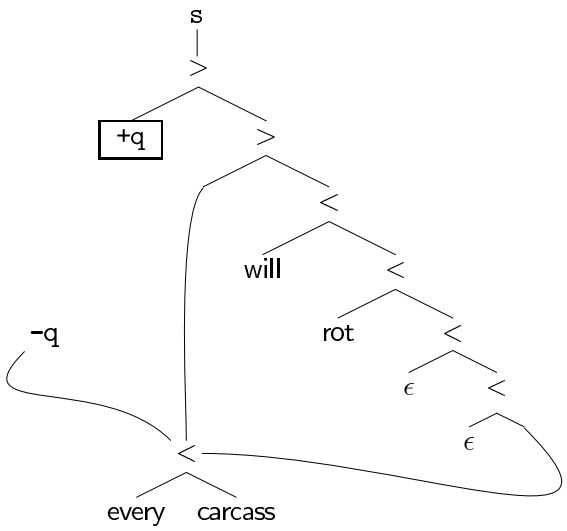


The derivation continues with the successive merger of ϵ::=>v prog, ϵ::=>prog perf, and then will::=perf +k +q s. The licensor features of *will* can be thought of as licensing 'internal merger' of a dangling chain position of an expression into a specifier of the phrase headed by *will*.[4] As *will* has two licensor features, +k and +q, it licenses two specifier positions, which are shown in the below with the licensing features 'boxed in'.

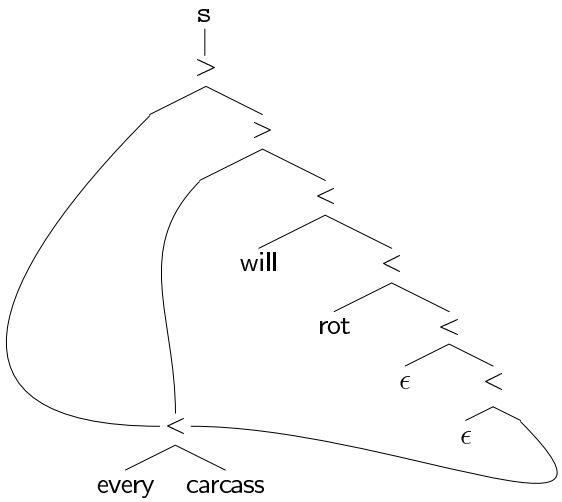


[4]Movement is sometimes referred to as internal merger (at which point merger is called external merger) in order to emphasize the similarity between the operations move and merge. The hope is clearly that these two operations will be reducable to a single operation. The intent is obvious, but is difficult to state precisely, as, trivially, the union of merge and move is itself a function, but we would not want to count this as a successful reduction.

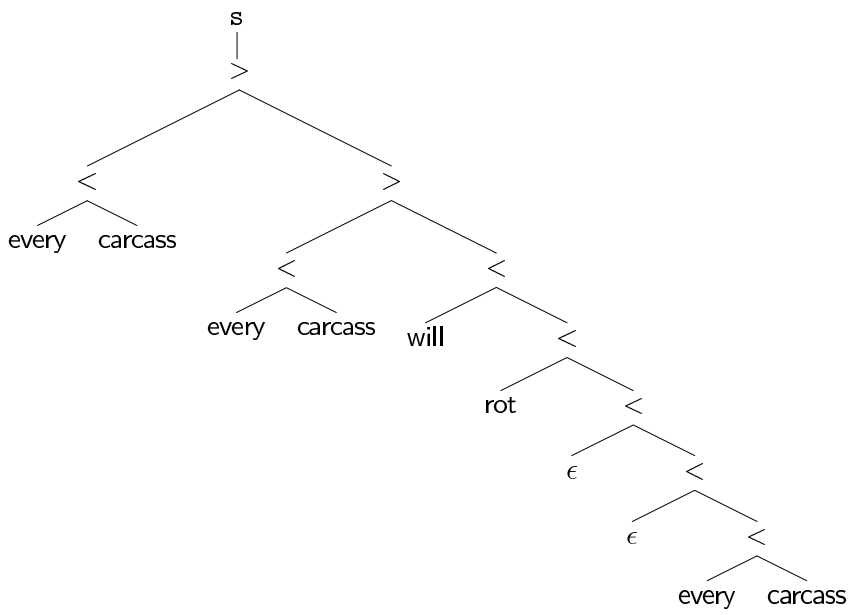
In the next step the dangling `-k` chain position of *every carcass* is internally merged into the appropriate specifier of the phrase headed by *will*.



The final step in the derivation of this expression is to internally merge the last dangling chain position (`-q`) of *every carcass* into the `+q` specifier of *will*'s phrase.



Note again that this is the *very same expression* as the one below. The difference between the representations is that the three copies of *every carcass* are explicitly represented in the below, while implicitly represented in the above.



1.2.1 Phases

We have seen already that, in the context of our present system, neither movement nor reconstruction requires us to work with expressions any more structured than sequences of syntactically categorized interface objects. In other words, while trees may be quite useful to the linguist, this is a fact about linguists, not about language. At least, this is so *provided that none of the operations we need to describe language require a richer representation.* At first blush, an operation of structure copying would seem to require us to provide for the existence of the structure copied. However, consider what we want the structure in the copies to *do*

> We can think of the [derived tree] as a complex of instructions for these performance systems, providing information relevant to their functions. ([39], pg. 168)

In fact, we might ask, since the shape of the derived tree of the moving (i.e. copied) expression is unchanging, why not simply send it to be interpreted at the various interfaces once, and *copy that interpreted material*? It turns out that a minimal adjustment to our tree-less notation from chapter 2 suffices to implement this idea. We go through a derivation of *every carcass will rot* using our tree-less notation.

In the first step, we merge *every* and *carcass* together, interpreting the result immediately at both interfaces (only the phonological interpretation is shown below—our semantics as developed in chapter 2 carries over unchanged to this present system).

1. merge(every::=n d -k -q, carcass::n)

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

Next we merge *rot* with *every carcass*. A copy of the phonological interpretation of *every carcass* is combined with the phonological interpretation of *rot*. Another copy of the phonological interpretation of *every carcass* is put into storage, along with its syntactic features (of which there is but one copy).

2. merge(rot::=d v, 1)

$$(\epsilon, \text{rot}, \text{every carcass}) : \texttt{v}, (\text{every carcass}, \texttt{-k -q})$$

Next we merge ϵ::=>v prog, ϵ::=>prog perf, and then will::=perf +k +q s with the expression *rot every carcass*.

3. merge(ϵ::=>v prog, 2)

$$(\epsilon, \text{rot}, \text{every carcass}) : \texttt{prog}, (\text{every carcass}, \texttt{-k -q})$$

4. merge(ϵ::=>prog perf, 3)

(ϵ, rot, every carcass) : perf, (every carcass, -k -q)

5. merge(will::=perf +k +q s, 4)

(ϵ, will, rot every carcass) : +k +q s, (every carcass, -k -q)

We next internally merge *every carcass* into a +k-licensed specifier of *will*—this amounts to combining a copy of the phonological interpretation of *every carcass* with the rest of the phrase, and checking the feature pair (+k, -k).

6. move(5)

(every carcass, will, rot every carcass) : +q s, (every carcass, -q)

Finally we internally merge *every carcass* a last time, satisfying all of its syntactic requirements. There is no need to make another copy of the phonological interpretation of *every carcass*—as one sits already in storage, we use that.

7. move(6)

(every carcass every carcass, will, rot every carcass) : s

A bevy of particulars bear pointing out. First, to allay any unease, the sentence we have just derived is not grammatical in any variety of English of which I have ever heard tell. Matters will not be left at this—the subject of pronunciation of copies will be taken up in § 2, after we deal with the second possible approach to syntactic copying in § 1.3. Second, what of the claim we made in chapter 2 to the effect that movement was strongly successive cyclic—clearly, here we have only three copies of *every carcass*, not the six we would expect if *every carcass*

were indeed moving to each intermediate specifier position. Note that we have relegated copying to the phonological component of the grammar—we are thus seeing that our phonological interface is not currently taking advantage of the successive cyclicity of our system, just as our semantics was not, earlier. By virtue of keeping track of which expressions have unchecked features, our syntax makes them available at any point in the derivation (until their features have been all checked, at which point they are no longer treated seperately syntactically). It would be a fact about our interface maps (if true) that they only deal with feature checking steps (and thus not with non-feature driven successive cyclic movements), not about our syntax. Finally, note that we did not need to revise our semantic interface to deal with the addition of derived tree copying to our system; currently only our phonological interface is taking advantage of the copying inherent to our syntax.[5]

1.3 External Merge

The derivational process imposes a particular structure upon an object independently of any derived structure we may or may not choose to assign to that object in the course of its derivation. Although we may try to eliminate derived structure, its derivation is an intrisic property of an expression, and for this reason is a natural (arguably, the most natural) referent of the term 'syntactic structure'.

But how are we to determine whether two processes culminating in the same thing are themselves identical? One option is to encode a description of (the relevant aspects of) the process itself in the thing derived. Pursuing this option lands us back in the 'identity of derived structure' camp we discussed a section ago. The other option, which we shall explore here, is not to wait until the

[5]See footnote 30 in chapter 2 for discussion.

processes are complete to compare them, but instead to run them in parallel, and to check at each step whether they are performing the same action. We return to this intuition in a moment.

Chomsky [32] correctly notes that

> [External merge] requires some [...] device to distinguish copies from unrelated occurrences.

I shall call this 'Chomsky's problem'. Certainly, we can have identically derived subexpressions that we do not wish to treat as 'copies.' Consider the following sentence, in which the fact that the subject and object are derivationally identical is not a grammatically interesting one.

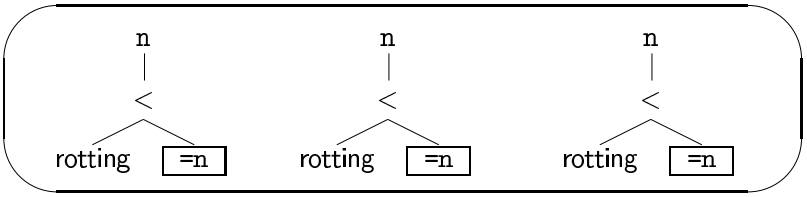
(3.12) An ex-girlfriend of mine bumped into an ex-girlfriend of mine.

We make the pre-theoretical judgement that, in a particular sentence, derivational identity is 'accidental', and thus not grammatically relevant based on the same considerations that lead us to identify sentences as belonging to a 'NP V NP' construction, but not to a 'NP V Mary' construction: there are no significant generalizations to be made about the 'NP V Mary' construction that are not consequences of the 'NP V NP' construction. Similarly, the 'NP V CopyNP' construction is not usefully isolable by the analyst (at least not in English—Lee [108] argues that San Lucas Quiaviní Zapotec uses a construction like 'NP V CopyNP' to express what English speakers would by means of a reflexive pronoun). As such, this embodies an empirical hypothesis about the proper regimentation of natural languages into construction types.

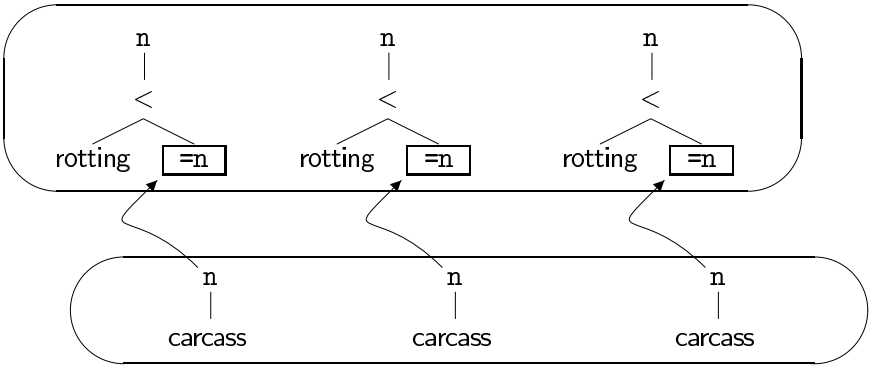
Returning now to our intuition above, of checking at each step whether two processes were doing the same thing, this provides us with a natural solution

to Chomsky's problem; identically derived objects are non-accidentally so just in case their derivational processes proceeded in tandem. We implement this by adapting an idea from the Tree-Adjoining Grammar (TAG) tradition [152–155] into our system. We allow ourselves to look at multiple expressions at the same time, and to perform the same grammatical operation on each of them. This ensures that, if we start out with identical expressions, we end up with (larger) identical expressions. Thus we are able to reduce the operation of copying arbitrarily large derivations to an operation copying lexical items, and operations which perform the same action on multiple expressions at once.

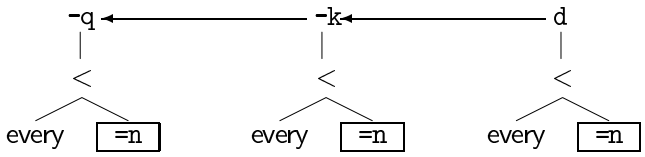
Let's see how this can be made to work. Imagine that we wanted three copies of the expression *rotting carcass* (for what deviant purpose we will discuss in a bit). Since rotting::=n n is a trivial chain, we must select *rotting* from the lexicon three times at once, forming the single, composite, object below. The composite object itself can be thought of as a kind of workspace, tying together (or *synchronizing*) multiple (in this case, three) independent processes.



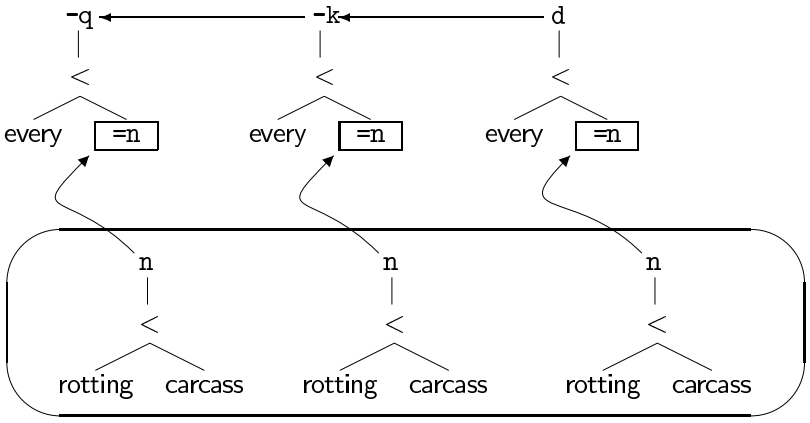
We perform the same thrice-selection of *carcass*, and then merge these two composite objects together. Since we are operating with the synchronization of multiple processes, the natural choice for how to interpret the merger of two composite objects is as merging each expression in the first with its counterpart in the second.



Now, returning to the question of why anyone would ever want three copies of the expression *rotting carcass*, consider the fact that the lexical item *every* has three chain positions, each of which selects for a noun phrase.



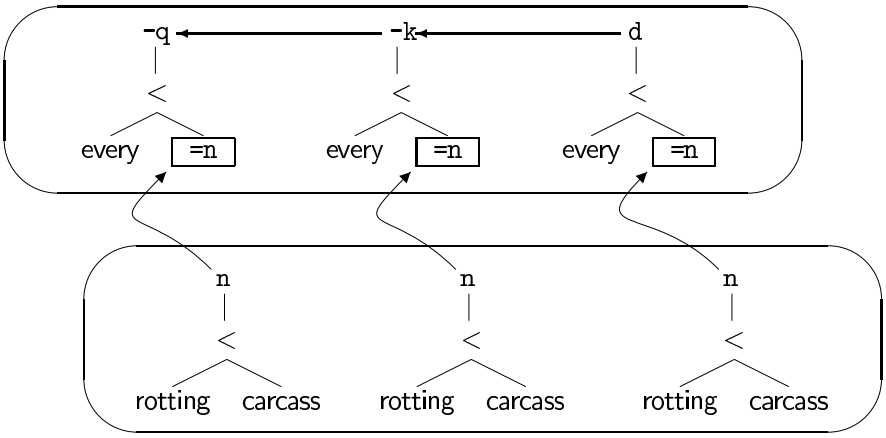
To derive the phrase *every rotting carcass*, we need to combine the expression *every* with three copies of *rotting carcass*—one copy for each chain position of *every*. That is, non-trivial chains combine with composite expressions, which represent the synchronization of the derivations of multiple independent expressions.



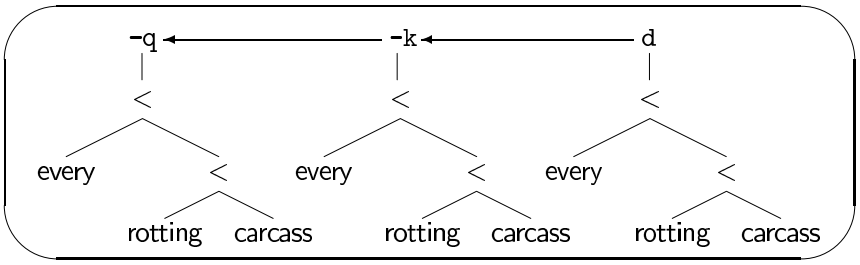
The major difference between our system and the synchronous TAG formalism

developed by Shieber and his colleagues, which derives sets of composite expressions, is that we allow composite expressions to be reintegrated into a single expression, which then has multiple 'copies' of the expressions whose derivations were synchronized.

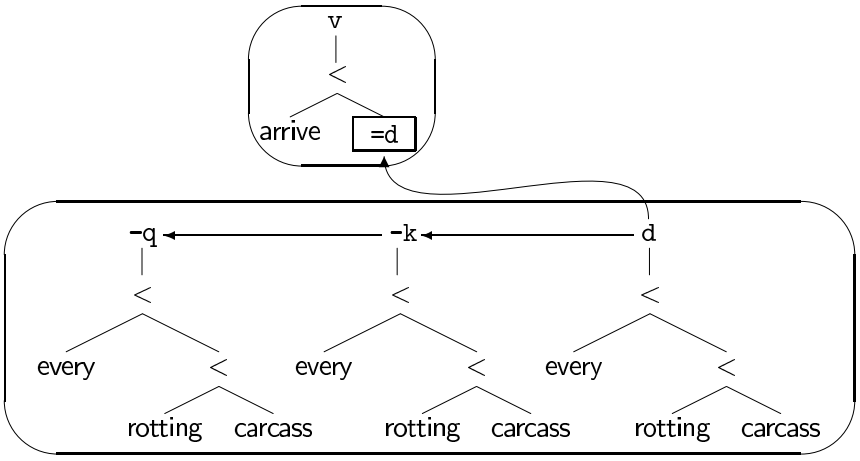
We show an example derivation of the expression *every rotting carcass arrived*. First, we merge together three synchronized derivations of rotting::=n n with three synchronized derivations of carcass::n, as shown above. Next, we merge together a single derivation of every::=n d -k -q with the three synchronized derivations of *rotting carcass*,



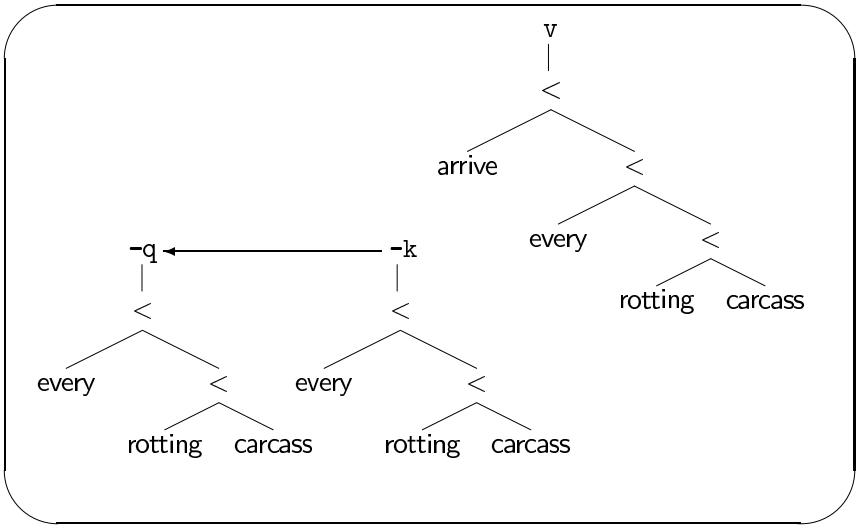
which results in the single derivation of *every rotting carcass* shown below.



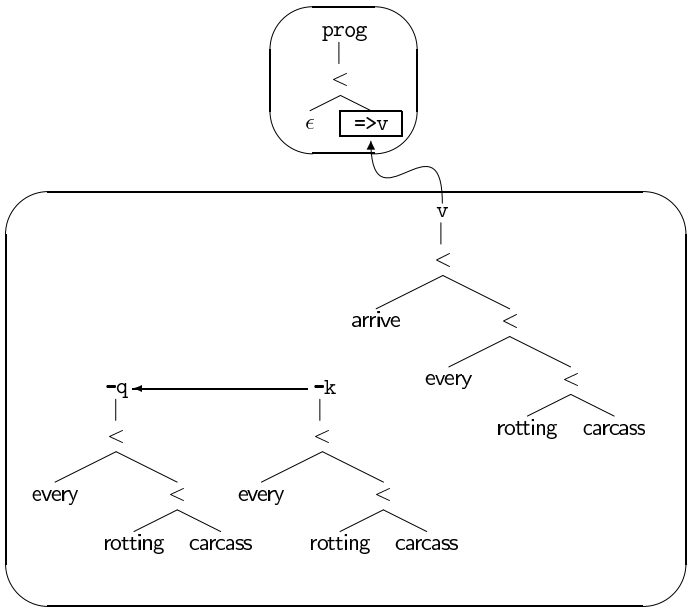
Next we merge the above expression and a single derivation of arrive::=d v,



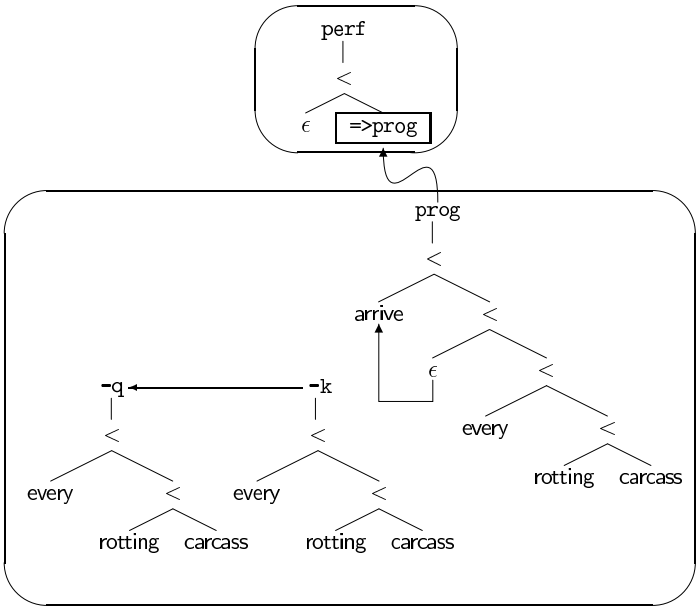
which results in the single derivation with the `-k` and `-q` chain links of *every rotting carcass* in storage.



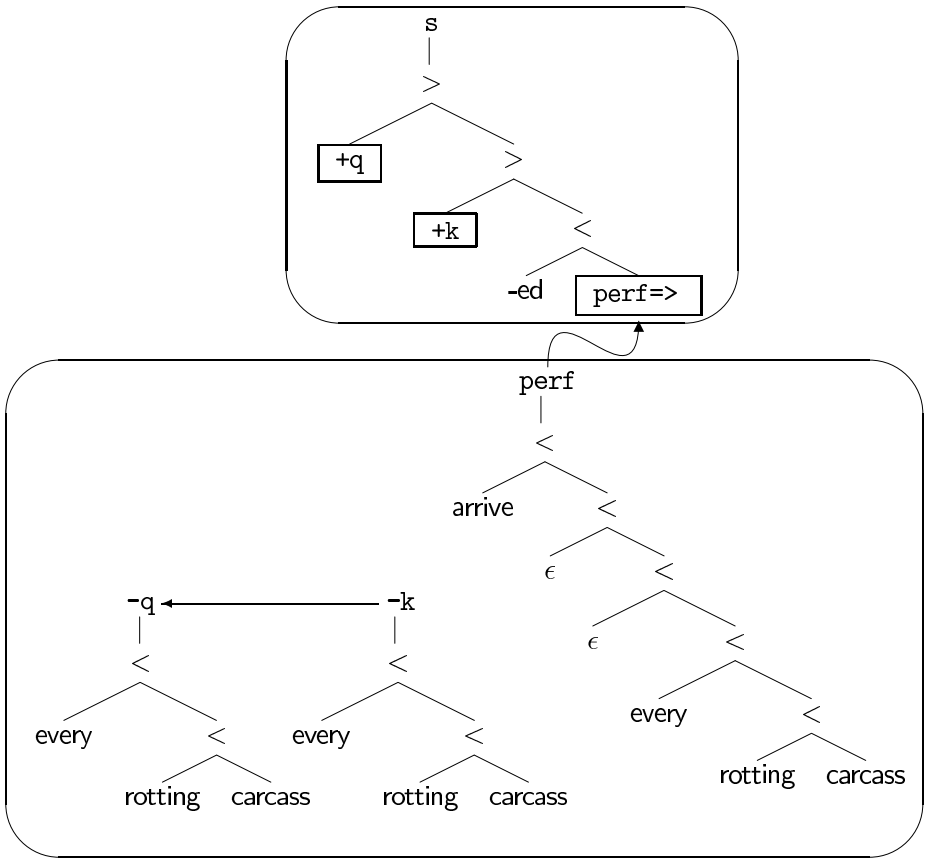
We next merge together the above single derivation with a single derivation of ϵ::`=>v prog`,



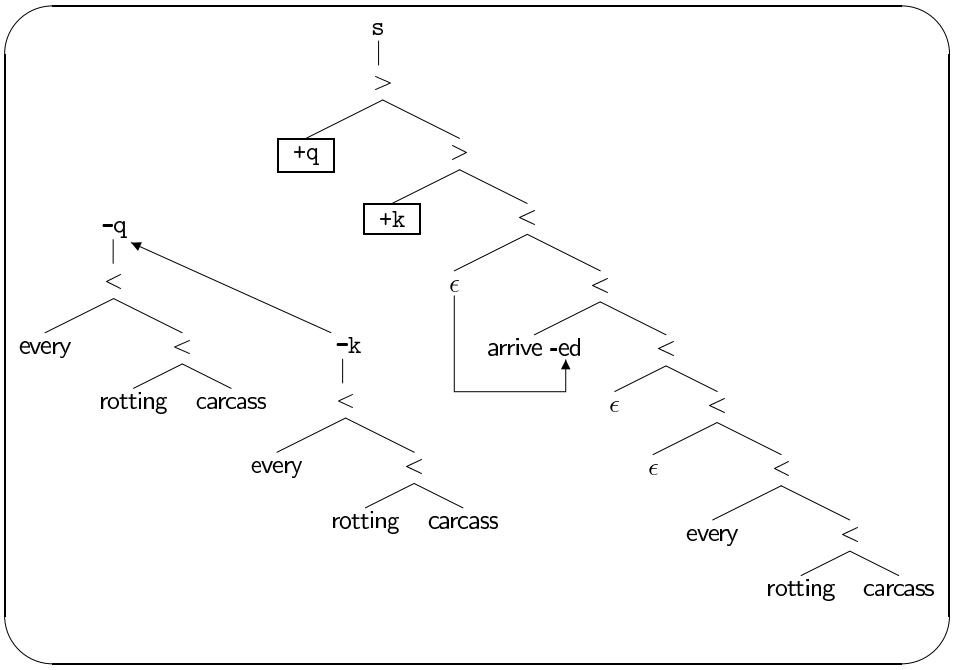
and then with a single derivation of ϵ::`=>prog perf`.



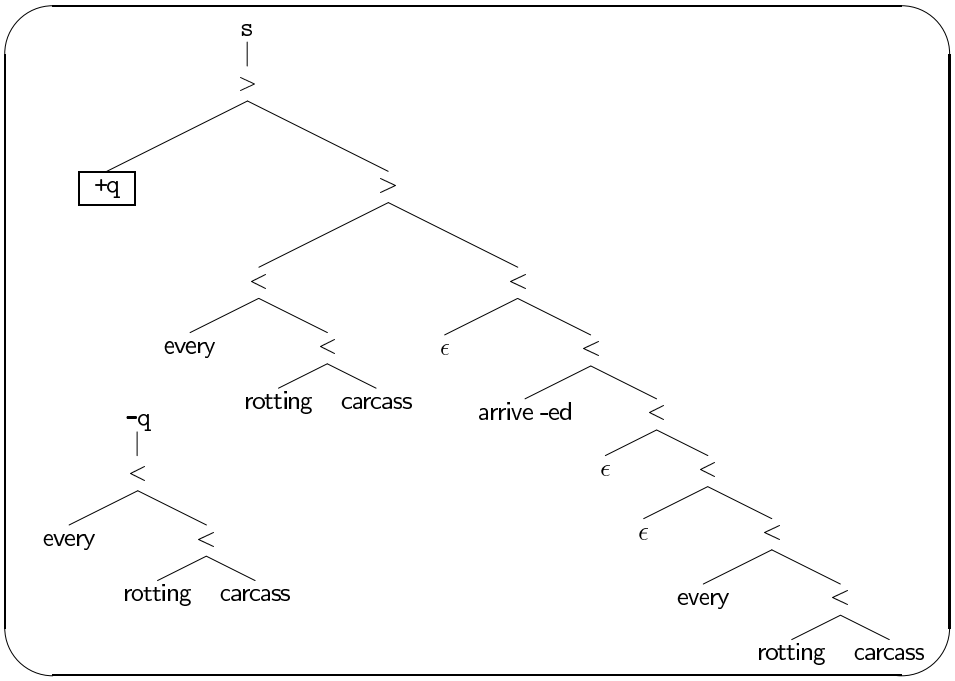
Next we merge a single derivation of `-ed`::`perf=> +k +q s` with the single derivation above,



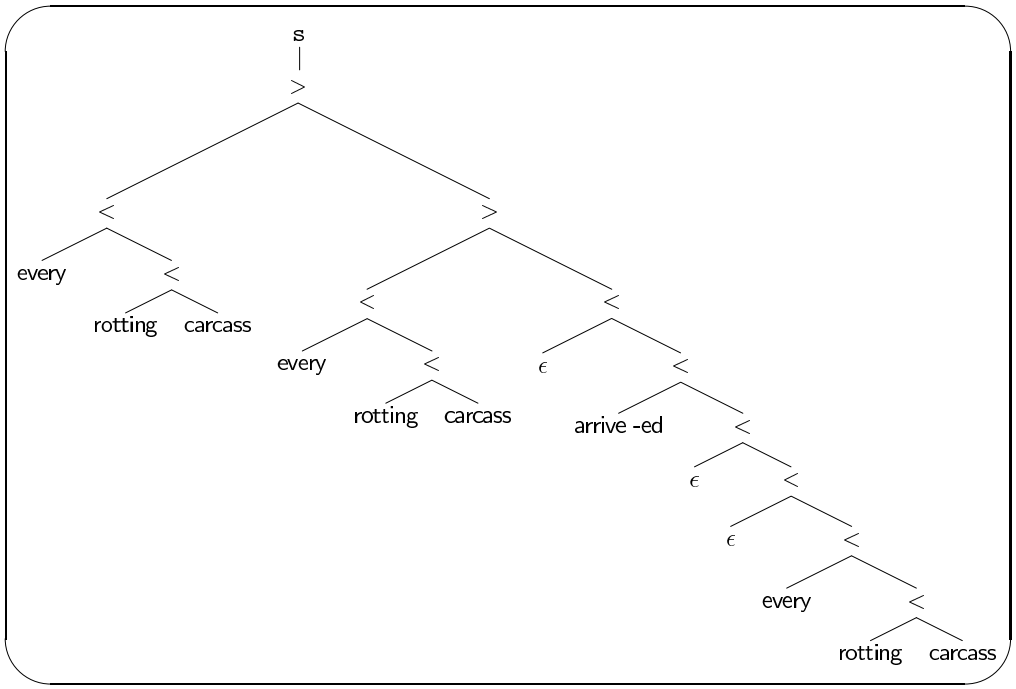
which, after head movement, yields the below expression.



We next move the next link in the chain of *every rotting carcass* into the available `+k` specifier position. If there were multiple synchronized processes, move would apply to each of them.



Finally, we fill the `+q` specifier with the last link in the chain of *every rotting carcass*. Thus, we end with a workspace consisting of just one expression, with no dangling chains.



1.3.1 Phases

We can employ a similar abbreviatory convention as used previously to eliminate the tree structure of our expressions. We still need, however, to represent synchronization of derivations, as we have no access to copying operations. We write each component of a composite expression within curly brackets, and indicate that they comprise the composite expression by enclosing them in round brackets. The intended meaning of the notation is best understood by comparing each step in the derivation below with its counterpart above.

First we select three copies of *rotting* and *carcass* from the lexicon.

1. select(select(select(rotting::=n n)))

$$\left(\begin{array}{l} \{\text{rotting} :: \texttt{=n n}\} \\ \{\text{rotting} :: \texttt{=n n}\} \\ \{\text{rotting} :: \texttt{=n n}\} \end{array} \right)$$

2. select(select(select(carcass::n)))

$$\left(\begin{array}{l} \{\text{carcass} :: \texttt{n}\} \\ \{\text{carcass} :: \texttt{n}\} \\ \{\text{carcass} :: \texttt{n}\} \end{array} \right)$$

Next we merge these two composite expressions together, yielding the more complex composite expression with three component parts shown below.

3. merge(1, 2)

$$\left(\begin{array}{l} \{(\epsilon, \text{rotting}, \text{carcass}) : \texttt{n}\} \\ \{(\epsilon, \text{rotting}, \text{carcass}) : \texttt{n}\} \\ \{(\epsilon, \text{rotting}, \text{carcass}) : \texttt{n}\} \end{array} \right)$$

We next select a single copy of *every*. Note that we represent each chain link of *every*, and that these three links together form a single, simple, expression (i.e. are enclosed within a single pair of curly brackets).

4. select(every::=n d -k -q)

$$\left(\left\{ \begin{array}{l} \text{every} :: \texttt{=n -q} \\ \text{every} :: \texttt{=n -k} \\ \text{every} :: \texttt{=n d} \end{array} \right\} \right)$$

We then merge *every* with the three copies of *rotting carcass* derived earlier. All three copies of *rotting carcass* are combined into the single copy of the three-link chain *every*.

5. merge(4, 3)

$$\left(\left\{\begin{array}{l}(\epsilon,\ \text{every, rotting carcass}) : \texttt{-q}\\(\epsilon,\ \text{every, rotting carcass}) : \texttt{-k}\\(\epsilon,\ \text{every, rotting carcass}) : \texttt{d}\end{array}\right\}\right)$$

We select one copy of *arrive*, which is a trivial chain.

6. select(arrive::=d v)

$$\left(\left\{\ \text{arrive} :: \texttt{=d v}\ \right\}\right)$$

When we merge *arrive* with *every rotting carcass*, we need to keep track of the fact that there are two copies of this latter chain left. We enclose these two copies within square brackets.

7. merge(6, 5)

$$\left(\left\{\begin{array}{l}(\epsilon,\ \text{arrive, every rotting carcass}) : \texttt{v},\ \left[\begin{array}{l}(\text{every rotting carcass, } \texttt{-q})\\(\text{every rotting carcass, } \texttt{-k})\end{array}\right]\end{array}\right\}\right)$$

We successively select and merge one copy each of our aspectual heads ϵ::=>v prog and ϵ::=>prog perf.

8. merge(select(ϵ::=>prog perf), merge(select(ϵ::=>v prog), 7))

$$\left(\left\{\begin{array}{l}(\epsilon,\ \text{arrive, every rotting carcass}) : \texttt{perf},\ \left[\begin{array}{l}(\text{every rotting carcass, } \texttt{-q})\\(\text{every rotting carcass, } \texttt{-k})\end{array}\right]\end{array}\right\}\right)$$

The derivation enters into the last stretch with the selection and merger of -ed::perf=> +k +q s.

9. merge(select(-ed::perf=> +k +q s), 8)

$$\left\langle \left\{ \begin{array}{l} (\epsilon,\ \epsilon,\ \text{arrive -ed every rotting carcass}) : \texttt{+k +q s}, \\ \qquad\qquad \left[\begin{array}{l} (\text{every rotting carcass},\ \texttt{-q}) \\ (\text{every rotting carcass},\ \texttt{-k}) \end{array} \right] \end{array} \right\} \right\rangle$$

We now move the two copies of *every rotting carcass*.

10. move(9)

$$\left\langle \left\{ \begin{array}{l} (\text{every rotting carcass},\ \epsilon,\ \text{arrive -ed every rotting carcass}) : \texttt{+q s}, \\ \qquad\qquad \left[\begin{array}{l} (\text{every rotting carcass},\ \texttt{-q}) \end{array} \right] \end{array} \right\} \right\rangle$$

11. move(10)

$$(\{(\text{every rotting carcass every rotting carcass},\ \epsilon,\ \text{arrive -ed every rotting carcass}) : \texttt{s}\})$$

1.3.2 Remarks on Derivational Copying

This approach to copying treats copying as fundamentally syntactic in nature, as opposed to the internal-merge approach which pushes the copying operation from the syntax into the various domains with which it interfaces. That is, this approach to copying treats the copies as syntactic resources—if a copy is needed, it must be constructed. This differs fundamentally from the 'internal merge' perspective on copying, according to which the operation of copying is not a resource sensitive operation, and has for this reason a *completely different character* from the rest of our syntactic operations.

We have talked a great deal about successive cyclic movement, claiming that even though such movement is not 'overtly' represented as occuring in minimalist grammars, this is more a fact about our interface operations than our syntax. I have tied this claim to a general notion of successive cyclicity, which was defined

to be whether, in principle, information about an expression is available at various points in the derivation. In our copying-as-synchronization paradigm we see that feature-driven movements are treated quite differently from non-feature driven (i.e. successive cyclic) movements. Here we treat feature-driven movements as the 'external merger' of identically derived material. Certainly, there is no successive cyclic external merger of identically derived material. However, complete information about 'moving' expressions is still potentially available at every step of the derivation from those expressions' first merge positions to their last-move positions. Therefore in this derivational copying system, too, there exists strict successive cyclicity.

2 The Pronunciation of Copy-Chains

Now that we have seen two ways of conceiving of structure copying, and an implementation of each, we turn next to patch a glaring hole in our idealization—we don't generally pronounce all the copies that we are led to postulate given our movement-filled analyses of language. In this section we explore various increasingly refined strategies of pronouncing structures with copies.

What we have done in the previous section has been to increase the power of our syntactic formalism, so as to allow it to represent copying. What our goal is here is to show how we can harness this otherwise indiscriminate increase in power, and employ it in a controlled fashion.

Although unable to allow for constructions involving copying, our previous, pre-copy approach to grammar had quite a bit going for it. In particular, it accounted for the apparent facts that generally only one copy of an expression is ever pronounced, and that generally the higher the chain link, the more likely it

will be pronounced. Insofar as we want to be able to describe copying, or spell-out of non-highest chain links, our previous approach cannot be right. However, it performs remarkably well on a wide array of cases, and so, ideally, we would rather augment it than replace it. Here we will see how to do just that. We begin by discussing how to recover our pre-copying pronunciation regimen, according to which only the 'top-most' chain link is pronounced, and show that our system generates the 'right' results even in tricky cases of remnant movement (i.e. of copies within copies). We then introduce local non-determinism into the determination of which copies are pronounced, which allows us to state disjunctive conditions on spellout (e.g. if there are strong heads, pronounce the copy adjacent to the highest strong head, otherwise pronounce the lowest copy). With this perspective on the relation between pronunciation and movement, we see that 'agree' as a grammatical operation is simply movement with particular interface effects. We show how adding local indeterminacy allows us to interface our syntax with morphological filters that rule out certain combinations of morphemes, resulting in different copies being pronounced. Finally we show how to require that multiple copies be pronounced.

2.1 Pronounce Highest

The notion of the 'highest' chain link, although simple to define over derived trees in normal cases (the highest c-commanding link), becomes quite complex once we allow copies to occur within copies (e.g. in the case of remnant movement). We redefine the notion of height as 'derivationally last', and show that this simple definition coincides not only with our c-command based definition in normal cases, but also with our intuition in cases of remnant movement. Moreover, we can 'inline' this derivational pronunciation regimen into the derivation of an

expression, rendering this pronounce-highest scheme compatible with our strong theory of phases.

In our purely derivational system, it is easy to determine whether a given link is 'derivationally last'—if a merged or moved expression has no more features, it is the derivationally last link in its chain, and must therefore be pronounced. Other merge or move steps are followed by later, derivationally higher, ones, and thus cannot host pronounced material. This is effected slightly differently according to how we understand copying.

If we adopt an internal merge perspective on copying, then since what is copied is the interpreted phonological content of the moving expression, 'not pronouncing' a copy simply means not copying. Consider that copying the derived tree just means copying non-syntactic representations (adopting a strong theory of phases)—syntax has but one chance to affect the shape of these non-syntactic representations; it cannot choose to copy them, and then later delete them. As such, we see that the internal merge approach to copying with the pronounce-highest scheme just *is* our previous formalism.

Adopting an external merge perspective on copying on the other hand, we must still derive all copies, but we may choose whether or not to incorporate the phonological interpretation of each at the point it is inserted into its chain position. Pronounce highest is then just the special case where a link's phonological interpretation is incorporated into a containing expression just in case that link is the highest in its chain.

Regardless of the perspective on copying, we distinguish between pronounced and ~~unpronounced~~ material as just shown. We show the derivation of the sentence *John will rot* below with a pronounce-highest regimen both with an internal merge, and an external merge perspective on copying.

(3.13) John will rot.

We begin by merging together (a single copy each of) *rot* and *John*. As *John* is not a trivial chain (i.e. it has features left to be satisfied), the newly merged chain position is not (or rather, will not be) highest in its chain. Therefore, according to our pronounce highest regimen, we do not pronounce it (which is marked by being struck through).

1. merge(rot::=d v, John::d -k -q)

$$(\epsilon,\ \text{rot},\ \sout{\text{John}}) : \texttt{v}, (\text{John},\ \texttt{-k}\ \texttt{-q})$$

2. merge(select(rot::=d v), select(John::d -k -q))

$$\left(\left\{ (\epsilon,\ \text{rot},\ \sout{\text{John}}) : \texttt{v}, \begin{bmatrix} (\text{John},\ \texttt{-q}) \\ (\text{John},\ \texttt{-k}) \end{bmatrix} \right\} \right)$$

At the next step, we merge (one copy of) ϵ::=>v prog with the expression above. Since the expression above *is* the highest link in its (admittedly trivial) chain, it is pronounced.

1. merge(ϵ::=>v prog, 1a)

$$(\epsilon,\ \text{rot},\ \sout{\text{John}}) : \texttt{prog}, (\text{John},\ \texttt{-k}\ \texttt{-q})$$

2. merge(select(ϵ::=>v prog), 1b)

$$\left(\left\{ (\epsilon,\ \text{rot},\ \sout{\text{John}}) : \texttt{prog}, \begin{bmatrix} (\text{John},\ \texttt{-q}) \\ (\text{John},\ \texttt{-k}) \end{bmatrix} \right\} \right)$$

The expression above is again highest in its chain, and is pronounced upon merger with (a single copy of) ϵ::=>prog perf.

1. merge(ϵ::=>prog perf, 2a)

$$(\epsilon,\ \text{rot},\ \sout{\text{John}}) : \texttt{perf}, (\text{John},\ \texttt{-k}\ \texttt{-q})$$

2. merge(select(ϵ::=>prog perf), 2b)

$$\left(\left\{(\epsilon,\ \text{rot},\ \sout{\text{John}}) : \texttt{perf}, \begin{bmatrix} (\text{John},\ \texttt{-q}) \\ (\text{John},\ \texttt{-k}) \end{bmatrix}\right\}\right)$$

Again, because the expression above is a trivial chain, it is highest in its chain, and is thus pronounced in its merge position with *will* in the below.

1. merge(will::=perf +k +q s, 3a)

$$(\epsilon,\ \text{will},\ \text{rot}\ \sout{\text{John}}) : \texttt{+k}\ \texttt{+q}\ \texttt{s}, (\text{John},\ \texttt{-k}\ \texttt{-q})$$

2. merge(select(will::=perf +k +q s), 3b)

$$\left(\left\{(\epsilon,\ \text{will},\ \text{rot}\ \sout{\text{John}}) : \texttt{+k}\ \texttt{+q}\ \texttt{s}, \begin{bmatrix} (\text{John},\ \texttt{-q}) \\ (\text{John},\ \texttt{-k}) \end{bmatrix}\right\}\right)$$

The +k feature on the head of the above expression triggers the move operation, determining the position of the next link of the chain of *John*. Because the -k link is not highest, it is not pronounced.

1. move(4a)

$$(\sout{\text{John}},\ \text{will},\ \text{rot}\ \sout{\text{John}}) : \texttt{+q}\ \texttt{s}, (\text{John},\ \texttt{-q})$$

2. move(4b)

$$\left(\left\{(\sout{\text{John}},\ \text{will},\ \text{rot}\ \sout{\text{John}}) : \texttt{+q}\ \texttt{s}, \begin{bmatrix} (\text{John},\ \texttt{-q}) \end{bmatrix}\right\}\right)$$

Finally, the highest link of *John* is positioned in the expression, and pronounced.

1. move(5a)

$$(\text{John } \sout{\text{John}}, \text{will}, \text{rot } \sout{\text{John}}) : \texttt{s}$$

2. move(5b)

$$(\{(\text{John } \sout{\text{John}}, \text{will}, \text{rot } \sout{\text{John}}) : \texttt{s}\})$$

2.1.1 Remnant Movement

Now what of copies within copies? The problem, under any theory, is that they are not part of the same chain as the thing they are a copy within a copy of. Consider the VP topicalized sentence in 3.14 below, which is 'underlyingly' as in 3.15 (each 'copy' in which is annotated with the feature determining its chain link).

(3.14) (. . . and) rot John will.

(3.15) [rot $\sout{\text{John}_{\texttt{d}}}$]$_{\texttt{-top}}$ John$_{\texttt{-q}}$ $\sout{\text{John}_{\texttt{-k}}}$ will $\sout{[\text{rot } \sout{\text{John}_{\texttt{d}}}]_{\texttt{v}}}$.

For concreteness' sake, let us assume that VP-topicalized sentences are derived by means of the lexical items ϵ::`=>v v -top`, which gives any vP a `-top` feature, and ϵ::`=s +top s`, which provides tensed clauses with positions in which subexpressions can check a `-top` feature. We work through a derivation of sentence 3.14 in the context of our external merge approach to copying, as it forces us to be explicit about a number of otherwise ignorable points.

As we will need two copies of the vP *rot John*, we select two copies of *rot*, and two copies of the chain *John* from the lexicon, merging them together. Since the merged link of the chain *John* is not the highest one in its chain, its phonological features are not incorporated into the derived expression.

1. merge(select(select(rot::=d v)), select(select(John::d -k -q)))

$$\left(\begin{array}{l} \left\{ \begin{array}{l} (\epsilon,\ \text{rot},\ \sout{\text{John}}) : \text{v},\ \left[\begin{array}{l} (\text{John},\ \text{-q}) \\ (\text{John},\ \text{-k}) \end{array} \right] \end{array} \right\} \\ \\ \left\{ \begin{array}{l} (\epsilon,\ \text{rot},\ \sout{\text{John}}) : \text{v},\ \left[\begin{array}{l} (\text{John},\ \text{-q}) \\ (\text{John},\ \text{-k}) \end{array} \right] \end{array} \right\} \end{array} \right)$$

Note that there are currently six tokens (deleted or not) of the string "John" in the expression above, but only four tokens in the expression 3.15 above. We next select a single copy of the topicalization feature introducing lexical item ϵ::=>v v -top, and merge it with the expression above.

2. merge(select(ϵ::=>v v -top), 1)

$$\left(\left\{ \begin{array}{l} (\epsilon,\ \text{rot},\ \sout{\text{John}}) : \text{-top},\ \left[\begin{array}{l} (\text{John},\ \text{-q}) \\ (\text{John},\ \text{-k}) \end{array} \right] \\ \\ (\epsilon,\ \text{rot},\ \sout{\text{John}}) : \text{v},\ \left[\begin{array}{l} (\text{John},\ \text{-q}) \\ (\text{John},\ \text{-k}) \end{array} \right] \end{array} \right\} \right)$$

We next select and merge one copy of the lexical item ϵ::=>v prog from the lexicon. We are at this point confronted with the fact that the cases of merge that we considered earlier were not exhaustive—now the chain links of the merged expression themselves contain moving subconstituents! Consider the expression we would have at this point had we been using instead the internal merge approach to copying.

$$(\epsilon,\ \sout{\text{rot}},\ \sout{\text{John}}) : \text{prog}, (\text{rot}\ \sout{\text{John}},\ \text{-top}), (\text{John},\ \text{-k}\ \text{-q})$$

This expression is the internal merge version of the external merge variant below.

$$\left(\left\{ (\epsilon, \sout{\texttt{rot}}, \sout{\texttt{John}}) : \texttt{prog}, \left[\ (\texttt{rot } \sout{\texttt{John}}, \texttt{-top}) \ \right], \left[\begin{array}{l} (\texttt{John}, \texttt{-q}) \\ (\texttt{John}, \texttt{-k}) \end{array} \right] \right\} \right)$$

The road from the representation in 2 to this one is straightforward, and can be used as the definition of the behaviour of merge in the case where chain links contain moving expressions:[6]

> When a chain link is put into storage, all of its moving subexpressions are eliminated.

This is not as arbitrary as it might at first appear. Instead of being 'eliminated', we might instead think of the (identical!) moving subexpressions of each chain link as being prepared for across-the-board (ATB) movement. In a derivational system like the one developed here, ATB movement is most naturally thought of as consisting of two steps; the first involves *collapsing* distinct tokens of identical material, and the second step is simply our normal movement operation. The question of how to determine whether the material in question is collapsable (i.e. identical) is, at least in the case of remnant movement under consideration here, immediately resolved: all moving subexpressions across chain links of the same chain are by virtue of our synchronization operations identical. We have, therefore, the below.

[6]On the semantic side, the elimination of the moving subexpressions should coincide with lambda abstraction over the resulting unbound variables in some predetermined order. This can be done purely mechanically, as unbound variables in the semantic representation are shadowed by their binders in storage. We may have to enrich our representation of stored semantic values somewhat (i.e. instead of $\mathtt{G}(\mathcal{Q})(\lambda_i)$ we can store the pair $\langle \mathcal{Q}, \lambda_i \rangle$).

In the case of vP topicalization, we want to make sure that the subject DP is interpreted high, so as to result in a property, and not a proposition, being topicalized (also capturing the fact that inverse scope is not generally possible in such constructions). There are numerous ways of ensuring this in (extensions to) our formal system; I will not consider them here.

3. merge(select(ϵ::=>v prog), 2)

$$\left(\left\{(\epsilon,\ \sout{\text{rot}},\ \sout{\text{John}}) : \texttt{prog},\ \left[\ (\text{rot}\ \sout{\text{John}},\ \texttt{-top})\ \right],\ \left[\begin{array}{l}(\text{John},\ \texttt{-q})\\(\text{John},\ \texttt{-k})\end{array}\right]\right\}\right)$$

We continue by selecting and merging a single copy each of our remaining tense and aspect lexical items.

4. merge(select(ϵ::=>prog perf), 3)

$$\left(\left\{(\epsilon,\ \sout{\text{rot}},\ \sout{\text{John}}) : \texttt{perf},\ \left[\ (\text{rot}\ \sout{\text{John}},\ \texttt{-top})\ \right],\ \left[\begin{array}{l}(\text{John},\ \texttt{-q})\\(\text{John},\ \texttt{-k})\end{array}\right]\right\}\right)$$

5. merge(select(will::=perf +k +q s), 4)

$$\left(\left\{(\epsilon,\ \text{will},\ \sout{\text{rot John}}) : \texttt{+k +q s},\ \left[\ (\text{rot}\ \sout{\text{John}},\ \texttt{-top})\ \right],\ \left[\begin{array}{l}(\text{John},\ \texttt{-q})\\(\text{John},\ \texttt{-k})\end{array}\right]\right\}\right)$$

We continue by positioning the final links of the chain *John*.

6. move(5)

$$\left(\left\{(\sout{\text{John}},\ \text{will},\ \sout{\text{rot John}}) : \texttt{+q s},\ \left[\ (\text{rot}\ \sout{\text{John}},\ \texttt{-top})\ \right],\ \left[\ (\text{John},\ \texttt{-q})\ \right]\right\}\right)$$

7. move(6)

$$\left(\left\{(\text{John}\ \sout{\text{John}},\ \text{will},\ \sout{\text{rot John}}) : \texttt{s},\ \left[\ (\text{rot}\ \sout{\text{John}},\ \texttt{-top})\ \right]\right\}\right)$$

We next select and merge one copy of the lexical item ϵ::=s +top s, which provides a +top specifier.

8. merge(select(ϵ::=s +top s), 7)

$$\left(\left\{(\epsilon,\ \epsilon,\ \text{John}\ \sout{\text{John}}\ \text{will}\ \sout{\text{rot John}}) : \texttt{+top s},\ \left[\ (\text{rot}\ \sout{\text{John}},\ \texttt{-top})\ \right]\right\}\right)$$

Finally, we move the vP in the above, checking all outstanding features.

9. move(8)

$$(\{(\text{rot } \sout{\text{John}},\ \epsilon,\ \text{John } \sout{\text{John}} \text{ will } \sout{\text{rot John}}) : \mathtt{s}\})$$

As we can see, so long as the pronunciation of chain links within copies obeys the simple rule of "all chain links within a copy are pronounced just as the corresponding chain link within another copy is pronounced," copies within copies (within copies...) give us no difficulty. This is not particular to our pronounce highest scheme, holding more generally for all of the more complex pronunciation schemes we will introduce in this chapter.

2.1.2 Variations on a Theme

To ensure that the highest chain link was pronounced, we computed in the course of the derivation whether a particular chain link was 'last,' pronouncing it just in that case. It is straightforward to formulate variations of pronounce highest, such as 'pronounce lowest', as well as 'pronounce second-highest', 'pronounce second-lowest' etc. We might alternatively wrest control of the pronunciation scheme from the grammar by marking the feature of the chain link to be pronounced with some diacritic (perhaps $\mathtt{-\hat{f}}$ or $\mathtt{\hat{f}}$). A lexicon is a 'pronounce highest' lexicon just in case all and only the last feature in a lexical item's feature bundle is so marked, etc.[7] This makes the pronunciation scheme a locus of linguistic variation, with some languages employing one, and some another. Additionally, we can formulate 'mixed' pronunciation schemes, with some lexical items being pronounced highest in their chains, and others second-highest, etc. In essence, each chain determins

[7]Were we to relax the requirement that feature bundles be totally ordered, pronounce highest and variants would no longer be representable in these terms. It would then become an empirical question which pronunciation scheme (derivational or lexical) were most enlightening.

its own pronunciation regimen.

2.2 Local Nondeterminism

Bošković [20] discusses a class of exceptions uniform across the Slavic languages to the robust generalization that wh-phrases in these languages may not remain *in-situ*. Examples 3.16 and 3.17 are illustrative of the general pattern of multiple wh-movement in Serbo-Croatian—all wh-words must be fronted.

(3.16) Ko šta kupuje?
who what buys
'Who buys what?'

(3.17) ?**Ko kupuje šta*?

However, as the examples below show, this is not always the case. Examples 3.18 and 3.19 show exactly the opposite pattern of judgements from the above.

(3.18) ?**Šta šta uslovjava*?

(3.19) Šta uslovjava šta?
what conditions what

If an adverb intervenes between the wh-phrases, then the 'expected' pattern of judgements reemerges.

(3.20) Šta neprestano šta uslovjava?
what constantly what conditions
'What constantly conditions what?'

(3.21) ?**Šta neprestano uslovjava šta*?

The generalization seems to be that an object wh-word may surface *in-situ* just in case if it had surfaced in its moved position, it would have been adjacent to

an identical wh-word. With a somewhat teleological spin, this can be rephrased in terms of the object wh-word avoiding an undesirable phonological outcome by surfacing in a non-standard position. One natural implementation of this idea (pursued by Bošković) is to claim that sentences like 3.19 have the same gross syntactic structure as their multiple wh-fronted brethren 3.20 and 3.16, and that the difference in surface form stems from a different chain link being pronounced.

Pronounce highest (and variations thereupon) has a local character in that the choice of which chain link to pronounce can be lexicalized—all the relevant information is present already in the lexical item itself. However, (Bošković's [20] analysis of) the Serbo-Croatian data above suggest the need for a more 'global' determination of which chain link to pronounce, one which takes into account the derived environment of each link in the chain.[8] We begin with a simple kind of derived environmental consideration, whether or not a chain link is in the projection of a head with a particular property P, and show how we can pronounce the link which is in the highest such position ('pronounce highest P'), without abandoning our strong approach to phases which disallows the construction of a derived tree. Our solution to this problem provides an insight into the relation between the grammatical operations of move and the more recent agree [29, 30]—agree is simulable by the move operation, if we adopt the 'pronounce highest EPP' spellout regimen we introduce below. The mechanism we adopt for implementing the phase-compatible 'pronounce highest P' spellout regimen fits naturally with interface constraints such as a ban on adjacent identical morphemes (such as seems to be necessary for a description of the Serbo-Croation data above), and

[8] We could of course encode (some finite amount of) information about syntactic environment into syntactic features, and stick with some member of the pronounce-highest family. However in extreme cases this would result in an explosive increase in the size of the lexicon, as each lexical item would need to encode how it would be pronounced in each possible syntactic environ.

allows for the implementation of a Serbo-Croatian-style pronunciation scheme.

2.2.1 Agree and Pronounce Highest EPP

Let's imagine that our selector and licensor features came in two varieties; the 'normal' variety (=f, =>f, f=>, +f) and the 'EPP' variety ($=\hat{f}, =>\hat{f}, \hat{f}=>, +\hat{f}$). Let's assume further that the EPP variety behaves the same as the normal variety from a syntactic perspective (e.g. +f and $+\hat{f}$ both check the same feature -f). The difference between the two kinds of feature lies only in their very indirect effects on the pronounced form of the generated expression: a chain is pronounced in the highest of its links that is checked by an EPP feature (and in its lowest link if none are checked by an EPP feature). In order to determine where a chain should be pronounced without having to first construct a tree, we need to keep track of *whether* a chain has already been pronounced (a single bit of information); if it has, then it must not encounter another EPP feature, if it hasn't, then it must!

We enrich our representation of moving subconstituents by associating with each a boolean value ('$\circ$' indicates that the subconstituent in question has *not* been pronounced, '$\bullet$' indicates that it *has*). An intermediate stage in the derivation of *a carcass was devoured* looks like the following, under this augmentation of the internal merge approach to copying.

$$(\epsilon, \text{devour -en}, \sout{\text{a carcass}}) : \texttt{pass}, (\text{a carcass} \circ \texttt{-k -q}) \tag{3.22}$$

Our grammar also generates the following expression, which differs from the above only in that the moving subconstituent has been spellt-out in its first merge position.

$$(\epsilon, \text{devour -en}, \text{a carcass}) : \texttt{pass}, (\text{a carcass} \bullet \texttt{-k -q}) \tag{3.23}$$

Let's assume that case features (+k) are uniformly EPP marked (a generalization

of the idea that finite T bears an EPP feature in English), and that scope features (`+q`) are uniformly not (in accord with the 'covert' perspective on quantifier movement). Then on this assumption, of the two expressions above only the former can converge, as when *a carcass* tries to check its `-k` feature against its EPP-marked counterpart, the derivation will crash if it turns out that *a carcass* has been spellt-out in a non-highest EPP chain-link.

The utility of such a pronunciation scheme lies in the simple account we may give to cases of long-distance agreement, as exemplified by the sentences below.

(3.24) There was/*were devoured a rotting carcass.

(3.25) There *was/were devoured three rotting carcasses.

(3.26) There seems/*seem to have been devoured a rotting carcass.

(3.27) There *seems/seem to have been devoured three rotting carcasses.

The conditions licensing expletive-*there* (the nature of the predicative element, as well as the restrictions on the associated DP) are quite complex, and will not be done justice here. Instead, we concentrate our attention exclusively on expletive-*there* in passive sentences, and ignore the role the associated DP plays in licensing expletive-*there*. As it is the long-distance agreement rather than the expletive itself that interests us here, we simply introduce a rule that inserts *there* into passive sentences, and ignore the particulars of the features that *there* has. We write

`there::@(pass,k)`

to indicate that, whatever the structure and nature of the feature bundle associated with *there* is, it has the function of merging with a passive participle, and

becoming associated with the case of an element with both a -k and at least one additional feature, which transfers the -k feature from that element to it. Accordingly, *there*-insertion acts on the expressions above to yield the below.

1. *there*-insertion(3.22, there::@(pass,k))

   (ϵ, devour -en, ~~a carcass~~) : pass, (a carcass ∘ -q), (there ∘ -k)

2. *there*-insertion(3.23, there::@(pass,k))

   (ϵ, devour -en, a carcass) : pass, (a carcass • -q), (there ∘ -k)

We continue with the derivation as usual, merging *be*, and then our empty aspectual heads, followed by *-ed*.

1. merge(be::=pass v, 1a)

   (ϵ, be, devour -en ~~a carcass~~) : v, (a carcass ∘ -q), (there ∘ -k)

2. merge(be::=pass v, 1b)

   (ϵ, be, devour -en a carcass) : v, (a carcass • -q), (there ∘ -k)

1. merge(ϵ::=>v prog, 2a)

   (ϵ, be, devour -en ~~a carcass~~) : prog, (a carcass ∘ -q), (there ∘ -k)

2. merge(ϵ::=>v prog, 2b)

   (ϵ, be, devour -en a carcass) : prog, (a carcass • -q), (there ∘ -k)

1. merge(ϵ::=>prog perf, 3a)

   (ϵ, be, devour -en ~~a carcass~~) : perf, (a carcass ∘ -q), (there ∘ -k)

2. merge(ϵ::=>prog perf, 3b)

   (ϵ, be, devour -en a carcass) : perf, (a carcass • −q), (there ○ −k)

1. merge(-ed::perf=> +$\hat{k}$ +q s, 4a)

   (ϵ, ϵ, be -ed devour -en ~~a carcass~~) : +$\hat{k}$ +q s, (a carcass ○ −q), (there ○ −k)

2. merge(-ed::perf=> +$\hat{k}$ +q s, 4b)

   (ϵ, ϵ, be -ed devour -en a carcass) : +$\hat{k}$ +q s, (a carcass • −q), (there ○ −k)

Now we check the EPP-marked +k feature on the head of the expressions above against the −k feature of *there*. As *there* was not pronounced earlier in either expression (indicated by the ○), it may be pronounced in this EPP position (and indeed must be, as this is the highest position it will be in).

1. move(5a)

   (there, ϵ, be -ed devour -en ~~a carcass~~) : +q s, (a carcass ○ −q)

2. move(5b)

   (there, ϵ, be -ed devour -en a carcass) : +q s, (a carcass • −q)

The last step in the derivation is to check the non-EPP-marked +q feature of the tense head. Because it does not bear an EPP diacritic, it does not license the pronunciation of a moved element, and so only elements may be attracted to it that are not in need of being pronounced there. Thus expression 6a crashes at this step, as *a carcass* has not been pronounced (○), and, as the −q feature is its last, must be now.

1. move(6a)

Crash!!!

2. move(6b)

(there, ϵ, be -ed devour -en a carcass) : s

As the object DP *a carcass*, although pronounced in situ, stands in a feature-checking relationship with the inflected verb, the stage is set for a more articulated theory of features and feature checking to explain the long-distance agreement facts. The relationship between this pronounce-highest EPP scheme and the agree-based syntax developed in [29, 30] is straight-forward; move as portrayed here just *is* agree.

2.2.2 Morphological Filters

Returning to the Serbo-Croatian data, it seems here that it is not some syntactic feature which determines the pronounced position of the wh-words, but rather a morpho-phonological ban on adjacent identical wh-words. In other words, non-syntactic properties seem to be playing the role of the EPP diacritics we explored above. Here we will sketch how this might be effected without compromising our view of convergence as an intrinsic property of syntactic derivations, determined without reference to other derivations. We will assume a more articulated PF branch, with a layer of 'morphology' (MF) between the output of syntax and the sequences of phonemes we have come to know and love (see figure 3.2). We shift the determination of which chain link to pronounce from our syntax to the morphology. Our syntax should generate therefore both the grammatical 3.16 and 3.19 as well as the ungrammatical 3.17 and 3.18 (repeated below).

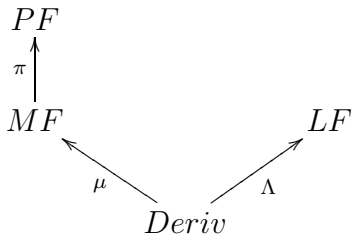


Figure 3.2: A Morphological Component

(3.16) Ko šta kupuje?
who what buys
'Who buys what?'

(3.17) ?*Ko kupuje šta?

(3.18) ?*Šta šta uslovjava?

(3.19) Šta uslovjava šta?
what conditions what

A simple ban on adjacent homophonous wh-words will suffice to rule out 3.18, but 3.17 is incorrectly let through to PF. What we want is a way to ensure that *if* a wh-word is pronounced in a non-highest position, all higher positions involve otherwise adjacent homophonous wh-words. This information is present already in our syntax, we need simply to hand it to MF. This we do by allowing syntax to insert 'anti-morphemes' in chain positions, which indicate that a particular morpheme *would have* appeared in a certain position, had it not been pronounced earlier. Anti-morphemes (m^{-1}) delete under string adjacency with the corresponding morpheme (m). MF, then, refuses to let through to PF any object with anti-morphemes (or equivalently, PF cannot interpret anti-morphemes). In the case of Serbo-Croatian (and apparently the other slavic and slavic-influenced languages) we need to say that an anti-morpheme is inserted in every chain position

higher than the one chosen as the pronounced link. If this were a general property of languages (that anti-phrases were introduced in all (EPP marked) chain positions higher than the pronounced link), we would have, due to a general lack of homophony, a justification for (something like) our original 'pronounce highest' pronunciation scheme—anti-morphemes cannot in general be eliminated, thus causing all derivations but the one in which the highest link is pronounced to crash. As this is a relatively simple (and principled) approach to anti-morpheme insertion, we will adopt it here.

Now we can sketch an analysis of the Serbo-Croatian data along the lines of Bošković [20]. We have located the source of the ungrammaticality of 3.17 and 3.18 in the MF component of grammar. Accordingly, we give the morphological form of sentences 3.16–3.19 below as 3.28–3.31.

(3.28) *Ko šta kupuje*?

(3.29) ?**Ko šta*$^{-1}$ *kupuje šta*?

(3.30) ?**Šta šta uslovjava*?

(3.31) *Šta šta*$^{-1}$ *uslovjava šta*?

Sentence 3.29 is ungrammatical because of the general prohibition of anti-morphemes at PF. This contrasts with 3.31, in which the anti-morpheme can be deleted under adjacency with its positive counterpart within the morphological component before being sent to PF. Sentence 3.30, on the other hand, is ruled out by the language particular ban against adjacent homophonous wh-words.

What remains to be determined is whether the lower instance of *šta* in example 3.19 is the object, or the subject, of the verb *uslovjava*. This question can be answered only if there are canonical 'subject' and 'object' properties that we

can identify, and then determine the lower copy of *šta* to have. For example, if there were canonical subject or object positions that were identifiable in terms of their position in the string relative to other material (adverbs, say), we could ask whether the lower copy of *šta* could only be in the canonical subject or object positions relative to this other material. If it could appear in both canonical subject and object positions, this would suggest that either *šta* could be pronounced in situ—something our current system predicts. Sentence 3.32 depicts the morphological form of a subject in situ variant of 3.19, and 3.33 the morphological form of an ungrammatical sentence (as neither anti-morpheme can be deleted), with both wh-words in situ.

(3.32) *Šta*$^{-1}$ *šta uslovjava šta*?

(3.33) **Šta*$^{-1}$ *šta*$^{-1}$ *uslovjava šta šta*?

However, Bošković [20] analyzes the lower *šta* as the object, noting that there is a strong tendency in Serbo-Croatian to interpret syntactically ambiguous pairs of noun phrases with the linearly first noun phrase as the subject, and the second as the object. As we have just seen, our current approach to anti-morphology does not allow us to discriminate between the in-situ derivations of the subject and the object. Accepting Bošković's analysis, we are led to a slightly richer view of (anti-)morphology, with anti-morphemes specifying the direction they are looking for their correspondants. Accordingly, we have two kinds of anti-morphemes, with m^r wanting to be to the right, and m^ℓ to the left, of its corresponding morpheme m. We may stipulate that natural languages (or at least Serbo-Croatian) make use only of right anti-morphemes (m^r). Now only the object in situ sentence is derivable; the subject in situ sentence contains an undeletable right anti-morpheme *šta*^{r}.

2.3 Pronouncing Multiple Copies

In Yoruba, as in many other West African languages, when predicates are clefted, two copies appear: one in the canonical left-peripheral cleft position (where it is nominalized), and one in the standard in situ predicate position. Given a simple intransitive sentence like 3.34, the accompanying predicate clefted sentence is as in 3.35.

(3.34) Tolu ku.
Tolu die
"Tolu died"

(3.35) Kiku ni Tolu ku.
Dying NI Tolu die
"Tolu DIED (not something else)."

In chapter 4 we investigate in detail the properties of this construction, arguing that it (and similar constructions in unrelated languages) provides persuasive evidence for the necessity of an operation which copies arbitrarily large swathes of structure. For now, and for concreteness, we pretend that English has a construction along the lines of examples 3.34 and 3.35, such that in VP topicalization constructions both copies of the VP are pronounced (cf. 3.36).

(3.36) Rot John will rot.

The question we now turn to is how, given our current lexical items, we can generate sentences like 3.36.

Adopting the variant of the pronounce highest scheme according to which pronunciation is determined by a featural diacritic, we may simply mark multiple features with this diacritic. Accordingly, the lexical item which introduces the `-top` feature has exceptionally two such featural diacritics: ϵ::`=>v` $\hat{\texttt{v}}$ `-t`$\hat{\texttt{o}}$`p`.

A different, less intrinsically determined, approach to the pronunciation of multiple copies is to locate the diacritic demanding pronunciation of a chain link not internal to the chain, but on a feature (+$\hat{\text{f}}$, =$\hat{\text{f}}$, =>$\hat{\text{f}}$, or $\hat{\text{f}}$=>) of a different expression. This basic approach is in line with Koopman [95], who argues that heads can sometimes require pronounced material in their specifiers, and with Nunes [131], who appeals to an operation of 'reanalysis', which essentially ensures that the phonological features of a particular phrase will be pronounced. Incorporating into our pronounce highest regimen the possibility of heads requiring that their specifiers host pronounced material, we mark on the lexical items which select v that they force pronunciation of this constituent: ϵ::=>$\hat{\text{v}}$ perf and -ing::=>$\hat{\text{v}}$ perf. Under this approach, we might expect that only some v-selecting heads will permit reanalysis, so that alongside 3.36 we could have 3.37 as well (in case *-ing* did not require reanalysis).[9]

(3.37) Rot John is doing.

Adding instead a mechanism of reanalysis to a pronouce highest P regimen, we obtain a system similar to Nunes's, according to which reanalysis 'removes' a chain link from consideration of the pronounce highest pronunciation scheme. In other words, Nunes [131] provides a 'pronounce highest non-reanalyzed link' spellout regimen. Accordingly, we may locate the reanalysis diacritic either on the v-selecting heads, or on the feature licensing the -top constituent: ϵ::=s +t$\hat{\text{o}}$p s. Adopting this latter approach, we derive sentence 3.36 as per the following.

1. merge(ϵ::=>v v -top, merge(rot::=d v, John::d -k -q))

$$(\epsilon, \text{rot}, \cancel{\text{John}}) : \text{v -top}, (\text{John} \circ \text{-k -q})$$

[9] *Do*-support is easy enough to hack into the system, but I do not have (nor do I know of) a principled account of this or similar phenomena.

If, at this next step, we do not pronounce the chain of *rot* ~~*John*~~, hoping that there will be a later, higher, non-reanalyzed position, the derivation will crash once this is determined not to be the case.

2. merge(ϵ::=>v prog, 1)

$$(\epsilon,\ \text{rot},\ \sout{\text{John}}) : \texttt{prog}, (\text{rot}\ \sout{\text{John}} \bullet \texttt{-top}), (\text{John} \circ \texttt{-k}\ \texttt{-q})$$

We continue with the derivation as shown below.

3. move(move(merge(will::=perf +k +q s, merge(ϵ::=>prog perf, 2))))

$$(\text{John}\ \sout{\text{John}},\ \text{will},\ \text{rot}\ \sout{\text{John}}) : \texttt{s}, (\text{rot}\ \sout{\text{John}} \bullet \texttt{-top})$$

We next add a topic-licensing position to the expression above.

4. merge(ϵ::=s +tôp s, 3)

$$(\epsilon,\ \epsilon,\ \text{John}\ \sout{\text{John}}\ \text{will rot}\ \sout{\text{John}}) : \texttt{+tôp s}, (\text{rot}\ \sout{\text{John}} \bullet \texttt{-top})$$

Finally, we check the -top feature of the vP, which triggers reanalysis of the moving phrase.

5. move(4)

$$(\text{rot}\ \sout{\text{John}},\ \epsilon,\ \text{John}\ \sout{\text{John}}\ \text{will rot}\ \sout{\text{John}}) : \texttt{s}$$

Had we not pronounced the vP in its base position in step 2, then at step 4 we would have had the following expression, which cannot be derived further, as the moving constituent 'gambled' incorrectly that there would be a later non-reanalyzing chain link.

4. merge(ϵ::=s +tôp s, 3′)

$$(\epsilon,\ \epsilon,\ \text{John}\ \sout{\text{John}}\ \text{will rot}\ \sout{\text{John}}) : \texttt{+tôp s}, (\text{rot}\ \sout{\text{John}} \circ \texttt{-top})$$

3 Summary

In this chapter we have explored two perspectives on copying structure, both of which are compatible with our elimination of all syntactic levels of representation, and consequently with a strong theory of phases. Although they take copying to be of different structures (the internal merge approach takes copying to be of PF (or MF) representations, while the external merge approach takes copying to be of the derivation), both approaches to copying structure we have developed here are equivalent in their expressive power. Additionally, even with copying, minimalist grammars are efficiently parsable. (These issues and others are taken up in the appendices to this chapter.) We have seen how sophisticated pronunciation strategies can be defined without making reference to derived trees, much less competition between derivations, interacting violable constraints, or other powerful and ill-understood formal mechanisms. Indeed, our derivational approach to pronunciation of copies gives us a simple and elegant treatment of remnant movement constructions, which have proven quite tricky to deal with at the level of the derived tree [62, 131, 132]. A treatment of long-distance agreement was proposed, and our fragment of English extended to deal with a well-defined subclass of *there*-insertion phenomena. Considering exceptions to the generalization that all wh-words front in Serbo-Croatian multiple wh-sentences, we have seen that making the choice of which chain link to pronounce dependent on morphophonological factors can be done in a morphological component, without sacrificing the 'one-pass' approach to grammar we have developed here. Indeed, in so doing we have arrived at a possible explanation for the overwhelming tendency for just a single link in each chain to be pronounced.

Although the phrase 'linearization of syntactic objects' occurs only this once in this chapter, and, indeed, no mention at all is made of complicated algorithms

for determining how to pronounce derived multiple-dominance structures, this is one of the main topics of this chapter. While the sophisticated pronunciation strategies we have developed could very well be re-stated over structures with copies, the relative elegance and simplicity of the derivational treatment given here speaks for itself.

Appendices

C–1 Two ways to copy

What seems to be the standard perspective on the 'meaning' of syntactic structure in the generative community is that the derived tree structures associated with expressions are best understood as representing actual syntactic objects, and that two such objects are identical only if their tree structures are. Copying then targets subparts of this structure, creating a new structure which has subparts which are identical. In a strict theory of phases, where interface instructions are interpreted as they are built, this amounts to copying the objects of the domains interfacing with syntax (minimally semantics and phonology) but over syntactically defined phrases. Thus, there is no copying of anything syntactic—copying is moved from the domain of syntax into the domain of phonology (and perhaps also of semantics).

Another perspective we might take is explicit in the categorial grammar community:

> [...]syntactic structure is merely the characterization of the process of constructing a logical form, rather than a representational level of structure that actually needs to be built[...] [pg. *xi* in 163]

According to this view, the structure copied is merely a record of the derivation of a syntactic object, and not an explicitly represented property of that object itself. Two syntactic objects would then have the same structure just in case they were derived in the same way. Copying structure thus becomes enforcing identity of derivation. In other words, we guarantee that two constituents have the same structure not by copying some complex object, but by building these constituents

up in the same way. This allows us to factor the operation of copying arbitrarily large swathes of syntactic material into a series of atomic (or at least boundedly large) steps in the following manner. Regardless of how large two expressions are, if we are assured that they are identical, then applying any polynomial function of one variable to them is guaranteed to preserve their identity.[10]

C–1.1 Copying the Derived Tree

Copying the derived tree is obtained by an immediate extension of the generating functions from our previous chapter. Instead of pronouncing the merged or moved expression just in case the current step involves its last syntactic feature (merge1, merge2, move1, affixRaise and affixLower), *all* feature checking results in pronunciation of the expression merged or moved.

The resulting system is no longer equivalent to MCFGs (which are equivalent to minimalist grammars without copying), as the former, but not the latter, can easily generate the language a^{2^n} (using the three lexical items in figure 3.3), which is not even semilinear. Below is a derivation using the lexicon of figure 3.3. At

`a::s` ϵ`::=s x -y`

ϵ`::=x +y s`

Figure 3.3: A grammar for the language a^{2^n}

steps 3, 6 and 9 we have sentences (expressions of type `s`) of length 2, 4, and 8, resepctively. Clearly, no sentences are derivable with length between 2 and 8 other than these.

[10] A unary function f is said to preserve a relation R just in case for every x and y in the domain of f, $xRy \rightarrow fxRfy$

1. merge(ϵ::=s x -y, a::s)

$$(\epsilon,\ \epsilon,\ \mathsf{a}) : \mathtt{x}\ \mathtt{-y}$$

2. merge(ϵ::=x +y s, 1)

$$(\epsilon,\ \mathsf{a},\ \epsilon) : \mathtt{+y}\ \mathtt{s},\ (\mathsf{a},\ \mathtt{-y})$$

3. move(2)

$$(\mathsf{a},\ \mathsf{a},\ \epsilon) : \mathtt{s}$$

4. merge(ϵ::=s x -y, 3)

$$(\epsilon,\ \epsilon,\ \mathsf{aa}) : \mathtt{x}\ \mathtt{-y}$$

5. merge(ϵ::=x +y s, 4)

$$(\epsilon,\ \mathsf{aa},\ \epsilon) : \mathtt{+y}\ \mathtt{s},\ (\mathsf{aa},\ \mathtt{-y})$$

6. move(5)

$$(\mathsf{aa},\ \mathsf{aa},\ \epsilon) : \mathtt{s}$$

7. merge(ϵ::=s x -y, 6)

$$(\epsilon,\ \epsilon,\ \mathsf{aaaa}) : \mathtt{x}\ \mathtt{-y}$$

8. merge(ϵ::=x +y s, 7)

$$(\epsilon,\ \mathsf{aaaa},\ \epsilon) : \mathtt{+y}\ \mathtt{s},\ (\mathsf{aaaa},\ \mathtt{-y})$$

9. move(8)

$$(\mathsf{aaaa},\ \mathsf{aaaa},\ \epsilon) : \mathtt{s}$$

The languages generated by minimalist grammars which copy the derived tree are easily shown to be *parallel* MCFLs [151];[11] the proof of [121] showing the inclusion

[11]Parallel MCFLs are a proper subclass of **P**, the languages recognizable in polynomial time on a deterministic turing machine. **P** is the intersection closure of the class PMCFL (see [112] for details).

of the minimalist languages in the MCFLs needs only minor and uninteresting modifications to show the inclusion of the minimalist languages with copying in the parallel MCFLs.

Although the copying itself can be done efficiently, it is possible to have such a complex way of determining which copies should be spellt out that the resulting family of languages is no longer contained in the PMCFLs, or even in **P**.[12] As long as the information relevant to pronunciation in our expressions is finitely bounded, the containment proof will be able to go through. Indeed, this has been behind our insistence on the treeless presentation of our MG modifications. All of the pronunciation schemes explored in this chapter are efficient in this sense of not taking us outside the expressive power of PMCFGs. The only even mildly interesting one such from the perspective of Michaelis's [121] proof is the 'pronounce highest P' family, which require us to record whether a moving constituent has been pronounced or not, and which results in multiplying the number of non-terminal symbols in the simulating PMCFG (due to the fact that each possible combination of such booleans on moving constituents must be represented by a separate non-terminal).

C–1.2 Copying the Derivation Tree

To implement copying of the derivation in minimalist grammars (without explicitly representing the derivation as a syntactic object in its own right), we need to expand our ontology of expressions to include syntactic composites of 'simple' expressions. A composite $\vec{e} = \langle e_1, \ldots e_n \rangle$ will be understood as indicating that each e_i and e_j, for $1 \leq i, j \leq n$, are copies of each other. For the move operation, our intended interpretation of these composite expressions leads us to the

[12] Or even in the class of recursively enumerable languages, if we choose our pronunciation scheme unwisely.

following identity, which we shall see later how to implement:

$$move(\vec{e}) = \langle move(e_1), \ldots, move(e_n) \rangle \tag{3.38}$$

The intended interpretation of these composites suggests that something like the following equation should be true of the merge operation (although we shall have cause to revise this later):

$$merge(\vec{d}, \vec{e}) = \langle merge(d_1, e_1), \ldots, merge(d_n, e_n) \rangle \tag{3.39}$$

While these schema are sufficient to generate syntactic composites all of whose parts are copies, we still lack a way to insert these copies into a single expression.[13] To do this, we want merge to do something like the following:

$$merge(d, \langle e_1, \ldots, e_n \rangle) = f \tag{3.40}$$

In equation 3.40, the intended interpretation of the composite $\vec{e}$ is that it is a chain—a single expression which is in multiple positions at once. *Ex hypothesi*, this chain is a movement chain, and thus the distribution of its parts should be left up to the move operation. Note that this is quite different from the previous interpretation assigned to these syntactic composites, namely, that of being identical copies, the relative distribution of which not even coming into question. We shall soon see that these interpretative differences will be reflected in our ontology. Let us begin then by examining the notion of a movement chain. In a (non-trivial) movement chain, each position past the first (i.e., bottom-most) contains the result of moving (here, copying) the contents of the previous position. Thus the elements comprising a movement chain are not necessarily identical, in so far as they may be moving to satisfy different requirements. In the context

[13] For some purposes (e.g. translation or interpretation [2, 130]) this may be sufficient.

of the minimalist grammar system, they may have distinct selectee or licensee features. Given a lexical item $\ell = \mathtt{s} :: \gamma\ \mathtt{x}\ \mathtt{-w_1} \cdots \mathtt{-w_n}$, we create the **copy chain**[14]

$$\vec{\ell} := \left\{ \begin{array}{c} \mathtt{s} :: \gamma\ \mathtt{-w_n} \\ \vdots \\ \mathtt{s} :: \gamma\ \mathtt{-w_1} \\ \mathtt{s} :: \gamma\ \mathtt{x} \end{array} \right\}$$

We will call a copy chain non-trivial just in case it contains at least two links. Note that all and only lexical items with licensee features give rise to non-trivial copy chains.

Recall that a standard minimalist expression is a pair $\langle \phi_0, \mathcal{N} \rangle$, where $\mathcal{N}$ is a sequence $\langle \phi_1, \ldots, \phi_k \rangle$ of moving expressions, the distribution of which is determined by the move operation. In order to allow the links of copy chains to be positioned by movement (i.e. in order to assure that copy chains are movement chains) we intutively identify each ϕ_i with a copy chain (although these we write within square brackets ('[' and ']', and no longer keep track of the phonological features of the head seperately). Thus, a non-composite expression has the form:

$$\langle \phi_0, \langle \phi_1, \ldots, \phi_k \rangle \rangle$$

where $\phi_0 \in \Sigma^* \times \Sigma^* \times \Sigma^* \times \{::, :\} \times \mathbf{Syn}^*$, and for $1 \leq i \leq k$, $\phi_i \in (\Sigma^* \times \mathbf{Syn})^+$.

Move is defined over simple expressions as per the following two subcases, and then extended pointwise over copy chains, and then again over sequences of copy

[14]We adopt here without further ado the notational convention of writing sequences $\langle a_1, \ldots, a_k \rangle$ 'vertically' as

$$\left\langle \begin{array}{c} a_k \\ \vdots \\ a_1 \end{array} \right\rangle$$

The choice of brackets, while technically irrelevant for the formalism, will be used to indicate 'type', with curly brackets ('{', '}') being reserved for copy chains, and parentheses for sequences thereof.

chains:

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\ (t, \texttt{-x})\ \right] \beta}{(ts_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \begin{bmatrix} \vdots \\ (t', \chi) \\ (t, \texttt{-x}) \end{bmatrix} \beta}{(ts_1, s_2, s_3) : \gamma, \alpha \begin{bmatrix} \vdots \\ (t', \chi) \end{bmatrix} \beta}\text{move2}$$

In the case where its first argument is simple, merge takes a copy chain as its second argument, and is defined as per the following cases (merge1–4, affixRaise1–2, and affixLower):

$$\frac{(s_1, s_2, s_3) :: \texttt{=x}\gamma, \alpha \quad \left\{\ (t_1, t_2, t_3) \cdot x, \beta\ \right\}}{(s_1, s_2, s_3 t_1 t_2 t_3) : \gamma, \alpha\beta}\text{merge1}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{=>x}\gamma, \alpha \quad \left\{\ (t_1, t_2, t_3) \cdot x, \beta\ \right\}}{(s_1, t_2 s_2, s_3 t_1 t_3) : \gamma, \alpha\beta}\text{affixRaise1}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{x=>}\ \gamma, \alpha \quad \left\{\ (t_1, t_2, t_3) \cdot x, \beta\ \right\}}{(s_1, \epsilon, s_3 t_1 t_2 s_2 t_3) : \gamma, \alpha\beta}\text{affixLower}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=x}\gamma, \alpha \quad \left\{\ (t_1, t_2, t_3) \cdot x, \beta\ \right\}}{(t_1 t_2 t_3 s_1, s_2, s_3) : \gamma, \alpha, \beta}\text{merge2}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{=x}\gamma, \alpha \quad \left\{ \begin{array}{c} \vdots \\ (t'_1, t'_2, t'_3) \cdot \chi, \beta' \\ (t_1, t_2, t_3) \cdot x, \beta \end{array} \right\}}{(s_1, s_2, s_3 t_1 t_2 t_3) : \gamma, \alpha \begin{bmatrix} \vdots \\ (t'_1 t'_2 t'_3, \chi) \end{bmatrix} \beta}\text{merge3}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{=>x}\gamma, \alpha \quad \left\{ \begin{array}{c} \vdots \\ (t'_1, t'_2, t'_3) \cdot \chi, \beta' \\ (t_1, t_2, t_3) \cdot x, \beta \end{array} \right\}}{(s_1, t_2 s_2, s_3 t_1 t_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t'_1 t'_2 t'_3, \chi) \end{array} \right] \beta} \text{affixRaise2}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=x}\gamma, \alpha \quad \left\{ \begin{array}{c} \vdots \\ (t'_1, t'_2, t'_3) \cdot \chi, \beta' \\ (t_1, t_2, t_3) \cdot x, \beta \end{array} \right\}}{(t_1 t_2 t_3 s_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t'_1 t'_2 t'_3, \chi) \end{array} \right] \beta} \text{merge4}$$

When the first argument to merge is a copy chain, the second argument is a sequence of copy chains, the length of which is equal to that of the first, and

$$merge(\left\{ \begin{array}{c} e_k \\ \vdots \\ e_1 \end{array} \right\}, \left(\begin{array}{c} \Delta_k \\ \vdots \\ \Delta_1 \end{array} \right)) := \left\{ \begin{array}{c} merge(e_k, \Delta_k) \\ \vdots \\ merge(e_1, \Delta_1) \end{array} \right\}$$

And finally when both arguments to merge are sequences of copy chains,

$$merge(\left(\begin{array}{c} \Gamma_k \\ \vdots \\ \Gamma_1 \end{array} \right), \left(\begin{array}{c} \Delta_\ell \\ \vdots \\ \Delta_1 \end{array} \right)) := \left(\begin{array}{c} merge(\Gamma_k, \left(\begin{array}{c} \Delta_\ell \\ \vdots \\ \Delta_{1+\Sigma_{i=1}^{k-1}|\Gamma_i|} \end{array} \right)) \\ \vdots \\ merge(\Gamma_1, \left(\begin{array}{c} \Delta_{|\Gamma_1|} \\ \vdots \\ \Delta_1 \end{array} \right)) \end{array} \right)$$

None of our rules above require that the components of the sequences used are identical. To ensure this, we introduce a new operation, *select*, which copies lexical items (thereby reducing copying of arbitrarily complex derivations to copying of atomic derivations). The domain of *select* is the disjoint union of lexical items (expressions in $\Sigma \times \Sigma \times \Sigma \times \{::\} \times \mathbf{Syn}^*$) and the set of sequences of copy chains. For ℓ a lexical item, c_ℓ its associated copy chain, and $c_1, \ldots, c_k$ copy chains, $k \in \mathbb{N}$,

$$select(\ell) = \left\langle\ c_\ell\ \right\rangle$$

$$select(\left\langle \begin{array}{c} c_k \\ \vdots \\ c_1 \end{array} \right\rangle) = \left\langle \begin{array}{c} c_k \\ \vdots \\ c_1 \\ c_1 \end{array} \right\rangle$$

A synchronous minimalist grammar is completely determined by an arbitrary finite subset Lex of the lexical items, its lexicon. The expressions generated by a synchronous minimalist grammar are defined in the standard way, as the closure of its lexicon under the generating functions. The (string) language generated by a synchronous minimalist grammar $G = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \mathsf{c} \rangle$ is

$$L(G) := \{w \in \Sigma^* : G \vdash \left\langle\ \left\{\ w \cdot \mathsf{c}\ \right\}\ \right\rangle\}$$

C–1.3 The Complexity of Derivational Copying

Given a synchronous minimalist grammar $G_{syn} = \langle \Sigma, \mathbf{lic}, \mathbf{sel}, Lex, \mathsf{c} \rangle$, it is possible to construct a weakly equivalent minimalist grammar with copying, G_{copy}. Indeed, $G_{copy} = G_{syn}$. Furthermore, there is a clear sense in which the two grammars are strongly equivalent as well: there are structure preserving bijections from derivations in one grammar to those in the other, such that derivations of complete expressions of category c in one grammar are mapped to derivations

of complete expressions of category c in the other, and moreover the expressions are 'identical but for the bracketing', so that given d_{syn} and d_{copy} derivations of complete expressions e_{syn} and e_{copy} of category c in G_{syn} and G_{copy} respectively that are mapped to each other by these structure preserving expressions,

$$e_{syn} := \Big(\big\{ (\mathsf{t},\ \mathsf{u},\ \mathsf{v}) : \mathsf{c} \big\} \Big)$$
$$e_{copy} := (\mathsf{t},\ \mathsf{u},\ \mathsf{v}) : \mathsf{c}$$

But what structure is preserved? Intuitively, derivations in G_{syn} just are derivations in G_{copy} to which have been added various numbers of unary branching internal nodes over the leaves labelled by *select*. And so we can obtain the corresponding G_{copy} derivation from any G_{syn} derivation just by deleting these *select* branches. We can model this via a homomorphism, h.

$$h(select(\alpha)) = h(\alpha)$$
$$h(move(\alpha)) = move(h(\alpha))$$
$$h(merge(\alpha, \beta)) = merge(h(\alpha), h(\beta))$$

Going the other way, from derivations in G_{copy} to derivations in G_{syn}, the claim is that there exists a function inverse to h above. This by itself is uninteresting, as between any two equinumerous sets there exist functions which preserve any invariant you desire.[15] The interesting question is *how difficult it is to compute this function*. We can think of the synchronous derivation as computing numerical relationships between lexical items: the number of copies of each that is required. The numerical relationship is simple to understand. We say that a leaf ℓ_1 in the derivation tree *m-commands* a leaf ℓ_2 just in case ℓ_1 is the left-most leaf of some node dominating both ℓ_1 and ℓ_2, and ℓ_1 is not ℓ_2. ℓ_1 *immediately m-commands* ℓ_2 just in case ℓ_1 is the left-most leaf of the first node dominating ℓ_2 that ℓ_2 is not the left-most leaf of. In figure 3.4, ℓ_1 m-commands all other leaves, ℓ_2

[15] Adopting a (relatively standard) non-constructive account of functions.

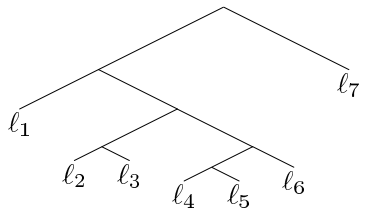


Figure 3.4: Computing m-command in the derivation tree

m-commands just ℓ_3, ℓ_4 m-commands ℓ_5 and ℓ_6, and ℓ_3, ℓ_5, ℓ_6 and ℓ_7 don't m-command anything. Now, each lexical item in a synchronous derivation tree needs to be selected as many times as the product of the number of categorial (`f`) and licensee (`-f`) features of each lexical item m-commanding it, unless it is the left-most leaf of the root, in which case it must be selected exactly once. We can give an inductive characterization of this relation in terms of immediate m-command as follows. We let n_ℓ be the number of select branches dominating leaf ℓ, and f_ℓ the sum of the number of categorial and licensee features of ℓ. Then

$$n_\ell = \begin{cases} 1 & \text{if } \ell \text{ is the left-most} \\ & \text{leaf of the root} \\ n_{\ell'} \times f_{\ell'} & \text{otherwise, for } \ell' \text{ the immediate} \\ & \text{m-commander of } \ell \end{cases}$$

We can compute this function in two steps. Each step is computed by a total deterministic macro tree transducer. In the first step we perform the necessary multiplications, representing numbers as trees with leaves the symbol $\star$. This representation allows us to perform multiplication by i without any additional computation; we simply introduce an i-ary branching node δ_i, and as its children we put i copies of our original number qua tree. We indicate that n copies of ℓ are required by replacing the leaf ℓ with the tree $\tau(\ell, x)$, where x is the representation

of n as described above.

Formally, we define $M_1 = \langle Q, P, \Sigma, \Lambda, \rightarrow, q, \delta \rangle$ to be a linear total deterministic 2-macro tree transducer with regular look-ahead, where

- $\langle P, \Sigma, \leftarrow \rangle$ is a total deterministic bottom-up tree automaton,
  - $P = \{p_i : i = f_\ell, \text{ some } \ell \in Lex\}$ is the set of states
  - Σ is the input alphabet, defined below
  - $\leftarrow$ is a finite set of productions consisting of

$$p_{f_\ell}(\ell) \leftarrow \ell$$
$$p_i(move(x)) \leftarrow move(p_i(x))$$
$$p_i(merge(x_1, x_2)) \leftarrow merge(p_i(x_1), p_j(x_2))$$

- $Q = \{q^{(1)}, q_i^{(2)} : i = f_\ell, \text{ some } \ell \in Lex\}$ is the ranked set of states,
- $\Sigma = \{merge^{(2)}, move^{(1)}, \ell^{(0)} : \ell \in Lex\}$ is the ranked input alphabet,
- $\Lambda = \{merge^{(2)}, move^{(1)}, \ell^{(0)}, \tau^{(2)}, \delta_i^{(i)}, \star^{(0)} : \ell \in Lex \text{ and } i = f_\ell\}$ is the output alphabet,
- $\rightarrow$ is a finite set of productions consisting of

$$q(x) \rightarrow q_i(x, \star), \ \langle p_i \rangle$$
$$q_i(\ell, y) \rightarrow \tau(\ell, y)$$
$$q_i(move(x), y) \rightarrow move(q_i(x, y))$$
$$q_i(merge(x_1, x_2), y) \rightarrow merge(q_i(x_1, y), q_j(x_2, \delta_i(y, \ldots, y))), \ \langle p_i, p_j \rangle$$

In the second step, we are engaged in transforming representations. Given the correct computation of the numerical relations from the first step, we next transform the tree-based representation of numbers into the list-based representation we need. Upon encountering a tree $\tau(\alpha, \beta)$, we know that α is some lexical item, and β is a tree representing the number of *select* branches that must dominate α. We traverse the tree β, putting one *select* branch over α for each $\star$ leaf we encounter.

Formally, we define $M_2 = \langle R, \Lambda, \Delta, \rightsquigarrow, r \rangle$ to be a linear total deterministic two-state 2-macro tree transducer, where

- $R = \{r^{(1)}, s^{(2)}\}$ is the ranked set of states,
- Λ, as defined above for M_1, is the ranked input alphabet
- $\Delta = \{merge^{(2)}, move^{(1)}, select^{(1)}, \ell^{(0)} : \ell \in Lex\}$ is the ranked output alphabet,
- $\rightsquigarrow$ is a finite set of productions, such that

$$r(merge(x_1, x_2)) \rightsquigarrow merge(r(x_1), r(x_2))$$
$$r(move(x)) \rightsquigarrow move(r(x))$$
$$r(\ell) \rightsquigarrow \ell$$
$$r(\tau(x_1, x_2)) \rightsquigarrow s(x_2, r(x_1))$$
$$s(\delta_i(x_1, \ldots, x_i), y) \rightsquigarrow s(x_1, s(\ldots, s(x_i, y)\ldots))$$
$$s(\star, y) \rightsquigarrow select(y)$$

We can compute h^{-1} by composing M_1 and M_2. We are now in a position to give a formal reconstruction of Chomsky's [32] claim that

> It has sometimes been suggested that [the internal merge approach to copying] should be eliminated in favor of [the external merge approach to copying]. [...] It [...] requires some other device to distinguish copies from unrelated occurrences. And it involves additional computation when a phrase that would otherwise be internally merged (moved) has to be generated independently before merger, then somehow identified as a copy [...]

We have already seen how to distinguish accidental from grammatically relevant copies. How much 'additional computation' is required to generate the right number of copies? We have shown that we need at most a total deterministic macro tree transducer working on the output of another.[16]

C–2 Pronunciation of Copies

The synchronization of minimalist grammars presented in the previous section allows us to implement the copy theory of movement, whereby a structural copy of an element is in each of its chain positions. However, the copy theory of movement makes more complex the mapping from derived structure, which is now populated with copies, to the pronounced form of a sentence (as well as its meaning), which still usually isn't. There are two problems here, which will be taken up in § C–2.1. First, that given the structures assigned to sentences, there is an overwhelming trend to pronounce only the structurally highest copy of a given expression, if multiple such exist. There are a number of proposals regarding how this is best formalized, and we shall see how our generating functions provide us with the

[16] Engelfriet and Vogler [54] show that any total deterministic macro tree transducer with regular look-ahead is equivalent to one without. Their construction requires $|P| - 1$ additional states (P is the set of look-ahead states), each of rank 3. They also show that total deterministic macro tree transducers are not closed under composition.

right distinctions to allow us to immediately implement these ideas in a natural and simple fashion. There are cases, however, which have been persuasively argued to involve the pronunciation of a lower copy instead of the default highest one. In § C–2.1.1 I will review proposals which make the choice of which copy to pronounce dependent on which copies of other chains are spellt-out, and will show how to implement these ideas in our current formalism. I then explore two ways to effect multiple spell-out of chains, based on ideas of Koopman [95] and of Nunes [131].

C–2.1 Spelling Out Only One Chain Link

There are a number of proposals as to how best to nullify at the PF-side the duplicative effect of the copy theory of movement—essentially, how to recover the pronunciation properties of the trace theory of movement. All of the proposals agree as to which copy should be pronounced (the structurally highest, although with the possibility of remnant movement a simple statement of this becomes challenging), however only Nunes [131] attempts to justify this in terms of more basic principles. Nunes envisions a theory of copying which copies not only the gross syntactic structure of the target phrase, but also the features contained in it. This has the consequence that higher copies have no more unchecked features than lower copies, and, when the movement is driven by unvalued features, may have fewer. Assuming that only a single member of a chain may be pronounced, Nunes derives the observation that it is the highest link by appeal to a more general dispreference for unchecked features at the interfaces.

In the context of (synchronous) minimalist grammars, a chain link is highest just in case it has no more licensee or selectee features to check. Thus, we recover the standard 'trace theory' of movement by incorporating the phonetic matrix of

the moved or merged phrase only in the case of move1 and merge1 and merge2 (which fact enables us to collapse the definitions of merge3 and merge4).[17]

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\ (t, \texttt{-x})\ \right] \beta}{(ts_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\begin{array}{c}\vdots\\(t', \chi)\\(t, \texttt{-x})\end{array}\right] \beta}{(\not{t}s_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c}\vdots\\(t', \chi)\end{array}\right] \beta}\text{move2}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{=x}\gamma, \alpha \quad \left\{\ (t_1, t_2, t_3) \cdot x, \beta\ \right\}}{(s_1, s_2, s_3 t_1 t_2 t_3) : \gamma, \alpha\beta}\text{merge1}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=x}\gamma, \alpha \quad \left\{\ (t_1, t_2, t_3) \cdot x, \beta\ \right\}}{(t_1 t_2 t_3 s_1, s_2, s_3) : \gamma, \alpha\beta}\text{merge2}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{=x}\gamma, \alpha \quad \left\{\begin{array}{c}\vdots\\(t'_1, t'_2, t'_3) \cdot \chi, \beta'\\(t_1, t_2, t_3) \cdot x, \beta\end{array}\right\}}{(s_1, s_2, s_3 \not{t}_1\not{t}_2\not{t}_3) : \gamma, \alpha \left[\begin{array}{c}\vdots\\(t'_1 t'_2 t'_3, \chi)\end{array}\right] \beta}\text{merge3}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=x}\gamma, \alpha \quad \left\{\begin{array}{c}\vdots\\(t'_1, t'_2, t'_3) \cdot \chi, \beta'\\(t_1, t_2, t_3) \cdot x, \beta\end{array}\right\}}{(\not{t}_1\not{t}_2\not{t}_3 s_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c}\vdots\\(t'_1 t'_2 t'_3, \chi)\end{array}\right] \beta}\text{merge4}$$

[17] affixLower and affixRaise1 are carried over from the above unchanged. affixRaise2, if we want to recover our old strategy, should be left out.

Note that this derivational formulation of the conditions on chain pronunciation correctly derives the distribution of overt copies within remnant moved constituents, a non-trivial (see [62]) feat.[18]

C–2.1.1 Allowing Chains to Interact

The pronunciation proposal articulated above assumes that, for each chain, the choice of which of its links to pronounce proceeds independently of the remainder of the chains in a structure. There have been a number of proposals that the choice of chain link to pronounce might be made dependent not only on syntactic properties, but also on the chain link's immediate phonological neighborhood (which can vary along with the choice of pronunciation for other chains). Recall the case of multiple wh-movement in Serbo-Croatian (repeated below). Examples 3.16 and 3.17 are illustrative of the general pattern of multiple wh-movement in Serbo-Croatian—wh-words must be fronted.

(3.16) Ko šta kupuje?
who what buys
'Who buys what?'

(3.17) ?**Ko kupuje šta*?

However, as the examples below show, this is not always the case. Examples 3.18 and 3.19 show exactly the opposite pattern of judgements from the above.

(3.18) ?**Šta šta uslovjava*?

[18]To see this, observe that 'remnant movement' describes cases in which a phrase α is at one point in the derivation a sister to a phrase β, both of which move independently later. In our system, α would be part of a non-trivial copy chain when attracted to β, and thus wouldn't leave any phonological residue, as only those cases of the generating functions affix phonological material to the head phrase which involve trivial copy chains.

(3.19) Šta uslovjava šta?
what conditions what

Interestingly, if an adverb can intervene between the wh-phrases, then the 'expected' pattern of judgements is restored.

(3.20) Šta neprestano šta uslovjava?
what constantly what conditions
'What constantly conditions what?'

(3.21) ?**Šta neprestano uslovjava šta*?

These examples have indicated to some the existence of a constraint militating against adjacent tokens of the same morpheme. The 'flipped' judgements of 3.18 and 3.19 are accounted for in terms of this constraint, violations of which can be avoided by pronouncing a lower copy of the wh-word (3.19). That this is a 'last resort' is appealed to in order to rule out 3.17 and 3.21.

In the context of (synchronous) minimalist grammars, we can mimic certain aspects of this kind of behaviour without enriching our representational scheme (much). We begin by defining a system that permits any chain link to be pronounced, so long as exactly one is. First, we re-define non-initial chains to include a boolean value the interpretation of which is that of recording whether the surface position of that particular non-initial chain has been reached (i.e. whether that non-initial chain has been pronounced ($\bullet$) or not ($\circ$)), so for ϕ_i a non-initial chain, $\phi_i \in (\Sigma^* \times \{\circ, \bullet\} \times \mathbb{F})^+$. Instead of re-presenting all of the cases of merge and move, I concentrate instead on the cases of move; the extensions needed to the others to implement this kind of phonological sensitivity should become obvious. Move2 is factored into two cases. In the first, the moved chain link is not pronounced, and so it doesn't matter whether or not some representative of its copy chain already has been ($\oplus \in \{\circ, \bullet\}$).

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \begin{bmatrix} \vdots \\ (t' \oplus \chi) \\ (t \oplus \texttt{-x}) \end{bmatrix} \beta}{(\not{t}s_1, s_2, s_3) : \gamma, \alpha \begin{bmatrix} \vdots \\ (t' \oplus \chi) \end{bmatrix} \beta}\text{move2a}$$

In the second case, the moved chain link is pronounced, and so no other link may have been pronounced before ($\circ$), and all remaining links in its copy chain are marked as having already been pronounced ($\bullet$).[19]

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \begin{bmatrix} \vdots \\ (t' \circ \chi) \\ (t \circ \texttt{-x}) \end{bmatrix} \beta}{(ts_1, s_2, s_3) : \gamma, \alpha \begin{bmatrix} \vdots \\ (t' \bullet \chi) \end{bmatrix} \beta}\text{move2b}$$

To ensure that at least one link has been pronounced, move1 is broken into cases as follows. In the first case, the moving chain has already been pronounced in a previous link ($\bullet$), and thus shouldn't be again.

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \begin{bmatrix} (t \bullet \texttt{-x}) \end{bmatrix} \beta}{(\not{t}s_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1a}$$

In the second case, the moving chain has not yet been pronounced ($\circ$). As this is the last chain link, and thus the last opportunity for pronunciation, it is pronounced.

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \begin{bmatrix} (t \circ \texttt{-x}) \end{bmatrix} \beta}{(ts_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1b}$$

[19]Strictly speaking, move is no longer a function, as the functions move2a and move2b overlap in their domains.

This extension to (synchronous) mgs allows for a simple definition of the candidate sets used in analyses of the data above—the members of a particular candidate set will have isomorphic derivation trees.[20] So as to avoid reifying candidate sets as grammatical objects, we will need to find a way to rule out 'bad' candidates without reference to the other candidates in their set. Assume for the sake of concreteness a single constraint which rules out adjacent tokens of identical morphemes. It is not possible in general to recover the 'as good as it gets' behaviour of a classical OT system, in which some alternative will be chosen even if there is no way to avoid sequences of identical morpheme tokens. In (obvious extensions to) the present system, derivations which position *any* identical tokens adjacent to one another will crash, and thus in the imagined case no derivation will converge. Although in principle possible to come up with such examples as would decide between these different behaviours, Serbo-Croatian marks case on dative objects, making it difficult to concoct examples in which the initial wh-cluster contains three adjacent mono-morphemic wh-words.

We might implement a ban on adjacent copies of a particular morpheme in a number of ways. First, we could do it post-syntactically, by means of a filter on generated sentences, prohibiting any which violate the ban. Another alternative is to crash a derivation as soon as it attempts to concatenate two strings together which have tokens of the same morpheme at the edges which are being concatenated together. In the general case, this amounts to applying a filter at each step in the derivation. However, many kinds of phonology-syntax interactions seem to be sensitive not just to adjacency, but also to surface constituency

[20]Recall that 'move' and 'merge', as the unions of the functions move1a, move1b, etc., are no longer functions, as these cases out of which they are composed overlap. Another way to state the canditate set given a derivation d is as $h^{-1}(h(d))$, where h maps the functions out of which move is composed to the symbol `v`, and the functions out of which merge is composed to the symbol `r`.

(or perhaps just to phonological phrasing). If this were a general property of phonology-syntax interactions, we would expect the adjacency of a deeply embedded wh-word with one structurally very distant not to force the pronunciation of a lower copy in the Serbo-Croatian cases.

To ensure that lower copies are pronounced only as a last resort, we need to be able to check whether or not higher positions would have been adjacent to pronounced identical morphemes. A simple way of doing this is to interpret the output of our syntax as a term in a more complicated structure than a free monoid. If we take a structure with directional inverses, we can introduce the term $x^r x$ in every chain position higher than the pronounced position. This term 'looks to the left' for an x; upon finding one, both the leftmost x and the x^r are cancelled out, leaving just the rightmost x behind. If there is no x on the left to be found, the term, we will decide, is uninterpretable by some necessary system on the PF branch of the derivation. Pregroups, as developed by Lambek [101, 102] (see also [23]), are elegant algebraic structures with right (and left) inverses, and will serve as the objects which, on the PF side, syntactic derivations are mapped into.[21] A free pregroup is given by a set of basic types **BasTyp** and a partial order over those types, which indicates the 'is-a' relation (e.g. a transitive verb is-a verb). For our purposes, we are interested in the rewriting system given by

[21] A pregroup is a structure $(G, \leq, \cdot, l, r, 1)$ such that $(G, \cdot, 1)$ is a monoid satisfying (for all $a, b, c \in G$)

$$\text{if } a \leq b, \text{ then } ca \leq cb \text{ and } ac \leq bc$$

and l, r are unary operations over G satisfying (for all $a \in G$)

$$a^l a \leq 1 \leq aa^l \text{ and } aa^r \leq 1 \leq a^r a$$

the following rules:

$$xaa^r y \rightarrow xy \qquad \text{(CON-r)}$$

$$xa^l ay \rightarrow xy \qquad \text{(CON-l)}$$

We take as our set **BasTyp** of basic types the string components of our lexical items.

We need modify only the move family of operations. In the case of a moving element which has already been pronounced, we insert the term $w^r w$.

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\; (t \bullet \texttt{-x}) \; \right] \beta}{(t^r t s_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1a}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \bullet \chi) \\ (t \bullet \texttt{-x}) \end{array} \right] \beta}{(t^r t s_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \bullet \chi) \end{array} \right] \beta}\text{move2a}$$

In the case where the moving chain has not yet been pronounced ($\circ$), we may either choose to pronounce it (move1b and move2b), or to do nothing (move2c).

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\; (t \circ \texttt{-x}) \; \right] \beta}{(t s_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1b}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \circ \chi) \\ (t \circ \texttt{-x}) \end{array} \right] \beta}{(t s_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \bullet \chi) \end{array} \right] \beta}\text{move2b}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \begin{bmatrix} \vdots \\ (t' \circ \chi) \\ (t \circ \texttt{-x}) \end{bmatrix} \beta}{(s_1, s_2, s_3) : \gamma, \alpha \begin{bmatrix} \vdots \\ (t' \circ \chi) \end{bmatrix} \beta}\text{move2c}$$

C–2.2 Spelling Out Multiple Chain Links

Koopman [95] introduces 'complexity filters,' which, stated over trees, require that pronounced material be within a certain distance from the root. She allows heads to be lexically associated with possibly different filters, which are satisfied in the course of a derivation just in case a subtree satisfying the filter is in the specifier of the head associated with it. A minor variant of this perspective views heads as able to require the spellout of their specifiers. In the present system, we allow for multiple spell-out by means of featural diacritics on selector ($\texttt{=}\hat{\texttt{x}}$) and licensor ($\texttt{+}\hat{\texttt{x}}$) features (which we can say 'require the spellout' of the link they license). For each case in the definitions of merge and move, we 'shadow' it with a parallel case involving a selector or licensor feature with the diacritic. The cases of move2 are given below.

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \begin{bmatrix} \vdots \\ (t', \chi) \\ (t, \texttt{-x}) \end{bmatrix} \beta}{(\not{t} s_1, s_2, s_3) : \gamma, \alpha \begin{bmatrix} \vdots \\ (t', \chi) \end{bmatrix} \beta}\text{move2}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+}\hat{\texttt{x}}\gamma, \alpha \begin{bmatrix} \vdots \\ (t', \chi) \\ (t, \texttt{-x}) \end{bmatrix} \beta}{(ts_1, s_2, s_3) : \gamma, \alpha \begin{bmatrix} \vdots \\ (t', \chi) \end{bmatrix} \beta}\text{move2r}$$

Nunes [131] examines a number of cases of what are commonly analyzed as the spell-out of multiple chain links. Having developed a theory which forbids more than one link being phonetically realized, Nunes postulates the existence of an operation which exceptionally allows a particular link to be realized without 'counting' as having been realized for the purposes of the rest of the chain. He dubs this operation 'reanalysis', and hypothesizes that it may have something to do with morphology, as in the cases involving multiple spell-out which he examines, there is an asymmetry between morphologically complex items, which cannot be multiply realized, and morphologically simple items, which can. To implement this, we keep the diacritics on the selector and licensor features that we introduced above, with the interpretation that the material in a diacritic licensed chain link must be spellt out. We also adopt the chain-interaction system using the markers $\bullet$ and $\circ$ to indicate whether a chain has been pronounced. However, we now reinterpret these booleans so as to indicate whether a chain has been pronounced *in a non-diacritic marked position*. As Nunes assumes that chains are pronounced in their highest non-diacritic marked link, we formulate our operations so as to enforce this.

The cases of move1 are as before, with the addition of move1r, which requires that the moving link's chain already have been spellt out, as the highest link will

be in a diacritic marked position.

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\ (t \bullet \texttt{-x}) \ \right] \beta}{(\not{t}s_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1a}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\ (t \circ \texttt{-x}) \ \right] \beta}{(ts_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1b}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+}\hat{\texttt{x}}\gamma, \alpha \left[\ (t \bullet \texttt{-x}) \ \right] \beta}{(ts_1, s_2, s_3) : \gamma, \alpha\beta}\text{move1r}$$

The cases of move2 are the same as before, with the addition of move2r. Move2r is exactly like move2a, but for the diacritic on the licensor feature. A moment's thought will convince us that this is as desired. As (non-final) reanalysis does not affect where a chain should be spellt out, it should apply regardless of whether the chain already has been or not.

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \oplus \chi) \\ (t \oplus \texttt{-x}) \end{array} \right] \beta}{(\not{t}s_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \oplus \chi) \end{array} \right] \beta}\text{move2a}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x}\gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \circ \chi) \\ (t \circ \texttt{-x}) \end{array} \right] \beta}{(ts_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \bullet \chi) \end{array} \right] \beta}\text{move2b}$$

$$\frac{(s_1, s_2, s_3) : \texttt{+x̂}\gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \oplus \chi) \\ (t \oplus \texttt{-x}) \end{array}\right] \beta}{(ts_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t' \oplus \chi) \end{array}\right] \beta}\text{move2r}$$

Merge1 and merge2 are unchanged, and the cases merge1r and merge2r are similarly uninteresting (differing from merge1 and merge2 just in the reanalysis diacritic on the licensor feature). Merge3 and merge4, along with their reanalysis variant merge3r, are shown below. There is no merge4r.

$$\frac{(s_1, s_2, s_3) :: \texttt{=x}\gamma, \alpha \left\{\begin{array}{c} \vdots \\ (t'_1, t'_2, t'_3) \cdot \chi, \beta' \\ (t_1, t_2, t_3) \cdot x, \beta \end{array}\right\}}{(s_1, s_2, s_3 t'_1 t'_2 t'_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t'_1 t'_2 t'_3 \circ \chi) \end{array}\right] \beta}\text{merge3}$$

$$\frac{(s_1, s_2, s_3) :: \texttt{=x̂}\gamma, \alpha \left\{\begin{array}{c} \vdots \\ (t'_1, t'_2, t'_3) \cdot \chi, \beta' \\ (t_1, t_2, t_3) \cdot x, \beta \end{array}\right\}}{(s_1, s_2, s_3 t_1 t_2 t_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t'_1 t'_2 t'_3 \circ \chi) \end{array}\right] \beta}\text{merge3r}$$

$$\frac{(s_1, s_2, s_3) : \texttt{=x}\gamma, \alpha \left\{\begin{array}{c} \vdots \\ (t'_1, t'_2, t'_3) \cdot \chi, \beta' \\ (t_1, t_2, t_3) \cdot x, \beta \end{array}\right\}}{(t'_1 t'_2 t'_3 s_1, s_2, s_3) : \gamma, \alpha \left[\begin{array}{c} \vdots \\ (t'_1 t'_2 t'_3 \bullet \chi) \end{array}\right] \beta}\text{merge4}$$

CHAPTER 4

Copying in Language

There have been a number of natural language constructions that immediately suggest to linguists a description in terms of copying, ranging from yes-no questions in Mandarin Chinese [81], to free relatives in languages as diverse as Malagasy, Bambara [49], and various dialects of Italian [70], to the more exotic X-or-no-X construction in the less exotic English. While linguists seem generally to be in favour of adopting some form of copy mechanism (though there are (justified) reservations), the particular nature of ours treats copying as of 'things that once were'—our copying is of objects at earlier stages of their derivation. As such, we allow for a certain kind of opaque relation between copies—our copies are not necessarily 'surface true'. In this section, we will exhibit and analyze data from the West African language Yoruba which our derivational approach to copying can give an elegant account of. Yoruba provides an ideal testing ground for theories of the mechanisms underlying copying, as, unlike the better known cases of copying in which just a single word (or morpheme) is copied, in Yoruba the copies are of syntactically complex expressions. In the appendices, we take a step back from any particular theory of grammar, and argue that natural languages, while still possibly polynomially parsable, are not of 'constant growth', and thus that the hypothesis of the mild context-sensitivity of natural language is false as stated.

1 Yoruba

One of the major languages of Nigeria, Yoruba (Niger-Congo:Benue-Kwa) is the mother tongue of nearly nineteen million nigerians.[1] Yoruba is an SVO language, which, with English, exhibits a number of head initial properties, such as verbs preceding their objects, complementizers preceding their sentences, and adpositions preceding their nouns. Yoruba also has nouns preceding adjectives, and possessors being preceded by the things possessed.

In this chapter, we will focus on a particular construction in Yoruba, the relativized predicate construction, as shown in 4.2 and 4.3.

(4.1) Jimọ ọ ra adiẹ
Jimọ HTS buy chicken
"Jimọ bought a chicken"

(4.2) Rira ti Jimọ ọ ra adiẹ
buying TI Jimọ HTS buy chicken
"the fact/way that Jimọ bought a chicken"

(4.3) Rira adiẹ ti Jimọ ọ ra adiẹ
buying chicken TI Jimọ HTS buy chicken
(same)

The obvious generalization (and the one made uniformly by those linguists working on this and related constructions in this and related languages) is that in

[1] Once classified as a Kwa language, Yoruba has been argued to be better situated in its own Yoruboid branch of the Benue-Congo phylum. There is some disagreement present in the literature. Yoruba orthography is similar to that of English, with the exception that the character 'p' represents the diphone /kp/ (the voiced version of which is written 'gb'). The letters 'ọ', 'ẹ', and 'ṣ' represent /ɔ/, /ɛ/ and /ʃ/ respectively. Although tone in Yoruba is phonemic (there are three level tones, written ́H, M, ̀L), I will not mark tone in the examples. 'Yoruba' is a broad cover term for a number of dialects, some more, some less mutually intelligible. The judgements reported herein, unless otherwise indicated, reflect those of my consultant, Damola Osinulu, a native speaker of the dialect spoken in Lagos (called 'standard Yoruba').

expressions like 4.2 and 4.3, the material to the left of *ti* is a copy of either the verb, or the verb phrase to the right of *ti*.

In Yoruba, as is the case with many of its neighbors, multiple verbs can be strung together in a single sentence (as in 4.4 and 4.5), to express a sequence of related events.[2]

(4.4) Jimọ ọ ra adiẹ se
Jimọ HTS buy chicken cook
"Jimọ bought the chicken to cook"

(4.5) Jimọ ọ ra adiẹ se jẹ
Jimọ HTS buy chicken cook eat
"Jimọ bought the chicken to cook and eat"

Such sentences give rise to a surprising diversity of relativized predicates. Examples 4.7 and 4.6 show the copied verb phrase and copied verb we might have expected given our previous examples. Example 4.8 on the other hand, appears to violate our intuitive generalization.

(4.6) Rira ti Jimọ ọ ra adiẹ se
buying TI Jimọ HTS buy chicken cook
"the fact/way that Jimọ bought the chicken to cook"

(4.7) Rira adiẹ se ti Jimọ ọ ra adiẹ se
buying chicken cook TI Jimọ HTS buy chicken cook

(4.8) Rira se ti Jimọ ọ ra adiẹ se
buying cook TI Jimọ HTS buy chicken cook

Clearly, the part to the left of *ti* in example 4.8 is a copy of *something*, but what? Our copying mechanism as developed in chapter 3 allows us to elegantly describe

[2]Not just any sequence of verbs makes for an acceptable sentence. The conditions under which this obtains are complex, and poorly understood. I will not attempt to remedy this situation here.

the relation between the two 'copies' in 4.8 as one of copying, rendered opaque by further derivation. We will see that the relation between copies in 4.8 is as simple and natural to describe using our derivational copy mechanism as are those in 4.6 and 4.7; there is no additional complexity in example 4.8, all three examples are derived in exactly the same way. To see this, however, we need first to develop a theory of the grammar of Yoruba, to which task we now turn.

1.1 Simple Sentences

Intransitive sentences in Yoruba have subject-verb order, as in 4.9. Between the full DP subject and the verb there appears a high tone syllable (HTS) that surfaces as a copy of the final, low-toned, coda of the subject, or as a high tone on the final, non-low-toned, coda of the subject, which we identify with the non-future tense (following [17]), and represent uniformly as a copy of the final syllable.[3]

(4.9) Akin in ṣubu
Akin HTS fall
"Akin fell"

In a transitive sentence, the object follows the verb.

(4.10) Jimọ ọ ti Akin
Jimọ HTS push Akin
"Jimọ pushed Akin"

[3]There is disagreement as to the categorial identity as well as the semantic contribution of the HTS. The HTS does not occur with morphemes such as the future marker *yóò* and the sentential negator *kò*, or when the subject is a pronoun, but may occur with the word *máa*, which is similar in meaning to *yóò*, as well as various other aspect markers (see e.g. [135]). It also occurs in infinitival clauses (see [6] for a historical perspective). We will largely ignore these subtleties here, and will populate our fragment with sentences lacking tense/aspect markers other than the HTS.

Both of these sentences can be accomodated if we assign types to lexical items as in figure 4.1.[4]

Jimọ::d	V::=v =d t	ṣubu::v
Akin::d		ti::=d v

Figure 4.1: Yoruba (I)

Although there is in general no passive operation in Yoruba, a class of verbs appears in both transitive and intransitive frames (the causative-inchoative alternation). Bode [17] suggests identifying this class with the class of causative predicates the logical subjects of which are not necessarily agentive.

(4.11) Jimọ ọ fo igo
Jimọ HTS break bottle
"Jimọ broke the bottle"

(4.12) Igo o fo
bottle HTS break
"The bottle broke"

The existence of such an alternation (limited though it may be) provides us with a language internal motivation for postulating that, as in our grammar for English, Yoruba DPs are merged (d) in positions which may be different from where they ultimately are pronounced (-k).[5] As now Yoruba DPs are non-trivial chains, it seems natural to carry over to Yoruba the semantics developed for English in chapter 2. Accordingly, we need to assign another licensee feature (-q) to DPs, so as to be able to interpret them without type raising. To capture the fact that

[4]We must also assume that the HTS (here represented as a high-toned abstract vowel 'V', is interpreted appropriately by the phonological component as a copy of a preceeding low toned coda, or as a simple high tone on a non-low coda.

[5]Another construction which might lead to similar conclusions is the 'object raising' OV complement clause construction examined in [138].

only some verbs participate in the causative-inchoative alternation, we assign these verbs the category `inch`, and take their semantic denotation to be that of their transitive version. The causative alternant is then derived with the lexical item ϵ::`=>inch V`, which is interpreted as the identity function. We derive the inchoative alternant by means of the lexical item ϵ::`=>inch v`, which we interpret as we did the English passive.[6]

$$[\![\epsilon :: \texttt{=>inch v}]\!] = \textbf{some}$$

Again, we assume that Yoruba DPs all have the same type, irrespective of whether they are syntactic subjects or objects. This entails, among other things, that even those verbs which are not of category `inch` must move around their objects so as to precede them in the sentence. Our new lexicon is as in figure 4.2.

Jimọ::`d -k -q`	V::`=perf +k +q s`	ṣubu::`=d v`
Akin::`d -k -q`	ϵ::`=>v perf`	ti::`=d V`
igo::`d -k -q`	ϵ::`=>V +k =d +q v`	fo::`=d inch`
	ϵ::`=>inch v`	
	ϵ::`=>inch V`	

Figure 4.2: Yoruba (II)

[6]This treatment of the alternation is almost certainly incorrect, as it assumes that we conceptualize all breaking as having a cause (and thus disallows spontaneous breakage). A natural approach would allow variables to range over events, and to take the inchoative meaning as the basic one. Then the causative lexical item could be interpreted as follows

$$[\![\epsilon :: \texttt{=>inch V}]\!] = \lambda V.\lambda x.\lambda e.\exists e'.\textbf{cause}(e, e') \wedge V(e') \wedge \textbf{agent}(e) = x$$

Our semantics supports such an extension to a two-sorted ontology, and indeed to any finite number of sorts. Assignment functions become pairs (or more generally, finite tuples) of infinite sequences over objects of each sort (which we can code up as a single assignment function, as in appendix B–2). Events might prove useful as well in capturing more accurately the entailments of serial verb sentences (see footnote 10).

We interpret *Jimọ* (and, more generally, all individual denoting expressions) as the generalized quantifier **Jimọ**, where, for **j** the constant function from assignments to Jimọ,

$$\textbf{Jimọ} = \lambda P.P(\textbf{j})$$

1.2 Serial Verbs

If we take sentences 4.9 and 4.10, we can 'put them together' as in 4.13.

(4.13) Jimọ ọ ti Akin ṣubu
Jimọ HTS push Akin fall
"Jimọ pushed Akin down"

In sentence 4.13, the DP *Akin* is 'shared' among the two verbs, playing a dual role as the logical object of *ti*, and the logical subject of *ṣubu*. This has been analyzed in two basic ways. First, Baker [8] and others (e.g. [107, 110]) have argued that there is a single syntactic occurance of the shared DP, and that the predicates *ti* and *ṣubu* form a single 'complex' predicate for the purposes of licensing this DP. Collins [42, 43] and others (e.g. [3, 105, 164]) have argued that each predicate selects its own argument, and that the selected arguments are later 'linked' syntactically.[7]

As we have already developed, for the analysis of control in English, a mechanism which allows for DPs to be shared by predicates, our minimalist bent

[7] Although this basic analytical distinction seems clear enough ('are there two syntactic argument positions, or just one?'), it is actually not straightforward to discern potential substantive differences from betwixt the non-essential differences in notation. (What does it mean for an obligatorially bound and never pronounced semantic argument to be realized syntactically?) It seems to me that the differences between analyses in either of these two camps all come down to which mechanisms are being co-opted by the analyst (i.e. which phenomena serial verbs are being attempted to be unified with). Thus a tighter typology of serial verb analyses would avoid the apparently artifical distinction articulated above about argument positions, and instead indicate with which other phenomena serial verbs are taken to have something in common.

pushes us toward this latter analytical option.[8] Accordingly, we need to combine the expression below

$$(\epsilon,\ \text{ṣubu},\ \epsilon) : \texttt{v}, (\text{Akin},\ \texttt{*d -k})$$

with the verb *ti*. We want to combine the expression above with *ti*, and then to control move *Akin* to satisfy the selectional requirements of the verb *ti*. We thus assign the type `=v =d V` to *ti*. However, now we have perpetuated a widespread lexical ambiguity, as two types (`=d V` and `=v =d V`) are assigned to every transitive verb (at least, to every transitive verb participating in a resultative serial verb construction, which is quite a few of them). Our existing grammatical operations and type system are powerful enough to allow us to describe this situation in the object language of our theory, simplifying our lexicon in the process. We assign to transitive verbs the type `tv`, and with the empty lexical items ϵ::`=>tv =d V` and ϵ::`=>tv =v =d V` we state that each transitive verb may optionally select a vP complement. Semantically, these lexical items are interpreted as sketched below.[9]

[8]There are two basic methodological options. The first, of which the minimalist program is the linguistic embodiement, urges against the reckless proliferation of mechanisms, encouraging reuse of already developed analytical tools whenever possible. This makes unification of diverse phenomena a methodological imperative. The other option, which has not been as clearly articulated, but which seems to be behind various criticisms of the principles and parameters approach to grammar, is to describe each construction as completely as possible, availing oneself of whatever and as many mechanisms as make for simple and elegant descriptions.

Practitioners of this first option sometimes motivate it on the basis of learning, claiming that if the analytical resources are limited, the learner's options at each point in time will be small in number, thereby presenting it with a more manageable situation. Practitioners of the second option sometimes motivate it on the basis of learning, claiming that allowing the learner more analytical options permits it to state simpler, more surface true generalizations, thereby presenting it with a more manageable situation. Both positions seem reasonable, as far as they go. To advance discussion it seems necessary to propose specific learning algorithms.

[9]The function **and** is intended to be the natural extension of the sentential connective of the same name to higher types. It has type $[\alpha \to \beta] \to [\alpha \to \beta] \to \alpha \to \beta$, where β 'ends in T' (a type α ends in T just in case either $\alpha = T$ or $\alpha = \beta \to \gamma$ and γ ends in T). We define **and** recursively as follows.

if X, Y are of type $\alpha \to T$ and a is of type α, then

$$(\mathbf{and}(X)(Y))(a) = \mathbf{true} \text{ iff } X(a) = Y(a) = \mathbf{true}$$

10

$$[\![\epsilon :: \texttt{=>tv =d V}]\!] = \texttt{id}$$

$$[\![\epsilon :: \texttt{=>tv =v =d V}]\!] = \lambda R.\lambda\Phi.\lambda x.\lambda y.\mathbf{and}(R(x)(y))(\Phi)$$

We derive sentence 4.13 in the following way, with a derived tree structure as in figure 4.3.

1. cmerge(ṣubu::=d v, Akin::*d -k -q)

$$(\epsilon, \text{ṣubu}, \epsilon) : \texttt{v},\ (\text{Akin}, \texttt{*d -k -q})$$

$$\mathbf{fall}(\mathbf{x_0}),\ \boxed{\langle \mathbf{x_0}, \texttt{G}(\mathbf{Akin})(\lambda_0)\rangle}$$

2. merge(ϵ::=>tv =v =d V, ti::tv)

$$(\epsilon, \text{ti}, \epsilon) : \texttt{=v =d V}$$

$$\lambda\Phi.\lambda x.\lambda y.\mathbf{and}(\mathbf{push}(x)(y))(\Phi)))$$

3. merge(2, 1)

$$(\epsilon, \text{ti}, \text{ṣubu}) : \texttt{=d V},\ (\text{Akin}, \texttt{*d -k -q})$$

$$\lambda x.\lambda y.\mathbf{and}(\mathbf{push}(x)(y))(\mathbf{fall}(\mathbf{x_0})),\ \boxed{\langle \mathbf{x_0}, \texttt{G}(\mathbf{Akin})(\lambda_0)\rangle}$$

4. cmove1(3)

$$(\epsilon, \text{ti}, \text{ṣubu}) : \texttt{V},\ (\text{Akin}, \texttt{-k -q})$$

$$\lambda y.\mathbf{and}(\mathbf{push}(\mathbf{x_0})(y))(\mathbf{fall}(\mathbf{x_0})),\ \boxed{\texttt{G}(\mathbf{Akin})(\lambda_0)}$$

otherwise for X, Y of type $\alpha \to \beta$ (where β ends in T) and a of type α,

$$(\mathbf{and}(X)(Y))(a) = \mathbf{and}(X(a))(Y(a))$$

[10]This treats serial verb sentences as semantic conjunctions of their component verb phrases. This is not quite right, as in sentences like 4.13 Akin's falling is asserted to be a direct result of his having been pushed by Jimo. One way to capture this aspect of the meaning of such sentences is to enrich our semantic ontology to include events and a relation of direct causation.

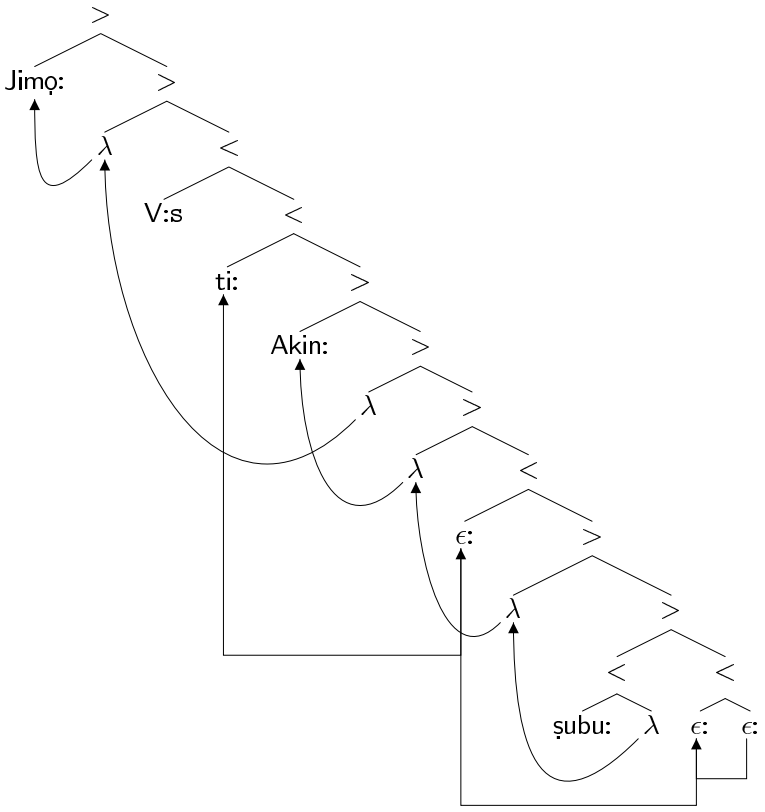


Figure 4.3: The derived structure for sentence 4.13

5. merge(ϵ::=>V +k =d +q v, 4)

(ϵ, ti, ṣubu) : +k =d +q v, (Akin, -k -q)

$\lambda y.\mathbf{and}(\mathbf{push}(\mathbf{x_0})(y))(\mathbf{fall}(\mathbf{x_0}))$, $\boxed{\mathtt{G}(\mathbf{Akin})(\lambda_0)}$

6. move(5)

(ϵ, ti, ṣubu) : =d +q v, (Akin, -q)

$\lambda y.\mathbf{and}(\mathbf{push}(\mathbf{x_0})(y))(\mathbf{fall}(\mathbf{x_0}))$, $\boxed{\mathtt{G}(\mathbf{Akin})(\lambda_0)}$

7. merge(6, Jimọ::*d -k -q)

(ϵ, ti, ṣubu) : +q v, (Akin, -q), (Jimọ, -k -q)

$$\mathbf{and}(\mathbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\mathbf{fall}(\mathbf{x_0})),\ \boxed{\texttt{G}(\mathbf{Akin})(\lambda_0),\ \texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

8. move(7)

$$(\text{Akin, ti, ṣubu}) : \texttt{v},\ (\text{Jimọ, -k -q})$$

$$\mathbf{Akin}(\lambda_0(\mathbf{and}(\mathbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\mathbf{fall}(\mathbf{x_0})))),\ \boxed{\texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

9. merge(ϵ::=>v perf, 8)

$$(\epsilon\text{, ti, Akin ṣubu}) : \texttt{perf},\ (\text{Jimọ, -k -q})$$

$$\mathbf{Akin}(\lambda_0(\mathbf{and}(\mathbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\mathbf{fall}(\mathbf{x_0})))),\ \boxed{\texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

10. merge(V::=perf +k +q s, 9)

$$(\epsilon\text{, V, ti Akin ṣubu}) : \texttt{+k +q s},\ (\text{Jimọ, -k -q})$$

$$\mathbf{Akin}(\lambda_0(\mathbf{and}(\mathbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\mathbf{fall}(\mathbf{x_0})))),\ \boxed{\texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

11. move(10)

$$(\epsilon\text{, V, ti Akin ṣubu}) : \texttt{+q s},\ (\text{Jimọ, -q})$$

$$\mathbf{Akin}(\lambda_0(\mathbf{and}(\mathbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\mathbf{fall}(\mathbf{x_0})))),\ \boxed{\texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

12. move(11)

$$(\text{Jimọ, V, ti Akin ṣubu}) : \texttt{s}$$

$$\mathbf{Jimọ}(\lambda_1(\mathbf{Akin}(\lambda_0(\mathbf{and}(\mathbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\mathbf{fall}(\mathbf{x_0})))))))$$

The meaning representation in 12 can be rewritten more perspicuously as[11]

$$\begin{aligned}
&\textbf{Jimọ}(\lambda_1(\textbf{Akin}(\lambda_0(\textbf{and}(\textbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\textbf{fall}(\mathbf{x_0})))))) \\
&\quad = (\lambda_1(\textbf{Akin}(\lambda_0(\textbf{and}(\textbf{push}(\mathbf{x_0})(\mathbf{x_1}))(\textbf{fall}(\mathbf{x_0})))))(\mathbf{j}) \\
&\quad = \textbf{Akin}(\lambda_0(\textbf{and}(\textbf{push}(\mathbf{x_0})(\mathbf{j}))(\textbf{fall}(\mathbf{x_0})))) \\
&\quad = (\lambda_0(\textbf{and}(\textbf{push}(\mathbf{x_0})(\mathbf{j}))(\textbf{fall}(\mathbf{x_0}))))(\mathbf{a}) \\
&\quad = \textbf{and}(\textbf{push}(\mathbf{a})(\mathbf{j}))(\textbf{fall}(\mathbf{a})) \\
&\quad = \textbf{push}(\mathbf{a})(\mathbf{j}) \cap \textbf{fall}(\mathbf{a})
\end{aligned}$$

Although the second verb in the examples we have looked at thus far has been intransitive, this is not a necessary property of serial verb constructions in Yoruba, as examples like 4.14 show.

(4.14) Jimọ ọ jaa Akin ti
Jimọ HTS fight Akin push
"Jimọ fought with Akin and pushed him"

Sentences like 4.14 above pose a problem for us, as the two transitive verbs *jaa* and *ti* share both subject and object. We have mediated the object sharing in the case of 'resultative' serial verb constructions as in 4.13 via control movement. However, our argument introducing heads require the object to check its case before the subject is introduced, which prohibits both subject and object sharing to be mediated by control. Even with a different approach to argument introduction, given our principle of immediacy, we cannot have both subject and object in storage for the purposes of control simultaneously. Therefore, we must mediate

[11]To allay a potential source of confusion, I am not using β-conversion here. I cannot, as our meaning representations are not lambda-terms. Instead, I am asserting the equivalence of each of the objects denoted by their respective meaning terms. Sometimes, as with **and**, this follows directly from the definition of the function. In other cases, such as when an individual is substituted for a variable 'bound' by a lambda, this is a less direct consequence of the definition of the λ_i function. It is nonetheless straightforward; an example is given in § 2.4 of chapter 2.

at least one sharing relationship by some means other than our familiar control movement operation. We add the lexical item ϵ::=>tv =V =d V, and interpret it as shown below.

$$[\![\epsilon :: \texttt{=>tv =V =d V}]\!] = \lambda R.\lambda P.\lambda x.\mathbf{and}(R(x))(P)$$

We derive sentence 4.14 as follows, with the derived structure as shown in figure 4.4.

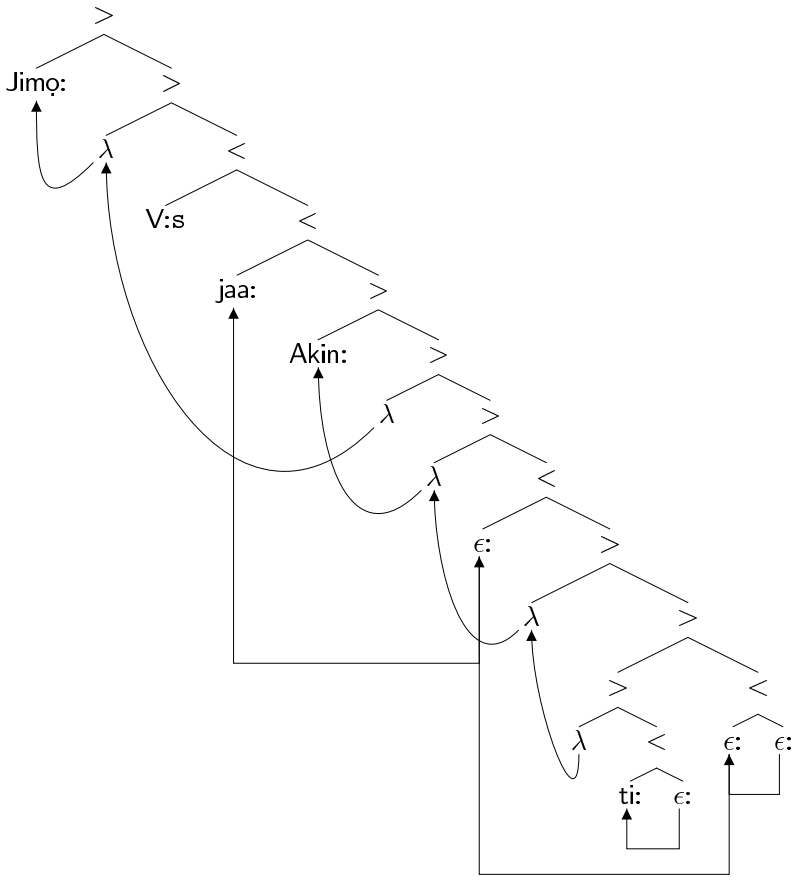


Figure 4.4: The derived structure for sentence 4.14

1. merge(ϵ::=>tv =d V, ti::tv)

$$(\epsilon,\ \mathsf{ti},\ \epsilon) : \texttt{=d V}$$

push

2. cmerge(1, Akin::*d -k -q)

$$(\epsilon,\ \mathsf{ti},\ \epsilon) : \mathtt{V},\ (\mathsf{Akin},\ \mathtt{*d\ -k\ -q})$$

$$\mathbf{push}(\mathbf{x_0}),\ \boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{Akin})(\lambda_0)\rangle}$$

3. merge(ϵ::=>tv =V =d V, jaa::tv)

$$(\epsilon,\ \mathsf{jaa},\ \epsilon) : \mathtt{=V\ =d\ V}$$

$$\lambda P.\lambda x.\mathbf{and}(\mathbf{fight}(x))(P)$$

4. merge(3, 2)

$$(\epsilon,\ \mathsf{jaa},\ \mathsf{ti}) : \mathtt{=d\ V},\ (\mathsf{Akin},\ \mathtt{*d\ -k\ -q})$$

$$\lambda x.\mathbf{and}(\mathbf{fight}(x))(\mathbf{push}(\mathbf{x_0})),\ \boxed{\langle \mathbf{x_0}, \mathtt{G}(\mathbf{Akin})(\lambda_0)\rangle}$$

5. cmove1(4)

$$(\epsilon,\ \mathsf{jaa},\ \mathsf{ti}) : \mathtt{V},\ (\mathsf{Akin},\ \mathtt{-k\ -q})$$

$$\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})),\ \boxed{\mathtt{G}(\mathbf{Akin})(\lambda_0)}$$

6. merge(ϵ::=>V +k =d +q v, 5)

$$(\epsilon,\ \mathsf{jaa},\ \mathsf{ti}) : \mathtt{+k\ =d\ +q\ v},\ (\mathsf{Akin},\ \mathtt{-k\ -q})$$

$$\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})),\ \boxed{\mathtt{G}(\mathbf{Akin})(\lambda_0)}$$

7. move(6)

$$(\epsilon,\ \mathsf{jaa},\ \mathsf{ti}) : \mathtt{=d\ +q\ v},\ (\mathsf{Akin},\ \mathtt{-q})$$

$$\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})),\ \boxed{\mathtt{G}(\mathbf{Akin})(\lambda_0)}$$

8. merge(7, Jimọ::*d -k -q)

(ϵ, jaa, ti) : =d +q v, (Akin, -q), (Jimọ, -k -q)

$(\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})))(\mathbf{x_1})$, $\boxed{\mathtt{G}(\mathbf{Akin})(\lambda_0),\ \mathtt{G}(\mathbf{Jimọ})(\lambda_1)}$

9. move(8)

(Akin, jaa, ti) : v, (Jimọ, -k -q)

$\mathbf{Akin}(\lambda_0((\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})))(\mathbf{x_1})))$, $\boxed{\mathtt{G}(\mathbf{Jimọ})(\lambda_1)}$

10. merge(ϵ::=>v perf, 9)

(ϵ, jaa, Akin ti) : perf, (Jimọ, -k -q)

$\mathbf{Akin}(\lambda_0((\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})))(\mathbf{x_1})))$, $\boxed{\mathtt{G}(\mathbf{Jimọ})(\lambda_1)}$

11. merge(V::=perf +k +q s, 10)

(ϵ, V, jaa Akin ti) : +k +q s, (Jimọ, -k -q)

$\mathbf{Akin}(\lambda_0((\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})))(\mathbf{x_1})))$, $\boxed{\mathtt{G}(\mathbf{Jimọ})(\lambda_1)}$

12. move(11)

(ϵ, V, jaa Akin ti) : +q s, (Jimọ, -q)

$\mathbf{Akin}(\lambda_0((\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})))(\mathbf{x_1})))$, $\boxed{\mathtt{G}(\mathbf{Jimọ})(\lambda_1)}$

13. move(12)

(Jimọ, V, jaa Akin ti) : s

$\mathbf{Jimọ}(\lambda_1(\mathbf{Akin}(\lambda_0((\mathbf{and}(\mathbf{fight}(\mathbf{x_0}))(\mathbf{push}(\mathbf{x_0})))(\mathbf{x_1})))))$

Again, a more perspicuous representation of the denotation in 13 can be derived as follows.

$$\begin{aligned}
&\textbf{Jimọ}(\lambda_1(\textbf{Akin}(\lambda_0((\textbf{and}(\textbf{fight}(\mathbf{x_0}))(\textbf{push}(\mathbf{x_0})))(\mathbf{x_1})))))\\
&\quad = (\lambda_1(\textbf{Akin}(\lambda_0((\textbf{and}(\textbf{fight}(\mathbf{x_0}))(\textbf{push}(\mathbf{x_0})))(\mathbf{x_1})))))(\mathbf{j})\\
&\quad = \textbf{Akin}(\lambda_0((\textbf{and}(\textbf{fight}(\mathbf{x_0}))(\textbf{push}(\mathbf{x_0})))(\mathbf{j})))\\
&\quad = (\lambda_0((\textbf{and}(\textbf{fight}(\mathbf{x_0}))(\textbf{push}(\mathbf{x_0})))(\mathbf{j})))(\mathbf{a})\\
&\quad = (\textbf{and}(\textbf{fight}(\mathbf{a}))(\textbf{push}(\mathbf{a})))(\mathbf{j})\\
&\quad = \textbf{and}(\textbf{fight}(\mathbf{a})(\mathbf{j}))(\textbf{push}(\mathbf{a})(\mathbf{j}))\\
&\quad = \textbf{fight}(\mathbf{a})(\mathbf{j}) \cap \textbf{push}(\mathbf{a})(\mathbf{j})
\end{aligned}$$

The number of transitive verbs sharing an object in a serial verb construction does not seem to have a principled upper bound (4.15). Furthermore, intransitives and transitives may be mixed without apparent bound as well (4.16).

(4.15) Jimọ ọ wa aṣọ ri ji wọ
Jimọ HTS look for clothes find steal wear
"Jimọ looked for clothes, found some, and stole them to wear"

(4.16) Jimọ ọ jaa Akin ti ṣubu
Jimọ HTS fight Akin push fall
"Jimọ fought with Akin and pushed him down"

These facts are predicted by our current lexical type assignments. Consider the derivation of sentence 4.13 (*Jimọ ọ ti Akin ṣubu*) above. At step 4, if, instead of applying cmove1, we apply cmove2, we obtain the expression below.

$$(\epsilon,\ \mathsf{ti},\ \mathsf{ṣubu}) : \mathtt{V},\ (\mathsf{Akin},\ \mathtt{*d\ -k\ -q})$$

As this expression has the same syntactic type as the expression in step 2 of the derivation of sentence 4.14 (*Jimọ ọ jaa Akin ti*), we can substitute it for the

expression currently there in step 2, and derive thereby sentence 4.16 above, with the meaning below.

$$\textbf{fight}(\mathbf{a})(\mathbf{j}) \cap \textbf{push}(\mathbf{a})(\mathbf{j}) \cap \textbf{fall}(\mathbf{a})$$

1.3 Relative Clauses

Relative clauses in Yoruba are formed by placing a particle *ti* in front of a clause containing a gap (in the case of relativization of an object) or a resumptive pronoun (in the case of relativization of a subject or a possessor). Example 4.17 illustrates relativization of an object, and 4.18 of a subject.

(4.17) Adiẹ ti Jimọ ọ ra
chicken TI Jimọ HTS buy
"The chicken that Jimọ bought"

(4.18) Eni ti o ra adiẹ
person TI 3S buy chicken
"The person who bought a chicken"

We will concentrate on the general shape of the relative clause construction, and ignore the difference between subject and object extraction with respect to the occurance of the resumptive pronoun.[12] We take nouns to have the type `n`, and to then be coerced into DPs by means of the lexical item ϵ::`=n *d -k -q`, or to be readied for use in a relative clause by the lexical item ϵ::`=n *d -k -q -f`.[13] To the particle *ti* we assign the type `=s +f n`, and interpret it as the identity function.

[12]Subject pronouns in Yoruba (resumptive or otherwise) behave differently from full DPs. In addition to varying their shape when adjacent to the sentential negator *kò*, they do not co-occur with the HTS.

[13]Yoruba 'bare' nouns are translated as English definite or indefinite nouns depending on the context. In other words, a bare noun in Yoruba is ambiguous as to whether it is definite or indefinite. The Yoruba word *náà* is sometimes translated as *the*, but plays a role in the language much different from the simple definite in English (see e.g. [149] for discussion).

The lexical item ϵ::=n *d -k -q -f we interpret as the function **and**. We herewith adopt a version of the raising analysis of relative clauses, recently resurrected by Kayne [90], and championed by [12, 14, 51].

With this in hand, we derive the noun phrase in 4.17 in the following manner.

1. merge(ϵ::=>tv =d V, ra::tv)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \mathtt{=d\ V}$$

$$\mathbf{buy}$$

2. merge(ϵ::=n *d -k -q -f, adiẹ::n)

$$(\epsilon,\ \epsilon,\ \mathsf{adiẹ}) : \mathtt{*d\ -k\ -q\ -f}$$

$$\mathbf{and}(\mathbf{chicken})$$

3. merge(1, 2)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \mathtt{V},\ (\mathsf{adiẹ},\ \mathtt{-k\ -q\ -f})$$

$$\mathbf{buy}(\mathbf{x_0}),\ \boxed{\mathtt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0)}$$

4. merge(ϵ::=>V +k =d +q v, 3)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \mathtt{+k\ =d\ +q\ v},\ (\mathsf{adiẹ},\ \mathtt{-k\ -q\ -f})$$

$$\mathbf{buy}(\mathbf{x_0}),\ \boxed{\mathtt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0)}$$

5. move(4)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \mathtt{=d\ +q\ v},\ (\mathsf{adiẹ},\ \mathtt{-q\ -f})$$

$$\mathbf{buy}(\mathbf{x_0}),\ \boxed{\mathtt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0)}$$

6. merge(5, Jimọ::*d -k -q)

$$(\epsilon,\ \text{ra},\ \epsilon) : \texttt{+q v},\ (\text{adiẹ},\ \texttt{-q -f}),\ (\text{Jimọ},\ \texttt{-k -q})$$

$$\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0),\ \texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

7. move(6)

$$(\epsilon,\ \text{ra},\ \epsilon) : \texttt{v},\ (\text{adiẹ},\ \texttt{-f}),\ (\text{Jimọ},\ \texttt{-k -q})$$

$$\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0),\ \texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

8. merge(ϵ::=>v perf, 7)

$$(\epsilon,\ \text{ra},\ \epsilon) : \texttt{perf},\ (\text{adiẹ},\ \texttt{-f}),\ (\text{Jimọ},\ \texttt{-k -q})$$

$$\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0),\ \texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

9. merge(V::=perf +k +q s, 8)

$$(\epsilon,\ \text{V},\ \text{ra}) : \texttt{+k +q s},\ (\text{adiẹ},\ \texttt{-f}),\ (\text{Jimọ},\ \texttt{-k -q})$$

$$\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0),\ \texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

10. move(9)

$$(\epsilon,\ \text{V},\ \text{ra}) : \texttt{+q s},\ (\text{adiẹ},\ \texttt{-f}),\ (\text{Jimọ},\ \texttt{-q})$$

$$\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}),\ \boxed{\texttt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0),\ \texttt{G}(\mathbf{Jimọ})(\lambda_1)}$$

11. move(10)

$$(\text{Jimọ},\ \text{V},\ \text{ra}) : \texttt{s},\ (\text{adiẹ},\ \texttt{-f})$$

$$\mathbf{Jimọ}(\lambda_1(\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}))),\ \boxed{\texttt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0)}$$

12. merge(ti::=s +f n, 11)

$$(\epsilon,\ \text{ti},\ \text{Jimọ V ra}) : \texttt{+f n},\ (\text{adiẹ},\ \texttt{-f})$$

$$\mathbf{Jimọ}(\lambda_1(\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}))),\ \boxed{\texttt{G}(\mathbf{and}(\mathbf{chicken}))(\lambda_0)}$$

13. move(12)

$$\text{(adiẹ, ti, Jimọ V ra)} : \mathtt{n}$$

$$\mathbf{and}(\mathbf{chicken})(\lambda_0(\mathbf{Jimọ}(\lambda_1(\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1})))))$$

Again, the representation in 13 has the more transparent form below. For $f : [G \to E]$,

$$\begin{aligned}
&(\mathbf{and}(\mathbf{chicken})(\lambda_0(\mathbf{Jimọ}(\lambda_1(\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}))))))(f) \\
&\quad = \mathbf{and}(\mathbf{chicken}(f))(\lambda_0(\mathbf{Jimọ}(\lambda_1(\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1})))))(f)) \\
&\quad = \mathbf{chicken}(f) \cap \lambda_0(\mathbf{Jimọ}(\lambda_1(\mathbf{buy}(\mathbf{x_0})(\mathbf{x_1}))))(f) \\
&\quad = \mathbf{chicken}(f) \cap \mathbf{Jimọ}(\lambda_1(\mathbf{buy}(f)(\mathbf{x_1}))) \\
&\quad = \mathbf{chicken}(f) \cap (\lambda_1(\mathbf{buy}(f)(\mathbf{x_1})))(\mathbf{j}) \\
&\quad = \mathbf{chicken}(f) \cap \mathbf{buy}(f)(\mathbf{j})
\end{aligned}$$

Note that since a noun-plus-relative clause has the same type as a bare noun (n), we can stack relative clauses ad infinitum, as exemplified for the case of two in 4.19.

(4.19) Adiẹ ti Jimọ ọ ra ti o tobi
chicken TI Jimọ HTS buy TI 3S be big
"The big chicken that Jimọ bought"

We thus have a syntactic analysis of stacked relative clauses similar to that of Bianchi [14], in which later relative clauses scope over earlier ones (as opposed to a 'flat' syntactic structure), which can be schematzed in the following manner.

$$[\ldots[[\text{N RC}_1]\ \text{RC}_2]\ldots\text{RC}_n]$$

The nested syntactic structure we assign to stacked relative clauses notwithstanding, our simple conjunctive semantics has the desired effect of yielding a semantically 'flat' representation. Our semantics yields the following interpretation for

example 4.19 above (treating *tobi* as a simple intransitive verb).

$$\textbf{chicken}(f) \cap \textbf{buy}(f)(\textbf{j}) \cap \textbf{big}(f)$$

1.3.1 The Relativized Predicate

Verb phrases may appear in constructions very much like the relative clauses discussed above. When they do, the resulting phrase has the distribution of a(n abstract) noun. Of particular interest to us is the relation between the verb (phrase) which acts as head of the relative clause, and the verb (phrase) that is inside the relative clause. Consider the following examples.

(4.20) Rira ti Jimọ ọ ra adiẹ
buying TI Jimọ HTS buy chicken
"The fact/way Jimọ bought a chicken"

(4.21) *Jije ti Jimọ ọ ra adiẹ
eating TI Jimọ HTS buy chicken

(4.22) Rira adiẹ ti Jimọ ọ ra adiẹ
buying chicken TI Jimọ HTS buy chicken
"The fact/way Jimọ bought a chicken"

(4.23) *Rira nkan ti Jimọ ọ ra adiẹ
buying something TI Jimọ HTS buy chicken

(4.24) *Rira adiẹ ti Jimọ ọ ra nkan
buying chicken TI Jimọ HTS buy something

The obvious generalization (and the one made uniformly by linguists working on Yoruba and other West African languages) is that in a verbal relative clause the element to the left of *ti* is a copy of the predicate to the right of the *ti*.[14] We

[14]The relativized predicate has only recently come into focus (Bùlì [76], Krio [133], and Yoruba [9, 10] are some of the few languages to have been subject to such scrutiny) in the

will concentrate here on the relation between the relativized predicate and the predicate in its relative clause, ignoring the fact that the relativized predicate bears nominal morphology (the gerundive prefix *Cí-* which is quite productive in combining with verb phrases in the language). We go through a derivation of the verb phrase *ra adiẹ* (from the sentence *Jimọ ọ ra adiẹ*) immediately below, in order to determine which phrases exist and can be copied.

1. merge(ϵ::=>tv =d V, ra::tv)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \texttt{=d V}$$

2. merge(ϵ::=n *d -k -q, adiẹ::n)

$$(\epsilon,\ \epsilon,\ \mathsf{adiẹ}) : \texttt{*d -k -q}$$

3. merge(1, 2)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \texttt{V},\ (\mathsf{adiẹ},\ \texttt{-k -q})$$

4. merge(ϵ::=>V +k =d +q v, 3)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \texttt{+k =d +q v},\ (\mathsf{adiẹ},\ \texttt{-k -q})$$

5. move(4)

$$(\epsilon,\ \mathsf{ra},\ \epsilon) : \texttt{=d +q v},\ (\mathsf{adiẹ},\ \texttt{-q})$$

languages of West Africa. However, the relativized predicate in Yoruba looks quite similar to the much more widely investigated predicate cleft construction, for which there is a not insubstantial body of cross-linguistic data. In many languages with this construction (Twi, Nupe [89], Bùlì [76]), the predicate clefted has been nominalized (in Twi this is evident particularly when the predicate clefted is defective in its paradigm, like *tea mu* "shout"). Kandybowicz [89] accounts for the nominalization by claiming that the scope of the nominalization is the entire clause. The merits of such an approach to Nupe predicate clefts notwithstanding, it does not fit well with the simple fact that the relativized predicate in Yoruba is in the normal gerundive nominalized form that occurs abundantly elsewhere in the language. In short, we would like to treat the nominal morphology on gerunds as identical to the nominal morphology on relativized predicates. I do not know how to do this elegantly.

6. merge(5, Jimọ::*d -k -q)

$$(\epsilon,\ \text{ra},\ \epsilon) : \text{+q v},\ (\text{adiẹ},\ \text{-q}),\ (\text{Jimọ},\ \text{-k -q})$$

7. move(6)

$$(\text{adiẹ},\ \text{ra},\ \epsilon) : \text{v},\ (\text{Jimọ},\ \text{-k -q})$$

As we can see, there are two stages in the derivation above at which just the verb *ra* is copyable, and a single point at which the verb plus its object is copyable (shown in figure 4.5). In order to account for the verb phrase copying in 4.22, we

verb copying	verb phrase copying
ra::tv	(adiẹ, ra, ϵ):v, (Jimọ, -k -q)
(ϵ, ra, ϵ):V, (adiẹ, -k -q)	

Figure 4.5: Copyable Constituents

add the lexical item ϵ::=>v v -f, which will combine with the expression in step 7 to yield

8. merge(ϵ::=>v v -f, 7)

$$(\epsilon,\ \text{ra},\ \text{adiẹ}) : \text{v -f},\ (\text{Jimọ},\ \text{-k -q})$$

The derivation continues as in the relative clause example above.

9. merge(ϵ::=>v perf, 8)

$$(\epsilon,\ \epsilon,\ \epsilon) : \text{perf},\ (\text{ra adiẹ},\ \text{-f}),\ (\text{Jimọ},\ \text{-k -q})$$

10. merge(V::=perf +k +q s, 9)

$$(\epsilon,\ \text{V},\ \epsilon) : \text{+k +q s},\ (\text{ra adiẹ},\ \text{-f}),\ (\text{Jimọ},\ \text{-k -q})$$

11. move(10)

$$(\epsilon,\ \mathsf{V},\ \epsilon) : \texttt{+q s},\ (\text{ra adiẹ},\ \texttt{-f}),\ (\text{Jimọ},\ \texttt{-q})$$

12. move(11)

$$(\text{Jimọ},\ \mathsf{V},\ \epsilon) : \texttt{s},\ (\text{ra adiẹ},\ \texttt{-f})$$

13. merge(ti::=s +f n, 12)

$$(\epsilon,\ \text{ti},\ \text{Jimọ V}) : \texttt{+f n},\ (\text{ra adiẹ},\ \texttt{-f})$$

14. move(13)

$$(\text{ra adiẹ},\ \text{ti},\ \text{Jimọ V}) : \texttt{n}$$

Notice, of course, that the vP is by default moved, and not copied. Something thus must be said to change this default behaviour. In chapter 3 we explored two strategies for the pronunciation of multiple copies. One, based on Nunes' optimality theoretic framework, uses reanalysis to force pronunciation of a chain link, while simultaneously removing it from consideration when evaluating which (remaining) chain link should have the distinction of being pronounced. The other, based loosely on Koopman's complexity filters, simply forces pronunciation of a chain link, without otherwise affecting its enclosing chain. Adopting the Koopmanian approach, we are led to put a 'reanalysis' diacritic on the =>v feature of the perfective head (which becomes ϵ::=>v̂ perf). This diacritic forces pronounciation of the lower copy of *ra adiẹ* in step 9, yielding

9. merge(ϵ::=>v̂ perf, 8)

$$(\epsilon,\ \text{ra},\ \text{adiẹ}) : \texttt{perf},\ (\text{ra adiẹ},\ \texttt{-f}),\ (\text{Jimọ},\ \texttt{-k -q})$$

10. merge(V::=perf +k +q s, 9′)

$$(\epsilon, \text{V, ra adiẹ}) : \text{+k +q s},\ (\text{ra adiẹ, -f}),\ (\text{Jimọ, -k -q})$$

11. move(10′)

$$(\epsilon, \text{V, ra adiẹ}) : \text{+q s},\ (\text{ra adiẹ, -f}),\ (\text{Jimọ, -q})$$

12. move(11′)

$$(\text{Jimọ, V, ra adiẹ}) : \text{s},\ (\text{ra adiẹ, -f})$$

13. merge(ti::=s +f n, 12′)

$$(\epsilon, \text{ti, Jimọ V ra adiẹ}) : \text{+f n},\ (\text{ra adiẹ, -f})$$

14. move(13′)

$$(\text{ra adiẹ, ti, Jimọ V ra adiẹ}) : \text{n}$$

Note that, if we were to treat the higher copy of the predicate as having been reanalyzed in Nunes' sense (i.e. the +f feature on *ti* has the reanalysis diacritic), we could no longer maintain the formal connection between the relativized predicate construction and the relative clause construction (i.e. the *ti* in the relativized predicate construction would be different from the *ti* in the relative clause construction), as otherwise we would erroneously predict copying in *all* relative constructions.

Turning now to the derivation of the verb copying in example 4.20, we are confronted with a choice—which of the two stages of the derivation which would give rise to the correct form for the relativized predicate in 4.20 should we decide to copy? Relativizing at the tv level could be achieved with the lexical item ϵ::=>tv tv -f, and at the V level with the lexical item ϵ::=>V V -f. Copying at

these levels requires all `tv` respectively `V` selecting features to be marked with the reanalysis diacritic. We will need both of these lexical items, as we shall now see.

Sentences like 4.13 (repeated below) have not only the relativized predicative forms as in 4.25 and 4.27, where just the verb or the verb phrase are copied respectively, but also as shown in 4.26, where what is copied is a discontinuous surface string.[15]

(4.13) Jimọ ọ ti Akin ṣubu
Jimọ HTS push Akin fall
"Jimọ pushed Akin down"

(4.25) Titi ti Jimọ ọ ti Akin ṣubu
pushing TI Jimọ HTS push Akin fall

(4.26) Titi ṣubu ti Jimọ ọ ti Akin ṣubu
pushing fall TI Jimọ HTS push Akin fall

(4.27) Titi Akin ṣubu ti Jimọ ọ ti Akin ṣubu
pushing Akin fall TI Jimọ HTS push Akin fall

We step through a derivation of each of these expressions (derivation a is for 4.25, b is for 4.26, and c is for 4.27). The derivations proceed identically at first, up to the point when split the copying targets different stages, and then are continued identically thereafter. We copy at steps 2a, 5b, and 9c respectively. Figure 4.6 shows the derivation of the basic sentence 4.13. The points at which we may copy, and the shape of the copy at that point are indicated with arrows. We begin by control merging *Akin* and *ṣubu*.

1. cmerge(ṣubu::=d v, Akin::*d -k -q)

$$(\epsilon, \text{ṣubu}, \epsilon) : \text{v}, \ (\text{Akin}, \text{*d -k -q})$$

[15]Which, incidentally, makes any attempt to analyze the copying in the Yoruba relativized predicate construction as post-syntactic (phonological) reduplication (as, for instance, has been argued for Bengali [57]) that much more difficult.

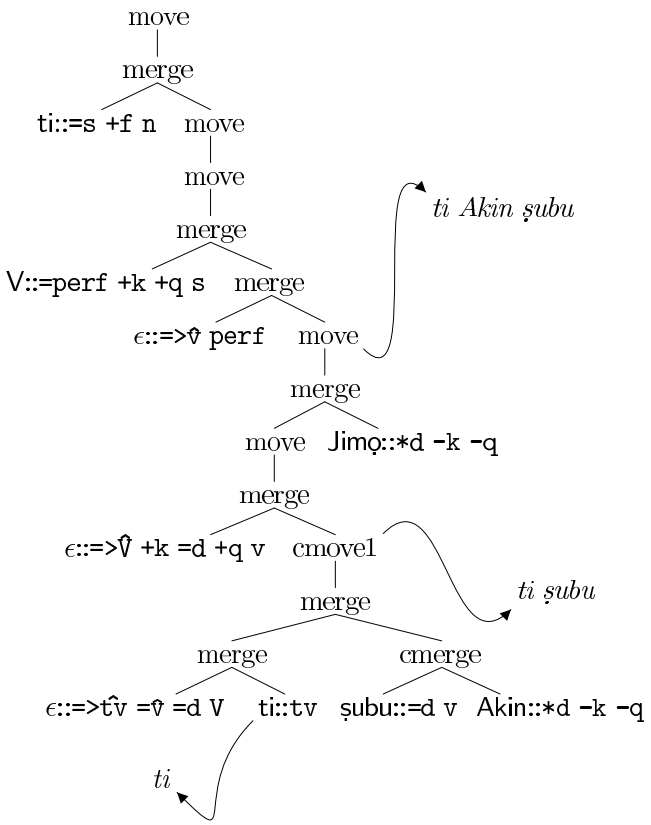


Figure 4.6: Copying at different points in the derivation

2. cmerge(ṣubu::=d v, Akin::*d -k -q)

$$(\epsilon,\ \text{ṣubu},\ \epsilon) : \texttt{v},\ (\text{Akin},\ \texttt{*d -k -q})$$

3. cmerge(ṣubu::=d v, Akin::*d -k -q)

$$(\epsilon,\ \text{ṣubu},\ \epsilon) : \texttt{v},\ (\text{Akin},\ \texttt{*d -k -q})$$

Next we prepare to copy the verb *ti* in derivation a, while in derivations b and c we ready *ti* to be in a serial verb construction.

1. merge(ϵ::=>t̂v tv -f, ti::tv)

$$(\epsilon, \text{ti}, \epsilon) : \text{tv -f}$$

2. merge(ϵ::=>t̂v =v̂ =d V, ti::tv)

$$(\epsilon, \text{ti}, \epsilon) : \text{=v =d V}$$

3. merge(ϵ::=>t̂v =v̂ =d V, ti::tv)

$$(\epsilon, \text{ti}, \epsilon) : \text{=v =d V}$$

Derivation a is now one step behind derivations b and c, which now merge the vP *ṣubu Akin* with *ti*.

1. merge(ϵ::=>t̂v =v̂ =d V, 2a)

$$(\epsilon, \text{ti}, \epsilon) : \text{=v =d V}, \ (\text{ti}, \text{-f})$$

2. merge(2b, 1b)

$$(\epsilon, \text{ti}, \text{ṣubu}) : \text{=d V}, \ (\text{Akin}, \text{*d -k -q})$$

3. merge(2c, 1c)

$$(\epsilon, \text{ti}, \text{ṣubu}) : \text{=d V}, \ (\text{Akin}, \text{*d -k -q})$$

Derivation a sets up *ti* and *ṣubu Akin* in a serial verb construction, and b and c control move *Akin*.

1. merge(3a, 1a)

$$(\epsilon, \text{ti}, \text{ṣubu}) : \text{=d V}, \ (\text{ti}, \text{-f}), \ (\text{Akin}, \text{*d -k -q})$$

2. cmove1(3b)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{V},\ (\text{Akin},\ \texttt{-k}\ \texttt{-q})$$

3. cmove1(3c)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{V},\ (\text{Akin},\ \texttt{-k}\ \texttt{-q})$$

Derivation a has caught up with b, which is preparing to copy the VP *ti ṣubu*. Derivation c continues building the vP.

1. cmove1(4a)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{V},\ (\text{ti},\ \texttt{-f}),\ (\text{Akin},\ \texttt{-k}\ \texttt{-q})$$

2. merge(ϵ::=>$\hat{\text{V}}$ V -f, 4b)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{V}\ \texttt{-f},\ (\text{Akin},\ \texttt{-k}\ \texttt{-q})$$

3. merge(ϵ::=>$\hat{\text{V}}$ +k =d +q v, 4c)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{+k}\ \texttt{=d}\ \texttt{+q}\ \texttt{v},\ (\text{Akin},\ \texttt{-k}\ \texttt{-q})$$

Derivations a and b are a step behind derivation c, which checks the case of *Akin*.

1. merge(ϵ::=>$\hat{\text{V}}$ +k =d +q v, 5a)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{+k}\ \texttt{=d}\ \texttt{+q}\ \texttt{v},\ (\text{ti},\ \texttt{-f}),\ (\text{Akin},\ \texttt{-k}\ \texttt{-q})$$

2. merge(ϵ::=>$\hat{\text{V}}$ +k =d +q v, 5b)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{+k}\ \texttt{=d}\ \texttt{+q}\ \texttt{v},\ (\text{ti ṣubu},\ \texttt{-f}),\ (\text{Akin},\ \texttt{-k}\ \texttt{-q})$$

3. move(5c)

$$(\epsilon,\ \text{ti},\ \text{ṣubu}) : \texttt{=d}\ \texttt{+q}\ \texttt{v},\ (\text{Akin},\ \texttt{-q})$$

Derivations a and b check *Akin*'s case, and derivation c introduces the subject *Jimọ*.

1. move(6a)

   (ϵ, ti, ṣubu) : =d +q v, (ti, -f), (Akin, -q)

2. move(6b)

   (ϵ, ti, ṣubu) : =d +q v, (ti ṣubu, -f), (Akin, -q)

3. merge(6c, Jimọ::*d -k -q)

   (ϵ, ti, ṣubu) : +q v, (Akin, -q), (Jimọ, -k -q)

Derivations a and b introduce *Jimọ*, and c checks the last feature of the object *Akin*. Note that derivation c has derived a vP.

1. merge(7a, Jimọ::*d -k -q)

   (ϵ, ti, ṣubu) : +q v, (ti, -f), (Akin, -q), (Jimọ, -k -q)

2. merge(7b, Jimọ::*d -k -q)

   (ϵ, ti, ṣubu) : +q v, (ti ṣubu, -f), (Akin, -q), (Jimọ, -k -q)

3. move(7c)

   (Akin, ti, ṣubu) : v, (Jimọ, -k -q)

Derivations a and b catch up with c, which is preparing to copy its newly constructed vP.

1. move(8a)

   (Akin, ti, ṣubu) : v, (ti, -f), (Jimọ, -k -q)

2. move(8b)

(Akin, ti, ṣubu) : v, (ti ṣubu, -f), (Jimọ, -k -q)

3. merge(ϵ::=>v̂ v -f, 8c)

(ϵ, ti, Akin ṣubu) : v -f, (Jimọ, -k -q)

The copying done, derivations a, b, and c proceed in tandem from here on out.

1. move(move(merge(V::=perf +k +q s, merge(ϵ::=>v̂ perf, 9a))))

(Jimọ, V, ti Akin ṣubu) : s, (ti, -f)

2. move(move(merge(V::=perf +k +q s, merge(ϵ::=>v̂ perf, 9b))))

(Jimọ, V, ti Akin ṣubu) : s, (ti ṣubu, -f)

3. move(move(merge(V::=perf +k +q s, merge(ϵ::=>v̂ perf, 9c))))

(Jimọ, V, ti Akin ṣubu) : s, (ti Akin ṣubu, -f)

Having constructed expressions of type s, each with a single moving constituent of type -f, we merge the relative pronoun *ti*.

1. merge(ti::=s +f n, 10a)

(ϵ, ti, Jimọ V ti Akin ṣubu) : +f n, (ti, -f)

2. merge(ti::=s +f n, 10b)

(ϵ, ti, Jimọ V ti Akin ṣubu) : +f n, (ti ṣubu, -f)

3. merge(ti::=s +f n, 10c)

$$(\epsilon, \text{ti, Jimọ V ti Akin ṣubu}) : \texttt{+f n}, (\text{ti Akin ṣubu, } \texttt{-f})$$

Finally, we move the -f marked verbal copies to the specifier of *ti*. Note that there is no need to differentiate between the sizes of the copies at this point, such having been done already at an earlier stage of the derivation.

1. move(11a)

$$(\text{ti, ti, Jimọ V ti Akin ṣubu}) : \texttt{n}$$

2. move(11b)

$$(\text{ti ṣubu, ti, Jimọ V ti Akin ṣubu}) : \texttt{n}$$

3. move(11c)

$$(\text{ti Akin ṣubu, ti, Jimọ V ti Akin ṣubu}) : \texttt{n}$$

Given a sentence with multiple verbs as in 4.5 (repeated below), we predict (correctly) that the first verb in the series (*ra*) may be copied (4.28), that the entire vP may be copied (4.30), that the entire sequence *ra-se-je* may be copied (4.29), but that the first two verbs may not be copied to the exclusion of the last (4.31).[16]

[16]We also predict, incorrectly, that just *se*, just *jẹ*, or the sequence *se-jẹ* can be copied. These have the flavour of an A-over-A violation, which is naturally expressed in our framework in terms of preferentially copying at the highest point possible (given a particular category copied—tv, V, or v). Though this smacks of transderivational economy, it can be implemented within our present system in much the same way we have dealt with other 'tricky' cases of putative competition between derivations. We might add a (finite valued) parameter to our moving copied predicates, which records their category, and prohibits them to move past any head of the same category. This is not equivalent to the more general transderivational economy condition requiring, for a particular category, the highest possible copy of it, as our mechanism, unlike the transderivational economy condition, builds in a window of applicability (as long as the copied predicate is moving), which correctly allows for copies within copies.

A more serious problem is that we predict that the first verb plus object (*ra adiẹ*) is not a copyable predicate. However, it seems to be. Interestingly, it seems to have something in

(4.5) Jimọ ọ ra adiẹ se jẹ
Jimọ HTS buy chicken cook eat
"Jimọ bought the chicken to cook and eat"

(4.28) Rira ti Jimọ ọ ra adiẹ se jẹ
buying TI Jimọ HTS buy chicken cook eat

(4.29) Rira se jẹ ti Jimọ ọ ra adiẹ se jẹ
buying cook eat TI Jimọ HTS buy chicken cook eat

(4.30) Rira adiẹ se jẹ ti Jimọ ọ ra adiẹ se jẹ
buying chicken cook eat TI Jimọ HTS buy chicken cook eat

(4.31) *Rira se ti Jimọ ọ ra adiẹ se jẹ
buying cook TI Jimọ HTS buy chicken cook eat

We rule out 4.31 because *ra se* is never a constituent to the exclusion of *jẹ*. Our treatment of object sharing serialization builds vPs up from right to left. Accordingly, *ra* combines with a vP contianing *se* and *jẹ*, but not with one containing just *se*.

Our complete lexicon is given in figure 4.7. Clearly, there are similarities between this lexicon and the one in chapter 2 for English, in particular among the functional lexical items. Some of these similarities are due to the way we have set up our semantics, and some are due to our principle of immediacy.[17] However, we did not set out to make our grammar of Yoruba resemble our grammar of English. Indeed, we required Yoruba-internal evidence for DP movement (which we found,

common with the other problem we have let lie: the fact that the copied predicate is nominalized. Both facts seem to be amenable to description in terms of (very restricted) sideward movement [131, 159]. If we allow the object *Akin* to control move to *ti*, merging its containing vP only after *Akin* has checked its licensee features, we have an appropriate-sized constituent for this non-standard copy. Similarly, if we allow the copied predicate to move sidewards into the complement position of the gerundive prefix *Ci-*, we can state the generalization that the prefix on the copied predicate *is* the gerundive prefix it so resembles. Needless to say, this remains a pipe dream (and a serious problem) until it is worked out.

[17]Others are possibly an artifact of a too constrained data set. I set these aside here, and focus instead on what it would mean, were the similarities to be real.

Jimọ::*d -k -q	
Akin::*d -k -q	
igi::n	ϵ::=n *d -k -q
adiẹ::n	ϵ::=n *d -k -q -f
ṣubu::=d v	tobi::=d v
fo::inch	ϵ::=>inch =d v
	ϵ::=>inch tv
ti::tv	ra::tv
jaa::tv	se::tv
	jẹ::tv
ϵ::=>$\hat{\text{t}}$v =d V	ϵ::=>$\hat{\text{t}}$v tv -f
ϵ::=>$\hat{\text{t}}$v =$\hat{\text{V}}$ =d V	ϵ::=>$\hat{\text{V}}$ V -f
ϵ::=>$\hat{\text{t}}$v =$\hat{\text{v}}$ =d V	ϵ::=>$\hat{\text{v}}$ v -f
ϵ::=>$\hat{\text{V}}$ +k =d +q v	
ϵ::=>$\hat{\text{v}}$ perf	
V::=perf +k +q s	ti::=s +f s

Figure 4.7: A grammar for a fragment of Yoruba

based on the causative-inchoative alternation). A major research question asks what the range of variation in human languages is. By working with minimalist

grammars (with copying), we already place a limit on the definable structures. As our semantics does not rule out any syntactically definable structure, it remains an open question *why* our descriptions of these languages look similar. One natural idea is to wonder whether the requirement that a particular language be able to express a particular range of semantic values forces minimalist grammars for those languages to look similar. Suppose that something like this is on the right track.[18] This amounts to formally reconstructing an extra-grammatical influence on language, and attributing to it the existence of certain 'language universals.'

2 Summary

Theories of the mechanisms underlying copying, such as developed in chapter 3, begin to diverge once confronted with data involving syntactically complex copies, such as presented herein, from Yoruba. The sometimes opaque relation between the two copies in the relativized predicate construction receives an elegant and unified description from our derivational copy mechanism, and renders untenable an alternative, post-syntactic mechanism of phonological reduplication (as has been suggested for phrasal copying in Bengali [57]). Along the way, we have developed an account of object sharing in serial verb constructions, and of relativization, both over DPs, as well as over predicates. Our account of object sharing in serial verb constructions treats this phenomenon as a species of control (with [43]), which unifies object sharing in serial verb constructions in Yoruba with control in English. Re-using mechanisms in this way allows us to pinpoint 'the difference' between Yoruba and English by virtue of which the former, and

[18] Indeed, in the context of our formalization of the syntax-semantics interface, we can even begin to fruitfully address this question!

not the latter, allows for serial verb constructions. A glance at our lexicon reveals the 'serial verb construction parameter' [164] to consist of the lexical items ϵ::=>t̂v =v̂ =d V and ϵ::=>t̂v =V̂ =d V, which allow transitive verbs in Yoruba to optionally select verb phrase (vP or VP) complements. Again and again, we have seen that the verbal categories (v, V, and tv) behave similarly in our grammar; they are each potential sites of relativization, all and only these verbal categories bear 'reanalysis' diacritics, and v and V are 'serializable' in the sense described above. Formally, we see that if there is a lexical item that selects one of these verbal categories, there is often another, otherwise identical one that selects another. This fact calls to mind Grimshaw's [68] notion of an extended projection, which our current system, with its unstructured categories, does not do justice to. Our analysis of relative clauses is of the raising variety, which has enjoyed a reinvigoration with the advent of [90]. Most interesting from this perspective is the straightforward analysis we have been able to give of stacked relative clauses. Although we view stacked relative clauses as hierarchically structured in the syntax (similarly to [14]), with later relatives modifying the head noun plus previous relatives (e.g. [[N RC_1] RC_2]), our simple treatment of the semantics of relative clauses derives elegantly a 'flat' conjunctive interpretation in cases of multiple relativization.

Appendices

D–1 On the complexity of natural language

In this section we step back from any particular analysis of Yoruba, and focus rather on the shape and complexity of its constructions, in particular its relativized predicate construction. Examining the pattern of copying present in Yoruba, we will conclude that any satisfactory grammar for Yoruba must be capable of copying copies (of copies...), and thus also of generating non-semi-linear languages like a^{2^n}.

This conclusion is a momentous one, contradicting (if not in letter then in spirit) the widely held hypothesis of the mild context sensitivity of natural language, which we recount in § D–1.1. As a practical consequence, it means that many of our grammatical theories are too weak to describe natural languages, predicting of things that do in fact exist, that they couldn't possibly.[19]

These conclusions have been argued for before, based on Suffixaufnahme (case stacking) in Old Georgian [122], and on scrambling in German [11]. What is different in the present context is that not only are we showing that, under a natural description, a particular phenomenon is not capturable by mildly context-sensitive formalisms, but we are also offering a slightly stronger formalism which is able to naturally describe this phenomenon in just the way people want to be able to.

[19] We can really only assess this for those theories that have been made explicit. Of these, two of the most widely used, Combinatory Categorial Grammar and Tree Adjoining Grammar in any of its incarnations (except those, such as [26], which add an operation of syntactic copying), are too weak. Head-Driven Phrase Structure Grammar, another popular theory, is untouched by this result, as the formalism makes no non-trivial predictions about natural language [85].

D–1.1 The hypothesis of the mild context-sensitivity of natural languages

The hypothesis of mild context sensitivity, first articulated in [86], is a claim that all human languages share certain characteristic properties, and that, moreover, these properties are non-accidental from a grammatical perspective. In other words, our theories of language should predict that only languages that are mildly context-sensitive exist. Like claims that natural language is regular or context-free, the claim that natural languages are mildly context-sensitive is statable independently of any particular grammar formalism, which fact makes its empirical content crystal clear.

Mild context-sensitivity can be given a bipartite characterization, as a list of conditions that a language must meet.[20] The first condition is that the language be among those whose strings are recognizable by a deterministic turing machine in polynomial time. This is often called *efficient recognizability*. Although it contains the words 'efficient' and 'recognize', this criterion is emphatically *not* related to ideas about human language processing. We might just as well have characterized this property as definability in first order logic with a least fixed point operator (see e.g. [83]). The essence of this criterion is to circumscribe a class of patterns of reasonable complexity. It is a non-trivial property. Copying (ww), reversal (ww^{R}), and exponential (a^{2^n}) patterns are efficiently recognizable, whereas primacy ($\{a^p : p \text{ is prime}\}$) and theorem-hood ($\{a^g : g \text{ a gödel number of a theorem of FOL}\}$) are not. The second condition is that the language be of *constant growth*.[21] A language is of constant growth

[20] A third condition is often added to this list. This third condition requires that there be a limited number of cross-serial dependencies. This intuition has been notoriously difficult to pin down in a meaningful, grammar independent, way. As it is not clear what it should mean, I will leave it out of my characterization of mild context-sensitivity.

[21] A stronger condition, more in keeping with the intent of the 'constant growth' criterion,

just in case the size of its strings doesn't grow 'too quickly'. Intuitively, the idea is that at each step in the derivation of a sentence, the rules add only a fixed amount of new material. The language a^{2^n} is not of constant growth. Constant growth says nothing about how the words within a sentence are arranged, and is therefore independent of the criterion of efficient recognizability.[22]

D–1.2 The structure of a challenge

At first blush it seems easy enough to mount a challenge to the MCS hypothesis. We might try and look for a natural language that had too many crossing dependencies, or wasn't recognizable in polynomial time, or wasn't of constant growth. We might sit down with a consultant, and, after a month's work of sleepless elicitation, we might, our beards grown long and hair disheveled, jump one moment to our feet, our right pointer finger a lightening rod to the heavens, "Eureka" on our lips. We might run to the nearest watering hole, approach the obligatory computational linguist, and, pallid faces beaming, present our nicely typeset elicitation notes, containing thousands of example sentences, for his perusal. His haughty sneer would break our hearts, and our nights would forevermore be haunted with the words "This finite data set isn't even worth writing a regular grammar for."

In a less dramatic way, what went wrong is that there is no way to challenge the MCS hypothesis on the basis of a corpus of data. The MCS hypothesis rules

is *semilinearity*. Whereas constant growth requires that the lengths of sentences not grow too fast, semilinearity requires this of the numbers of the individual words in a sentence. The language $a^{2^n}b^*$ (the language which has an exponential number of as followed by any number of bs) is of constant growth, but is not semilinear.

[22] Although the language of primes, a^p is not efficiently recognizable, and thus neither is the language a^pb^* (the language with a prime number of as followed by any number of bs), this latter *is* of constant growth. In other words, we can 'pad out' a set of strings of non-constant growth with dummy symbols to satisfy the constant growth property. The stronger property, semilinearity, is also independent of efficient recognizability, with $a^pb^* + b^*a^*$ (the language in which as precede bs if there are a prime number of them, and follow the bs otherwise) being semilinear, but not efficiently recognizable.

out only certain infinite sets of sentences. Corpora are of necessity finite. In order to challenge the MCS hypothesis, we need to first generalize from the observed data to an infinite set of potential data, of which the observed data is but a small sample. Then we can decide whether or not our generalization is compatible with the MCS hypothesis.

A good challenge will have the following two attributes. First, the generalization argued for is a reasonable one, in the sense that it is an instance of an already established type. That is, for the proposal to be empirically secure, we expect it to be parsimonious, compatible with and even supported by other independently motivated assumptions about language mechanisms. Second, there aren't alternative reasonable generalizations compatible with the data that are compatible with the MCS hypothesis. Accordingly, a response to a good challenge takes the form of coming up with a novel reasonable generalization compatible with both the data and the MCS hypothesis.

D–1.2.1 Scrambling in German

Becker et al. [11] challenge the MCS hypothesis on the basis of scrambling in German. In German, the order of major constituents is relatively free. Example 4.32 is a subordinate clause with five major non-predicative constituents, any permutation of which is said to preserve grammaticality.[23]

(4.32) . . . dass [eine hiesige Firma] [meinem Onkel] [die Möbel]
. . . that a local company (NOM) my uncle (DAT) the furniture (ACC)
[vor drei Tagen] [ohne Voranmeldung] zugestellt hat
three days ago without notice delivered has
". . . that a local company delivered the furniture to my uncle three days ago without advance warning"

[23] All examples are from [11].

The generalization argued for is that DPs can be fronted in any order not only within their clause, but also across clauses. Motivating this generalization, they present the following data. Examples 4.33 and 4.34 illustrate that arguments of a lower clause can scramble to a higher clause.

(4.33) . . . dass bisher noch niemand [den Kühlschrank zu
. . . that so far still no one (NOM) the refrigerator (ACC) to
reparieren] versprochen hat
repair promised has
". . . that, thus far, still no one has promised to repair the refrigerator"

(4.34) . . . dass [den Kühlschrank]$_i$ bisher noch niemand [t_i zu
. . . that the refrigerator (ACC) so far still no one (NOM) to
reparieren] versprochen hat
repair promised has

From example 4.35 we infer that there is no bound on the number of clauses a DP may scramble out of.

(4.35) . . . dass [den Kühlschrank]$_i$ bisher noch niemand [[t_i zu
. . . that the refrigerator (ACC) so far still no one (NOM) to
reparieren] zu versuchen] versprochen hat
repair to try promised has
". . . that so far no one has promised to try to repair the refrigerator"

Finally, scrambling of one element does not block the scrambling of another, even if these elements are embedded in infinitival clauses (4.36).

(4.36) . . . dass [dem Kunden]$_i$ [den Kühlschrank]$_j$ bisher noch
. . . that the client (DAT) the refrigerator (ACC) so far still
niemand t_i [[t_j zu reparieren] zu versuchen] versprochen hat
no one (NOM) to repair to try promised has
". . . that so far no one has promised the client to try to repair the refrigerator"

Becker et al. articulate their generalization in the following manner.

- There is no bound on the distance over which each element can scramble.
- There is no bound on the number of unbounded dependencies that can occur in each sentence.

Under the assumption that each element is introduced in the clause immediately containing the verb that assigns it case, Becker et al. show that no MCFG (and, by extension, none of the other equivalent MCS formalisms)[24] can define a language meeting these conditions.

Does this challenge satisfy our criteria above? The first criterion, that the generalization be natural, and be of an already established type, is met; scrambling as a phenomenon is abundantly attested, and has been the subject of much theoretical scrutiny. The generalization drawn is common currency in the field, and is thus hardly assailable on these grounds. Consequently, at the time of writing, the second criterion, that there be no alternative, weaker but still natural analyses, was met as well.

Joshi, Becker, and Rambow's Response Joshi et al. [87] note that speakers tend not to be able to comprehend (and thus judge) sentences in which a scrambled element is assigned case by a deeply embedded verb. They observe that (tree-local) MC-LTAGs, which generate the same string sets as standard (L)TAGs, assign structural descriptions to scrambled sentences which accord with our intuitions regarding which verbs introduce which arguments. However, MC-LTAGs can only derive sentences in which no element has scrambled out of more than two containing clauses. In effect, Joshi et al. argue that the standard ac-

[24]Even though the equivalence proofs establish that the formalisms generate the same (string) languages, they typically do this by showing how to construct a weakly equivalent grammar in the target formalism given one in the source as input. These constructions typically preserve a great deal of the derivational structure of expressions.

count of scrambling is correct, but that there is a principled upper bound of two on the number of clauses scrambling can move out of.

D–1.2.2 Case Stacking in Old Georgian

Michaelis and Kracht [122] present a challenge to the MCS hypothesis centered around the phenomenon of case stacking in Old Georgian.[25]

(4.37) govel-i igi sisxl-i saxl-isa-j m-is
all-NOM ART.NOM blood-NOM house-GEN-NOM ART-GEN
Saul-is-isa-j
Saul-GEN-GEN-NOM
"all the blood of the house of Saul"

The generalization Michaelis and Kracht argue for is roughly the following. Within a DP, the exponent of the case assigned to the head noun 'trickles down' to the DPs governed by the first. Thus, a rough structural characterization of 4.37 is as in figure 4.8. A DP will be marked with the sequence of cases determined

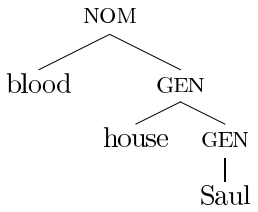


Figure 4.8: Case stacking in Old Georgian

by traversing the tree from the DP in question to the root. Thus, *Saul* is marked with the sequence GEN-GEN-NOM, *house* with GEN-NOM, and *blood* with just NOM. More generally, complex nominative DPs with k stacked genitive DPs have

[25]Examples are from [122].

the abstract form below.

$$N_1\text{-NOM}\ N_2\text{-GEN-NOM}\ N_3\text{-GEN}^2\text{-NOM}\ \ldots\ N_k\text{-GEN}^{k-1}\text{-NOM}$$

Notice that according to this generalization, if we have a complex nominative DP with k stacked genitive DPs and we add one more genitive DP, we get k extra genitive case markers. Adding one more results in $k+1$ extra genitive case markers. Michaelis and Kracht prove that the increase in case markers grows too quickly to be semilinear.

Is this a good challenge, according to our criteria? Although the generalization Michaelis and Kracht draw is an unusual one, it is not entirely without precedent. The collection [136] assembles data on a variety of languages, all of which exhibit the phenomenon of *Suffixaufnahme*, whereby an affix of a governor is inherited by a governee. Suffixaufnahme has not, unfortunately, been subjected to rigorous theoretical analysis, and thus Michaelis and Kracht's generalization enjoys only the grossest of pre-theoretical support. This blade is double-edged, however, and the lack of theoretical attention has resulted in a paucity of available alternative analyses of this phenomenon.

Bhatt and Joshi's Response Bhatt and Joshi [13] attack the generalization drawn by Michaelis and Kracht [122], arguing that upon closer inspection of the data a different, mildly context-sensitive, pattern emerges. They argue that only the last, most deeply embedded, genitive noun has copies of all its ancestors' affixes, and that non-deepest genitives bear copies just of the nominative affix.

$$N_1\text{-NOM}\ N_2\text{-GEN-NOM}\ldots N_i\text{-GEN-NOM}\ldots N_k\text{-GEN}^{k-1}\text{-NOM}$$

This generalization and the one made by Michaelis and Kracht agree for $k \leq 3$. For $k = 4$, Bhatt and Joshi predict the following, which has one less genitive

suffix than predicted by Michaelis and Kracht.

$$N_1\text{-NOM}\ N_2\text{-GEN-NOM}\ N_3\text{-GEN-NOM}\ N_4\text{-GEN-GEN-GEN-NOM}$$

And for $k = 5$, Bhatt and Joshi predict the following, which has three fewer genitive suffixes.

$$N_1\text{-NOM}\ N_2\text{-GEN-NOM}\ N_3\text{-GEN-NOM}\ N_4\text{-GEN-NOM}\ N_5\text{-GEN-GEN-GEN-GEN-NOM}$$

Although Bhatt and Joshi eliminate the challenge posed directly by Old Georgian to the hypothesis of mild context-sensitivity, they have given no mechanism for the generation of stacked cases, or of Suffixaufnahme in general. It is not clear how to account for cases of Suffixaufnahme without appeal to a mechanism of copying, which, as we will see next, can very quickly lead to non-semilinear patterns.

D–1.3 Copying (of copies)* in Yoruba

Here I will present another challenge to the MCS hypothesis, based on the verbal relative clause construction in Yoruba. I will argue that Yoruba relativized predicates can themselves contain relativized predicates (which in turn contain relativized predicates etc.). This means that copies can be of copies. In other words, copying operations can apply iteratively. Although the set of sentences of Yoruba does not cause trouble for the MCS hypothesis, the mechanisms that we require to describe these sentences elegantly do. The claim is, then, that as soon as we are able to give a natural account of languages like Yoruba, we are also able to describe non-MCS languages using the *very same* mechanisms in the *very same* way. Therefore, while it may be true that all attested languages are semilinear, this fact, like the fact that ww^{R} is not an attested language, does not receive a syntactic explanation.

My argument is simple and can be summarized as follows. I have already argued that the relativized predicate construction in Yoruba involves copying. Now I will argue that relative clauses can be copied in the relativized predicate construction (D–1.3.1). As relative clauses are clauses, and can contain arguments beyond the one abstracted over, we must countentance relativized predicates being copied in the relativized predicate construction, from which the broader conclusion follows.

D–1.3.1 On the size of the copied object

The examples below show that although the relative clause can appear either just in the lower copy 4.39, just in the higher copy 4.40, or in both 4.41, if there are relative clauses in both higher and lower VPs, they must be identical 4.42. This shows that although relative clauses are not *required* to be copied, the grammar of Yoruba *allows* them to be.

(4.38) Olu ra adiẹ ti o gọ
Olu buy chicken TI 3S dumb
"Olu bought the stupid chicken."

(4.39) Rira adiẹ ti Olu ra adiẹ ti o gọ ko da
buying chicken TI Olu buy chicken TI 3S dumb not good

(4.40) Rira adiẹ ti o gọ ti Olu ra adiẹ ko da
buying chicken TI 3S dumb TI Olu buy chicken not good

(4.41) Rira adiẹ ti o gọ ti Olu ra adiẹ ti o gọ
buying chicken TI 3S dumb TI Olu buy chicken TI 3S dumb
ko da
not good

(4.42) *Rira adiẹ ti o gọ ti Olu ra adiẹ ti o kere
buying chicken TI 3S dumb TI Olu buy chicken TI 3S small
ko da
not good

The sentences below illustrate the similar behaviour of adjectives, suggesting that the restriction on relative clauses above is a general feature of copying DP constituents, and is not parochial to the relative clause construction. These examples show that although it is permissable for an adjective to appear only in one of the two copies (4.44), if adjectives are in both copies, the adjectives of the higher copy must be included in the lower copy (4.45 vs 4.46). Sentence 4.43 is a simple sentence, with two adjectives modifying *adiẹ*. In 4.46, the adjective *nla* surfaces only in the lower clause. In the ungrammatical 4.45, which is the adjectival counterpart to 4.42, both copies have adjectives (which is fine), but the adjectives are different (which is not).

(4.43) O ra adiẹ dudu nla
3S buy chicken black big
"He bought the big black chicken."

(4.44) Rira adiẹ dudu ti o ra adiẹ ko da
buying chicken black TI 3S buy chicken not good
"His having bought the black chicken isn't good."

(4.45) *Rira adiẹ dudu ti o ra adiẹ nla ko da
buying chicken black TI 3S buy chicken big not good

(4.46) Rira adiẹ dudu ti o ra adiẹ dudu nla ko da
buying chicken black TI 3S buy chicken black big not good
"His having bought the big black chicken isn't good."

In the examples above we have seen that complex DPs with modifying relative clauses or adjectives can be copied. Example 4.48 shows that the copied relative

clause can itself contain a relative clause. From this fact we conclude that the size of the copied predicate does not have a principled upper bound, and thus we must have a mechanism that can copy *arbitrarily large things*.

(4.47) Ade ra aja nla ti o ge obinrin ti mo feran jẹ
Ade buy dog big TI 3S bite woman TI I like eat
"Ade bought the big dog that bit the woman that I love."

(4.48) Rira Φ ti Ade ra Φ ko da
buying TI Ade buy not good
"The fact that Ade bought the big dog that bit the woman that I love is not good."

where Φ ="aja nla ti o ge obinrin ti mo feran jẹ"

D–1.3.2 Assessing the challenge

The challenge to the MCS hypothesis is that given that Yoruba has constructions which involve copying of arbitrarily large structures which may itself contain copied structures, we need to have on hand a mechanism that is able to copy copies. Once we have such a mechanism in our grammar, we are able to generate languages that are not of constant growth.

Our challenge is a strong one. Not only is the generalization we have argued for the obvious one, it is robustly attested in numerous West African (influenced) languages. Bùlì (Gur) [76], Krio (Creole) [133], Twi (Kwa), Vata (Kru) [96], Wolof (Atlantic) and many others have relativized predicate constructions, in which a copy of a verbal constitutent appears both as the head of the relative clause, and internally to the clause. These languages differ with respect to the size of the copied constituent, with Twi on one end of the spectrum allowing only a single verb root to be copied (4.50), Wolof permitting verbal complexes to be

copied (4.52), and Yoruba on the other end allowing full VPs to be copied.[26]

(4.49) me-ma Kofi sika
1S-give Kofi money
"I give Kofi money."

(4.50) me-kyiri ma a me-ma Kofi sika
1S-hate give REL 1S-give Kofi money
"I hate that I give Kofi money."

(4.51) gaaw-na-ñu a door Isaa
quick-C-3P A hit Isaa
"They hit Isaa quickly."

(4.52) gaaw a door bi ñu gaaw a door Isaa moo-ma jaaxal
quick A hit CL.C 3P quick A hit Isaa 3S-3P surprise
"The fact that they hit Isaa quickly surprised me."

Moreover, analyses appealing to mechanisms like ours abound in the literature. Copying of some sort has been assumed to underlie ellipsis [40, 128, 150], A-not-A questions in Mandarin Chinese [82, 142], predicate clefts in languages as diverse as Hebrew [103] and Korean [27], and free relatives in languages as diverse as Bambara [49] and Italian [70], not to mention the plethora of analyses of non-surface-copying phenomena using the copy theory of movement in the minimalist program.

Copying is big trouble for current MCS formalisms—they can do it to a limited extent, but only by encoding the string component of a lexical item in its category. This makes the structures assigned to copies very unnatural, and therefore the *same* syntactic generalization needs to be stated twice: once over non-copies, and then once over copies, making the resulting grammars unnecessarily complex (a

[26]The Twi data is from my consultant, Selassie Ahorlu, a native speaker of the Asante dialect. Thanks to Harold Torrence for the Wolof data, amassed during countless hours of painstaking elicitation.

point made nicely by Pullum [139]). The natural and obvious generalizations I have drawn about Yoruba, where there exist copies with copies (of copies...) contained in them, is impossible to be stated in these formalisms.

CHAPTER 5

Conclusions

Copying exists. There are constructions in natural language that require reference to identity of subparts of expressions for their description. This much, at least, is uncontroversial. What is controversial is the proper locus of explanation of these facts; whether copying should be considered syntactic, phonological, semantic, or extra-grammatical. This is not something that can be decided a priori. What we need are explicit theories of copying at syntactic, phonological, semantic, and extra-grammatical levels, which can then be compared with each other, the victor being the one that results in the simplest overall theory of language, and mind.

This work contributes to this task by providing two fully explicit theories of copying at the syntactic level. Although framed in the context of a particular formal theory of syntax, minimalist grammars, the basic ideas are simple, and can be reimplemented in other syntactic frameworks. They can be succinctly presented as follows. Copying may be of intermediate stages in a derivation, or of the derivation itself. In minimalist grammars, these two perspectives coincide. Empirically, what they require of analyses of a given language is that there be enough derivational constituents to support the range of copies available for a given sentence. It is a nice feature of our grammatical framework that the same derivation provides us with all (and only) the necessary constituents in a wide range of cases.

There are two basic syntactic questions that can be asked as regards copies in

language. The first, investigated by the vast majority of linguists working within the chomskyian community, regards the distribution of copies: their relationship to each other, and to other formatives in the sentence. This kind of question has proven extremely fruitful in the past (e.g. the generalizations about the distribution of wh-words and their traces), and I see no reason to doubt its continued fecundity. The second question, investigated here, regards the construction of copies: how they are internally constituted, and the relation they bear to the constitution of the objects of which they are copies. In short; how copies get there to be distributed throughout the sentence in the first place. Clearly, any complete theory of our linguistic competence must contain answers to both of these questions.

Although we are conducting our investigations of language at the level of linguistic competence (the description of the relation between form and meaning that is computed during comprehension and production), it is natural to think of essential aspects of our descriptions of this relation as having some sort of psychological reality. A particularly intuitive perspective is that the derivation (i.e. the structure assigned to expressions by the competence grammar) is at least implicitly computed during parsing and generation. Adopting this perspective, our derivation-synchronization approach to copying does not commit us to a particular method of computing similarly derived copies during production or recognition. Indeed, a natural view to hold is that our derivational copy mechanism is simply a constraint on which portions of the derivation can be recognized as identical during parsing (in our case, subtrees). It will be an interesting exercise to construct an efficient glc parser for languages derived by synchronous minimalist grammars. A natural idea is that copies serve to 'narrow the beam'; that upon beginning to recognize a similar structure, the parser 'retries' that which worked before.

A constant question throughout this dissertation was what kinds of structures are needed to mediate the relation between form and meaning. I have tried to show that the two-level approach to syntax practiced by many (where the derivation exists alongside the derived tree) is unnecessary in a large number of cases, and that, moreover, the simpler structures provided by the derivation actually allow for a more elegant statement of the interface operations. In particular, reconstruction and quantifier scope, phenomena which have been thought to require a derived tree, have been shown to be easily capturable derivationally; instead of putting 'the meaning' of an expression in the wrong place, and then readjusting it later, we can simply do it right from the get-go.

The transformational approach to the description of the relation between form and meaning has resulted in a peculiar kind of syntactic object: the chain. Expressions are typically related to multiple positions in a sentence, with their meaning contribution being naturally expressed as similarly discontinuous. Although chains in this sense are the natural locus of semantic interpretation, it has not been known how to accomodate them in a simple semantic calculus. The semantics given in chapter 2 not only assigns easily manipulable semantic terms to expressions, but also permits us to view the derivation of an expression as an underspecified semantic representation in its own right. Our semantics is compositional in the well-defined (and completely standard) sense of there being a transduction (a top-down one, with regular look-ahead, in our case) mapping derivation trees to semantic terms (which are then mappable homomorphically into the intended model).

Most fundamentally, I have tried to show that natural descriptions of natural language phenomena involving copying allow for generation of patterns exhibiting exponential growth. This would contradict the influential hypothesis of the mild

context-sensitivity of natural language, which places severe limitations on the kinds of constructions we expect to find in human languages. The most important aspect of this hypothesis is its independence from any particular theory of language. Parochial to none, it is applicable to all. I have argued, in essence, that natural languages require more computational power for their description than can be provided by mildly context-sensitive mechanisms. My challenge to this hypothesis can be met either by constructing a similarly elegant formal theory of syntax which allows for copying but which does not permit copies to contain copies, or by reanalyzing the Yoruba data. I hope to have made this latter option difficult. Until such time as my challenge is met, the formal complexity of natural language is far from unconstrained. Indeed, one of the conditions on mild context-sensitivity (efficient recognizability) continues to hold. More than this, natural language phenomena seem all amenable to description by mechanisms equivalent in expressive power to parallel multiple context-free grammars, or to synchronous minimalist grammars; both of which generate patterns properly contained in $\mathbf{P}$.

Bibliography

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

[2] A. Abeillé. Synchronous TAGs and French pronomial clitics. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING)*, volume 1, pages 60–66, 1992.

[3] P. Agbedor. Verb serialization in Ewe. *Nordic Journal of African Studies*, 3(1):115–135, 1994.

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

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

[6] Y. Awoyale. On the development of the verb infinitive phrase in Yoruba. *Studies in African Linguistics*, 14(1):71–102, April 1983.

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

[8] M. Baker. Object sharing and projection in serial verb constructions. *Linguistic Inquiry*, 20(4):513–553, Fall 1989.

[9] A. Bámgbóṣé. Relative clauses and nominalized sentences in Yoruba. In *Proceedings of the Sixth Conference on African Linguistics*, volume 20 of *Ohio State University Working Papers in Linguistics*, pages 202–209, 1975.

[10] A. Bámgbóṣé. Relativization or nominalization?: A case of structure versus meaning. *Research in Yoruba Language and Literature*, 3:87–109, November 1992.

[11] T. Becker, O. Rambow, and M. Niv. The derivational generative power of formal systems, or, scrambling is beyond LCFRS. Technical Report IRCS-92-38, Institute for Research in Cognitive Science, University of Pennsylvania, 1992.

[12] R. Bhatt. The raising analysis of relative clauses: Evidence from adjectival modification. *Natural Language Semantics*, 10(1):43–90, 2002.

[13] R. Bhatt and A. Joshi. Semilinearity is a syntactic invariant: A reply to Michaelis and Kracht (1997). *Linguistic Inquiry*, 35(4):683–692, Fall 2004.

[14] V. Bianchi. The raising analysis of relative clauses: A reply to Borsley. *Linguistic Inquiry*, 31(1):123–140, Winter 2000.

[15] P. Blache, E. Stabler, J. Busquets, and R. Moot, editors. *Logical Aspects of Computational Linguistics*, volume 3492 of *Lecture Notes in Computer Science*, Berlin, 2005. Springer.

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

[17] O. G. Bode. *Yoruba Clause Structure*. PhD thesis, University of Iowa, 2000.

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

[19] J. Bos. Predicate logic unplugged. In P. Dekker and M. Stokhof, editors,

Proceedings of the Tenth Amsterdam Colloquium, pages 133–143, Amsterdam, 1996.

[20] Ž. Bošković. On multiple Wh-fronting. *Linguistic Inquiry*, 33(3):351–383, Summer 2002.

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

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

[23] W. Buszkowski. Lambek grammars based on pregroups. In de Groote et al. [50], pages 95–109.

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

[25] V. Carstens and F. Parkinson, editors. *Advances in African Linguistics*, volume 4 of *Trends in African Linguistics*. Africa World Press, Trenton, NJ, 2000.

[26] J. Chen-Main. *On the generation and linearization of multi-dominance structures*. PhD thesis, Johns Hopkins, 2006.

[27] E. Cho and K. Nishiyama. Yoruba predicate clefts from a comparative perspective. In Carstens and Parkinson [25], pages 37–49.

[28] N. Chomsky. *New Horizons in the Study of Language and Mind*. Cambridge University Press, NY, 2000.

[29] 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.

[30] N. Chomsky. Derivation by phase. In M. Kenstowicz, editor, *Ken Hale: A Life in Language*, pages 1–52. MIT Press, Cambridge, Massachusetts, 2001.

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

[32] N. Chomsky. On phases. ms., MIT, 2005.

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

[34] N. Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, Massachusetts, 1965.

[35] N. Chomsky. *Reflections on Language*. Pantheon, NY, 1975.

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

[37] N. Chomsky. A minimalist program for linguistic theory. In Hale and Keyser [71].

[38] 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.

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

[40] S. Chung, W. A. Ladusaw, and J. McCloskey. Sluicing and logical form. *Natural Language Semantics*, 3(3):239–282, 1995.

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

[42] C. Collins. *Topics in Ewe Syntax*. PhD thesis, Massachusetts Institute of Technology, 1993.

[43] C. Collins. Argument sharing in serial verb constructions. *Linguistic Inquiry*, 28(3):461–497, Summer 1997.

[44] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at `http://www.grappa.univ-lille3.fr/tata`, 2002.

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

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

[47] A. Cormack and N. Smith. Compositionality, copy theory, and control. In A. Neeleman and R. Vermeulen, editors, *University College London Working Papers in Linguistics*, volume 14, pages 355–373. University College London, 2002.

[48] 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.

[49] C. Culy. The complexity of the vocabulary of Bambara. *Linguistics and Philosophy*, 8(3):345–352, 1985.

[50] 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.

[51] M. de Vries. *The Syntax of Relativization*. PhD thesis, Universiteit van Amsterdam, 2002.

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

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

[54] J. Engelfriet and H. Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31:71–146, 1985.

[55] G. Fanselow and D. Čavar. Distributed deletion. In A. Alexiadou, editor, *Theoretical Approaches to Universals*, pages 67–107. Benjamins, Amsterdam, 2002.

[56] 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.

[57] J. Fitzpatrick-Cole. Reduplication meets the phonological phrase in Bengali. *The Linguistic Review*, 13:305–356, 1996.

[58] J. A. Fodor. *The Modularity of Mind*. MIT Press, Cambridge, Massachusetts, 1983.

[59] 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.

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

[61] H.-M. Gärtner. *Generalized Transformations and Beyond: Reflections on Minimalist Syntax*. Akademie Verlag, Berlin, 2002.

[62] H.-M. Gärtner. Review of the copy theory of movement and linearization of chains in the minimalist program. *GLOT International*, 8(3):16–20, 1998.

[63] H.-M. Gärtner and J. Michaelis. A note on the complexity of constraint interaction: Locality conditions and minimalist grammars. In Blache et al. [15], pages 114–130.

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

[65] F. Gécseg and M. Steinby. *Tree Automata*. Akadémiai Kiadó, Budapest, 1984.

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

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

[68] J. Grimshaw. *Words and Structure*. CSLI Publications, Stanford, 2005.

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

[70] A. Gullì. *Phrasal Reduplication in Syntax*. PhD thesis, The City University of New York, 2003.

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

[72] M. Halle and A. Marantz. Distributed morphology and the pieces of inflection. In Hale and Keyser [71], pages 111–176.

[73] H. Harkema. A characterization of minimalist grammars. In de Groote et al. [50].

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

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

[76] K. Hiraiwa. *Dimensions of Symmetry in Syntax: Agreement and Clausal Architecture*. PhD thesis, Massachusetts Institute of Technology, 2005.

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

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

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

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

[81] C.-T. J. Huang. *Logical relations in Chinese and the theory of grammar*. PhD thesis, MIT, 1982.

[82] C.-T. J. Huang. Modularity and Chinese A-not-A questions. In C. Georgopoulos and R. Ishihara, editors, *Interdisciplinary Approaches to Language, Essays in Honor of S.-Y. Kuroda*, pages 305–332. Kluwer, Dordrecht, 1991.

[83] N. Immerman. Descriptive complexity: a logician's approach to computation. *Notices of the American Mathematical Society*, 42(10):1127–1133, 1995.

[84] 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, August 2005.

[85] M. Johnson. *Attribute Value Logic and The Theory of Grammar*. Number 16 in CSLI Lecture Notes Series. Chicago University Press, Chicago, 1988.

[86] A. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing: Theoretical, Computational and Psychological Perspectives*, pages 206–250. Cambridge University Press, NY, 1985.

[87] A. K. Joshi, T. Becker, and O. Rambow. Complexity of scrambling: A new twist to the competence-performence distinction. In A. Abeillé and O. Rambow, editors, *Tree Adjoining Grammars: Formalisms, Linguistic Analysis and Processing*. CSLI Publications, Stanford, 2000.

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

[89] J. Kandybowicz. *Conditions on Multiple Copy Spell-Out and the Syntax-Phonology Interface*. PhD thesis, UCLA, 2006.

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

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

[92] 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.

[93] G. M. Kobele and J. Michaelis. Two type 0-variants of minimalist grammars. In Jäger et al. [84].

[94] M. Koizumi. *Phrase Structure In Minimalist Syntax*. PhD thesis, MIT, 1995.

[95] H. Koopman. Derivations and complexity filters. In A. Alexiadou, E. Anagnostopoulou, S. Barbiers, and H.-M. Gärtner, editors, *Dimensions of Movement: From features to remnants*, number 48 in Linguistik Aktuell/Linguistics Today, chapter 8, pages 151–188. John Benjamins, 2002.

[96] H. Koopman. *The Syntax of Verbs: From Verb Movement Rules in the Kru Languages to Universal Grammar*. Foris, Dordrecht, 1983.

[97] H. Koopman and D. Sportiche. The position of subjects. *Lingua*, 85:211–258, 1991. Reprinted in Dominique Sportiche, Partitions and Atoms of Clause Structure: Subjects, agreement, case and clitics. NY: Routledge.

[98] 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.

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

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

[101] J. Lambek. Type grammars as pregroups. *Grammars*, 4(1):21–35, 2001.

[102] J. Lambek. Type grammars revisited. In A. Lecomte, F. Lamarche, and G. Perrier, editors, *Logical Aspects of Computational Linguistics*, volume 1582 of *Lecture Notes in Artificial Intelligence*, pages 1–27. Springer, New York, 1999.

[103] I. Landau. Chain resolution in Hebrew V(P)-fronting. *Syntax*, 9(1):32–66, April 2006.

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

[105] R. K. Larson. Some issues in verb serialization. In Lefebvre [109], pages 185–210.

[106] 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.

[107] N. S. Lawal. Serial verbs in Yoruba as adjunct phrases. *Lingua*, 91:185–200, 1993.

[108] F. Lee. Anaphoric R-expressions as bound variables. *Syntax*, 6(1):84–114, 2003.

[109] C. Lefebvre, editor. *Serial Verbs: Grammatical, Comparative and Cognitive Approaches*, volume 8 of *Studies in the Sciences of Language Series*. John Benjamins, Amsterdam/Philadelphia, 1991.

[110] C. Lefebvre. *Take* serial verb constructions in Fon. In *Serial Verbs: Grammatical, Comparative and Cognitive Approaches* Lefebvre [109], pages 37–78.

[111] 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.

[112] P. Ljunglöf. A polynomial time extension of parallel multiple context-free grammar. In Blache et al. [15], pages 177–188.

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

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

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

[116] D. Marr. *Vision*. W. H. Freeman and Company, New York, 1982.

[117] 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.

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

[119] J. Michaelis. Transforming linear context-free rewriting systems into minimalist grammars. In de Groote et al. [50].

[120] J. Michaelis. An additional observation on strict derivational minimalism. In Jäger et al. [84].

[121] 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.

[122] J. Michaelis and M. Kracht. Semilinearity as a syntactic invariant. In C. Retoré, editor, *Logical Aspects of Computational Linguistics*, pages 37–40, NY, 1997. Springer-Verlag (Lecture Notes in Computer Science 1328).

[123] P. Monachesi, G. Penn, G. Satta, and S. Wintner, editors. *Proceedings of the 11th conference on Formal Grammar*, July 2006. CSLI.

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

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

[126] Y. N. Moschovakis. What is an algorithm? In B. Engquist and W. Schmid, editors, *Mathematics unlimited – 2001 and beyond*, pages 919–936, Berlin, 2001. Springer Verlag.

[127] Y. N. Moschovakis. On founding the theory of algorithms. In H. G. Dales and G. Oliveri, editors, *Truth in mathematics*, pages 71–104. Oxford University Press, 1998.

[128] E. Murguía. *Syntactic Identity and Locality Restrictions on Verbal Ellipsis*. PhD thesis, University of Maryland, 2004.

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

[130] R. Nesson and S. Shieber. Simpler TAG semantics through synchronization. In Monachesi et al. [123], pages 103–117.

[131] J. Nunes. *Linearization of Chains and Sideward Movement*, volume 43 of *Linguistic Inquiry Monographs*. MIT Press, Cambridge, Massachusetts, 2004.

[132] J. Nunes. *The copy theory of movement and linearization of chains in the Minimalist Program*. PhD thesis, University of Maryland, College Park, 1995.

[133] D. K. Nylander. Factivity, presupposition and the relativised predicate in Krio. *Studies in African Linguistics*, 16(3):323–336, December 1985.

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

[135] Ọ. O. Oyelaran. The category aux in the Yoruba phrase structure. *Research in Yoruba Language and Literature*, 3:59–86, November 1992.

[136] F. Plank, editor. *Double Case: Agreement by Suffixaufnahme*. Oxford University Press, 1995.

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

[138] M. Przezdziecki. Object raising in Yorùbá. In Carstens and Parkinson [25], pages 77–90.

[139] G. K. Pullum. Render unto syntax the things which are syntax. handout of talk, September 2006. Available at `http://www.phon.ucl.ac.uk/research/pullum.pdf`.

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

[141] Z. W. Pylyshyn. *Computation and Cognition: Toward a Foundation for Cognitive Science*. Bradford Books. MIT Press, Cambridge, Massachusetts, 1984.

[142] D. Radzinski. Unbounded syntactic copying in Mandarin Chinese. *Linguistics and Philosophy*, 13(1):113–127, 1990.

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

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

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

[146] 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.

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

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

[149] E. C. Rowlands. *Yoruba (Teach Yourself)*. NTC Publishing Group, Lincolnwood, IL, 1969.

[150] I. A. Sag. *Deletion and Logical Form*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1976.

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

[152] S. M. Shieber. Synchronous grammars as tree transducers. In *Proceedings of the Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms (TAG+7)*, pages 88–95, Vancouver, 2004.

[153] 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.

[154] S. M. Shieber. Restricting the weak-generative capacity of synchronous tree-adjoining grammars. *Computational Intelligence*, 10(4):371–385, November 1994.

[155] S. M. Shieber and Y. Schabes. Synchronous tree-adjoining grammars. In *Proceedings of the 13th International Conference on Computational Linguistics (COLING)*, volume 3, pages 253–258, Helsinki,Finland, 1990.

[156] 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.

[157] E. P. Stabler. Recognizing head movement. In de Groote et al. [50], pages 254–260.

[158] 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, May 2003.

[159] E. P. Stabler. Sidewards without copying. In Monachesi et al. [123], pages 133–146.

[160] 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.

[161] 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.

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

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

[164] O. T. Stewart. *The serial verb construction parameter*. PhD thesis, McGill University, 1998.

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

[166] 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.

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

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

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

[170] 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, October 2006.