Primitive Computations in Phrase Construction

by

Chieu V. Nguyen

S.B., Mathematics
Massachusetts Institute of Technology, 2008
S.B., Computer Science and Engineering
Massachusetts Institute of Technology, 2008

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2009

Author .....
Department of Electrical Engineering and Computer Science
21 August 2009

Certified by .....
Robert C. Berwick
Professor
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Department Committee on Graduate Theses

Primitive Computations in Phrase Construction

by

Chieu V. Nguyen

Submitted to the
Department of Electrical Engineering and Computer Science
21 August 2009
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Minimalist Program in current linguistic theory seeks to explain linguistic structure in terms of economy principles, under the assumption that the human language faculty is a perfect system that performs only enough work to satisfy interface requirements. We consider processing costs as a property of syntactic computation and propose that these principles of economy may be met by the availability of alternative operations, each favorable in different circumstances. We characterize the basic Merge operation as a collection of three nested operations that apply to three corresponding levels of nested syntactic data types. In this framework, we provide an analysis of coordinate structure that uses a goal of minimizing processing cost to explain a number of peculiar characteristics of coordination, including the Coordination of Likes Constraint, the Coordinate Structure Constraint, and apparent case and agreement violations.

Thesis Supervisor: Robert C. Berwick
Title: Professor of Computer Science and Engineering and Computational Linguistics

Acknowledgments

This project had its roots in one of the assignments from Ed Barrett's fall 2007 digital poetry class (21W.772), in which I composed a hypertext-interface poem overlaid on an integer lattice spanning the surface of a Klein bottle. The resulting work, in which different infinitely long sentences could be parsed from a sequence of eight repeating words in either direction, inspired me to think about syntactic representations of repeated nested structure versus repeated flat structure. Over the course of two years, the project has gradually evolved into its current form, a development that would not have been possible without the assistance of a number of individuals.

My supervisor, Bob Berwick, has my heartfelt gratitude for generously devoting considerable time and energy to discussing ideas with me, directing me to useful resources, and seeing this project to completion. Even in the most difficult of circumstances, Professor Berwick has has gone far beyond any reasonable expectations for a thesis supervisor in order to help me assemble my work into a coherent thesis, and these few words of appreciation cannot express the magnitude of my debt.

For their invaluable advice and suggestions and many a fruitful discussion, I would like to thank Enoch Aboh, Adam Albright, Pranav Anand, Karlos Arregi, Louis Braida, Eugene Charniak, Edwin Chen, Karen Chu, Abby Cohn, Emma Cunningham, Kai von Fintel, John Hale, Jorge Hankamer, Roni Katzir, Chris Kennedy, Andrew Nevins, Norvin Richards, Jason Riggle, Matt Wagers, and Brown Westrick, as well as everyone else with whom I have had the opportunity to discuss language and computation over the years. Mark Avara and Karen Chu have generously helped out in performing various tasks related to the preparation of this thesis, and their efforts are truly appreciated.

My family and friends have my thanks for their endless support, encouragement, and patience. I am also indebted to all of the teachers, academic or otherwise, from whom I have gained knowledge and insight over the years; their wisdom has taken me far in my life's journey. In particular, my parents, Lien and Nga, have made countless sacrifices for me to be able to be here today, and I cannot thank them enough.

In addition, I am grateful for the financial aid I received as an undergraduate from the Eugene and Margaret McDermott Fund, the General MIT Scholarship, the National Merit Scholarship, and the Barry M. Goldwater Scholarship, as well as the funding I received from the Department of Electrical Engineering and Computer Science for serving as a graduate teaching assistant during the fall 2008 and spring 2009 semesters.

Finally, the staff and members of Hello! Project have my respect and gratitude for being awesome, inspiring, and—amazingly enough—amenable to an analysis using group theory and field theory. In the right circumstances, chance and a little imagination can work wonders.

To everyone whose contributions I acknowledge, whether explicitly or implicitly, I dedicate my implementation of the applicative-order **Y** combinator in the PostScript programming language:[1]

```
{ [ exch { [ exch { cvx dup exec exec } aload pop ]
cvx } aload pop 8 7 roll { cvx exec } aload pop ]
cvx dup exec }
```

[1]In its own way, the **Y** combinator in PostScript has everything to do with the central themes of this thesis. But any further explanation would take us far away from the matter at hand, so for now it will merely serve as a symbolic epigraph.

Contents

1 Introduction **9**

2 Minimality and Processing Cost **13**

2.1 Minimality 13

2.2 The Cost of Primitive Operations 15

2.3 Recursion and Iteration 18

2.4 Grammar and Processing 20

2.5 The Cost of Node Construction 26

3 Characteristics of Coordination **29**

3.1 Basic Features 29

3.2 Properties and Peculiarities 31

4 Previous Analyses **37**

4.1 Problems with Binary Branching 37

4.2 Multidimensionality 39

5 Primitive Computations **41**

5.1 Beyond Merge and Move 41

5.2 Three Levels 42

5.3 Concatenate vs. Append vs. Project 46

5.4 Processing Costs 49

5.5 Coordination as a Product of Append 51

5.6 Examples 53

5.7 Consequences 57
5.8 Three Merges or Three Nested Levels? 62

6 Conclusion **65**
6.1 Future Directions 65
6.2 Closing Thoughts 68

References **70**

Chapter 1

Introduction

An important question in current linguistic research is the concept of minimality, a property both of linguistic theories and of language itself: a minimal theory employs a minimum set of basic axioms from which observed characteristics may be derived, and in a broad sense, a minimal system performs a minimum amount of work to meet the conditions imposed by external systems that interact with it. In linguistics, the notion has been applied to the derivation and representation of linguistic structure by the human language faculty, the linguistic processing capabilities that distinguish humans from other animals. The current Minimalist approach seeks to analyze linguistic operations in a way that minimizes the computations that need to take place to meet interface conditions imposed by the cognitive systems for sending and receiving physical information-bearing signals among language users, and for developing and operating on communicable concepts (Chomsky 1995).

In this study, we take a closer look at the processing costs of the primitive operations and representations that combine to construct complex linguistic structure. We examine basic combining operations and propose that the human language faculty may employ more than one such operation as different circumstances may favor one over another due to lower processing cost, and which operation is preferred may depend on relevant conditions such as the need to check features or categorial compatibility.

In contrast to previous approaches in Minimalism, we propose that the binary

branching resulting from the combinatory Merge operation is not ideal in all situations, and suggest that the properties of coordinate constructions may favor the use of an alternative operation that constructs flat structures instead.

Chapter 2 discusses memory management and processing efficiency in relation to recursive and iterative processes in computer science. We connect these notions with processing in the human language faculty and consider the evolution of linguistic operations as a development that favors efficient operations. We raise the question of whether efficiency in processing should be regarded as a matter of processing performance as opposed to the competence of a grammar, and argue that processing cost may have a direct impact on grammatical properties. In particular, we distinguish multiple-branching nodes from flat lists as a matter of grammar in that flat lists offer more than one point of access to a structure.

In chapter 3, we provide an overview of coordinate structures from a number of languages, and identify characteristics of coordination that distinguish it from other kinds of linguistic constructions.

Surveying previous work on coordination in chapter 4, we identify problems arising from a characterization of coordination as having a necessarily binary-branching structure. We also examine multidimensional treatments of coordination.

In chapter 5, we attempt to revise previous notions of primitive operations. Starting with the approach taken by Hornstein (2009), we consider the advantages of treating the Merge operation as the result of two more primitive operations: Concatenate and Label. We depart from this approach and consider syntactic objects as forming three levels of data types, with nodes constituting the lowest level. We define three analogous operations—Concatenate, Append, and Project—that combine objects at each of the three levels. The three operations together form a three-level conception of Merge.

We argue that this three-level analysis treats each operation as involving operations at higher levels as subroutines. This avoids reduplication of similar tasks by grouping identical tasks into distinct modules, and it naturally leads to a total ordering among the operations in terms of processing cost.

Applying this analysis to coordination, we argue that coordination is an application of Append and not Project, as in recursive embedding structures. We give examples of how derivations proceed with these operations, and demonstrate how our characterization naturally explains a number of peculiarities of coordination, such as the Coordination of Likes Constraint, the Coordinate Structure Constraint, and unexpected case and agreement patterns.

In addition, we contrast our three-level Merge with Langendoen's (2003) characterization of Merge as three alternative operations.

Finally, in chapter 6, we conclude the current exploration and suggest areas that would benefit from future research, particularly further theoretical treatment of coordination and other structures, computational modeling, and psycholinguistic experimentation.

Chapter 2

Minimality and Processing Cost

2.1 Minimality

The notion of minimality has been a central focus of contemporary linguistics, particularly in the Minimalist Program, a framework launched by Noam Chomsky in the 1990s, developing out of the previous Principles and Parameters approach in transformational-generative grammar (Chomsky 1995). The core idea of Principles and Parameters is that all natural languages share a set of basic principles underlying the production and comprehension of language—a *universal grammar*—and that the variation observed across languages then arises out of a set of parameters that configure how the principles apply in any individual language. This approach has the advantage of addressing language universally as an attempt to understand the workings of the human brain, the ability of children to acquire language from limited examples, and the universality of characteristics across the world's languages.

Minimalism takes a further step and seeks to simplify the principles involved, which in earlier stages of Principles and Parameters had grown to include operations that apply in the course of deriving a syntactic structure but do not produce any observable effects. A central goal of Minimalism is to relate the science of linguistics more closely to other cognitive sciences and accordingly to examine linguistic operations in the context of computations performed by the human brain. In particular, it focuses on the external interfaces to the language system: the articulatory-phonetic

system responsible for producing and receiving the physical sounds or signs that carry linguistic information between language users, and the conceptual-intentional system responsible for producing and encoding meaning in linguistic structure. Minimalism assumes that the language system performs in an optimal manner in generating linguistic structure that satisfies the needs of both interfaces. This optimal performance can be characterized as meeting principles of economy of representation and derivation: the way in which a linguistic unit is represented must not contain extra pieces of information that are not needed to interact with the external interfaces, and the process by which complex structures are generated must not use any more operations than necessary. In particular, individual steps in a derivation are optimized locally: that is, given a set of possible completions of an intermediate structure, the computational system chooses the path that appears most economical at that point in the derivation, rather than considering all possible derivations as a whole.

To minimize derivations, Chomsky (1995) proposes that the operations that produce syntactic structure can be reduced to a minimal set of primitive operations: Select, Merge, and Move. These operations apply to assemble basic units into complex structures that are ultimately interpreted by the two interfaces. The basic units are lexical items pulled from the lexicon, a pool of units bearing semantic and phonetic information along with additional features to facilitate assembly into larger structures. The initial input to a derivation is a *numeration*, a set of items drawn from the lexicon along with indices to differentiate repeated items. The operation Select introduces an item from the numeration into the current working domain so that it can be used in constructing larger syntactic objects. The operations that combine basic lexical items are Merge and Move. Merge takes two objects and combines them into a larger object. To account for instances in which parts of a phrase are dependent on others, such as the appearance of *wh*-phrases at the front of a sentence when they are thematically associated with another part of the sentence, Chomsky proposes a third operation, Move, that makes objects that are part of a derivation available for future operations. These primitive operations can generate a wide variety of possible structures, so additional constraints need to exist to ensure that they

generate only grammatical structures: a derivation must *converge* at the interfaces to be acceptable.

2.2 The Cost of Primitive Operations

Chomsky (1995: 226) states that "Select and Merge are 'costless'; they do not fall within the domain of discussion of convergence and economy." However, in a framework whose goal is to explain the nature of a physical system, especially its interfaces, the question arises as to whether we should address in some way the physical demands that these operations place on the linguistic components of the brain. Even though we formulate the operations in theoretical terms, the operations must map to physical actions since derivations occur to interact with two physical systems. It seems reasonable to allow that these operations may incur a processing cost and that this cost is a factor in determining whether a derivation can take place.

If Select and Merge are the only possible operations of their kind, it may be irrelevant to consider their cost, but could there exist alternative operations that may be favorable? To answer this, we must distinguish these alternative operations somehow. Leaving aside the question of possible alternatives to Select, let us examine Merge in more detail and identify its characteristics.

The input to Merge is a pair of syntactic objects, and the output is another syntactic object. In a hierarchical representation, Merge creates a node corresponding to the larger object that has as children nodes corresponding to the constituent objects. The result of applying Merge any number of times is a set of binary trees since the only combining operation takes exactly two inputs. But are binary trees an ideal representation for syntactic structure?

Chomsky accepts Kayne's (1984) proposal that binary branching is sufficient for representing syntactic structure, and while its simplicity is desirable, the use of strict binary branching does introduce characteristics that may be disadvantageous for processing. Most importantly, a binary tree distinguishes among possible immediate groupings of nodes: the Merge operation is not associative since applying it to pairs

of nodes in a different order results in a different binary tree. If this representation corresponds to a physical representation in the brain, then resources are required to maintain enough information to distinguish the correct representation from other possible arrangements of the same units—unless distinguishing them is not necessary, in which case the theoretical representation encodes more information than it needs to, and it is unclear how the brain represents a corresponding structure holding less information. If the binary-tree representation does not correspond to a physical representation, then it may be preferable to adopt a representation that does. For example, if a derivation produces a structure of the form $[[A[BC]]D]$, its internal representation must store enough information to distinguish this from $[A[B[CD]]]$ or $[[[AB]C]D]$ or $[A[[BC]D]]$. This is necessary to differentiate among ambiguous sentences like "I saw the cake with a telescope," which has at least two possible readings. So in general, an operation that introduces hierarchy is desirable.

However, the need for hierarchy for one kind of linguistic construction does not imply that hierarchy is desirable everywhere. In a construction in which it is not necessary to distinguish among possible groupings of adjacent nodes, employing an operation that necessarily generates binary-branching structure and a representation that encodes such structure requires the brain to store more information than it needs. If an alternative operation exists that serves the same purpose of assembling nodes together but without creating hierarchical structure, it would be economically preferable as no resources are wasted on maintaining unneeded information.

Let us suppose that such an operation is available and call it Concatenate. This takes two nodes as input and assembles them without creating a larger node, simply linking the inputs together into an ordered list. Any information that would have been associated with a new node is absent. Successive applications of Concatenate yield flat sequences of inputs in which groupings of adjacent inputs are irrelevant. To be useful, this operation needs to change the structure of the input in some way. We can simply assume that it serves the purpose of introducing a precedence relation to two nodes. Without introducing a relative order, Concatenate would be commutative, and its application to a set of n nodes would yield the same set, since neither order

nor grouping is relevant.

The number of possible distinct connected structures assembled from n items in a predefined order, using only Merge, is the $(n-1)$st number in the Catalan sequence $C_n = \frac{(2n)!}{(n+1)!n!}$, which starts out as 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, ... (Langendoen 1998). In contrast, using only Concatenate, only one possible structure can be produced: a flat list. The amount of information needed to distinguish among n possibilities is at least logarithmic with respect to n, or $\Omega(\log n)$, so to maintain a representation for n items assembled using Merge requires resources at least proportional to n, but only a constant amount for structures generated using Concatenate. As the size of a structure grows, it becomes increasingly more costly to maintain it if hierarchical information is needed.

Since hierarchical structure is a desirable but costly option, it would be advantageous for the linguistic processor to choose among alternative operations in order to minimize the resulting cost. Let us suppose that Concatenate is cheaper than Merge by virtue of not needing to construct a new node. If in one situation, Concatenate is sufficient, then there is no reason to use Merge since Concatenate is cheaper. If relative grouping is necessary, then the cost of Merge is justifiable since Concatenate fails to meet the requirements.

When might Concatenate be sufficient? This would be preferable in cases where phrases are similar in some fashion and the order in which successive pairs are connected does not matter. One linguistic example could be the phenomenon of coordination, in which phrases are connected using a conjunction, as in "The colors of the rainbow are red, orange, yellow, green, blue, indigo, and violet." Here the ordering of the individual colors is relevant, as a different order would not characterize a rainbow, but it is not necessary to group the colors into smaller sets. So in this example, it seems that avoiding Merge for the portion of the sentence listing individual colors would be more economical than using it.

Coordination is a fairly tricky phenomenon that exhibits a number of features absent from other syntactic structures and seems to violate rules that consistently apply elsewhere. Let us examine its characteristics in more detail in chapter 3.

2.3 Recursion and Iteration

As far as we know, the human species is unique among Earth's life forms in having the ability to develop and use language, a complex system of communication that assembles basic units of meaning and sounds together according to established rules. This form of communication meets the requirement of *discrete infinity*, the ability to produce arbitrarily long constructions that remain governed by underlying structural rules. One mechanism that allows this property to arise is *recursion*, in which structures of a given category may be nested within others of the same category to any arbitrary depth. The apparent uniqueness of this mechanism in distinguishing the complexity of human language from the systems of communication employed by animals has led Hauser, Chomsky and Fitch (2002) to propose that the human language faculty alone contains the ability to handle recursion, and that recursion alone distinguishes the capabilities of human language from those of animal communication. However, the hypothesis that recursion is a requirement for any human language is not an uncontroversial claim. Research over the past few decades—from findings in the Warlpiri language of Australia (Hale 1982) to more recent claims about the Pirahã language of South America (Everett 2005)[1]—has raised the question of whether recursion is indeed an essential component of a complete human language.

If we suppose the hypothesis to be false, we must then be able to account for the structural complexity of languages without recursion. However, even if we suppose the hypothesis to be true, we cannot exclude the possibility that natural languages employ—in addition to recursion—an alternate mechanism of generating complex structures.

Why might such a dual system evolve in the human language faculty? Suppose that the process of recursion, R, demands a considerable processing cost $C_R(x)$ that depends on the characteristics of a derivation x and that an alternative non-recursive process NR requires a different cost, $C_{NR}(x)$. For two derivations x and y, suppose that their characteristics are such that $C_R(x) < C_{NR}(x)$ and that $C_{NR}(y) < C_R(y)$.

[1]Though see Sauerland et al. (2009) and Nevins et al. (2009) for a refutation of Everett's claim.

Then assuming that derivations similar to x and y occur naturally in the course of human language evolution and that neither type of derivation is dispensable, a processing system that relies only on R or only on NR to perform all derivations will operate at a higher cost than one that employs R for derivations like x and NR for derivations like y.

Is there a measure by which recursion is more costly than an alternative non-recursive process? To answer this, we can observe that an immediate requirement of recursion is that the mechanism for extending the length of a sequence joins grouped structures into a hierarchical structure that increases in height as new structures are added to the existing structure. In particular, if we assume a binary tree representation of this hierarchy, then a structure of n elements must have a height of at least $\Theta(\log n)$. In effect, recursion is able to meet the criterion of discrete infinity, but at a cost: the structures it generates are unbounded with respect to their height.

So far, we have not related this abstract notion of "cost" to any actual resource. Now we may ask what practical consequences may arise from this necessary cost and whether there are alternative approaches that demonstrate an improvement over recursion. One possible limiting resource is short-term memory, which produces observable differences between people's ability to parse nested recursive structures and tail-recursive ones, a topic to be addressed in section 2.4.

As Miller and Chomsky (1963) have noted, since language must interact with the physical systems of the brain that handle production and logic, we expect that a limited resource will have an impact on the ability of the language faculty to function. If no such resource has any effect on the depth to which structures can grow, then the necessary cost of constructing deep levels of recursion is inconsequential, and we can conclude that recursion is the best mechanism that can satisfy the requirement of discrete infinity. On the other hand, if there is a resource that can limit the depth of an embedded structure, then we must examine alternative mechanisms to see whether they satisfy the condition of discrete infinity while presenting a lesser cost to the limiting resource.

One alternative would simply be that of joining structures in linear connections

without expanding the depth of the joined structure. This action could potentially generate arbitrary-length sequences by extending them horizontally. We define as *iteration* the process that joins a sequence of structures of the same category in this way. Iteration is thus analogous to recursion in that iteration is repetition generated by an operation of linear extension while recursion is repetition generated by an operation of embedding. An iterative structure of n elements would then be bounded only by a constant depth. This mechanism would thus offer an advantage over that of recursion by not placing a demand on any resources related to the depth of a structure.

A complete human language, however, must meet multiple criteria, and it is reasonable to imagine that some criteria may be more easily met by embedding than by linear extension, as a number of phenomena like c-command hold across nodes in a hierchical structure. As a result, we have at least two competing mechanisms, each of which offers advantages over the other. We may imagine that the human language faculty, as a system evolved toward a goal of optimizing the use of its available resources, makes use of both mechanisms in a way that takes advantage of the benefits provided by either mechanism.

2.4 Grammar and Processing

A key distinction to be made when discussing comparative complexities of linguistic structure is what characteristics should be assigned to a formal grammar that reflects the underlying structure of language and what characteristics should be assigned to the components of the language faculty that process the grammar and perform the computations that link sound and meaning. The latter components involve the functions of production—generating a structure from a set of atomic lexical elements and transforming the structure into an output stream—and parsing—receiving an input stream and constructing a corresponding structure. Both processes employ an underlying grammar to relate a phonetic stream and its corresponding semantic meaning.

A classic result from Chomsky (1956) proves that a finite-state grammar cannot generate arbitrarily nested recursive sentences like

(2.1) The mouse [the cat [the dog chased] bit] ran.

This example requires a dependency between the subject *the mouse [...]* and its predicate *ran*. However, the subject can be modified by an embedded clause that intercedes between the subject and the verb. This process can be continued indefinitely, allowing an arbitrary number of nested dependencies. A grammar that generates sentences of this type would need to keep track of a potentially unlimited number of states in order to decide whether all dependencies are met, and a finite-state grammar is not capable of this.

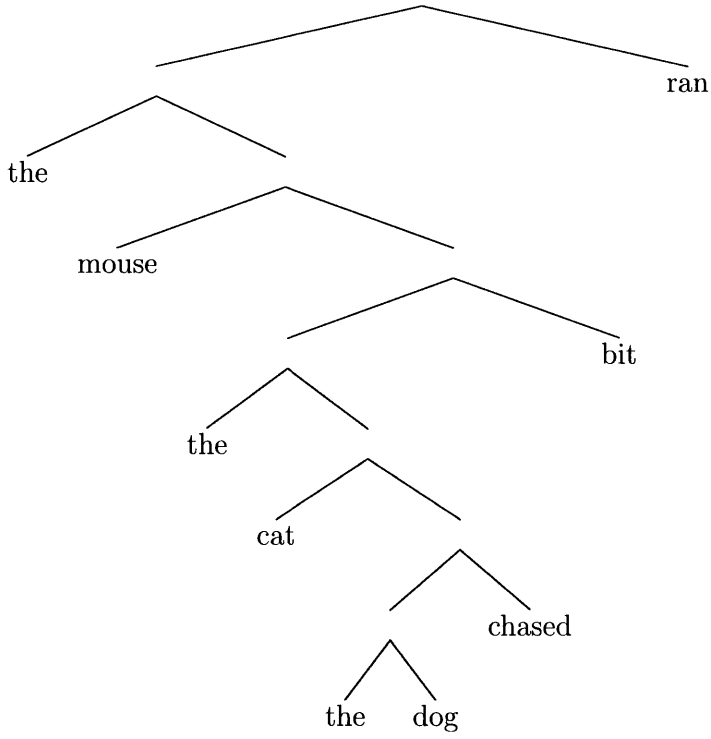
A sentence like (2.1), however, proves difficult or impossible for a human to parse. One may conclude from this that a human grammar is not actually capable of generating recursive structures to any arbitrary depth but that the grammar is a finite-state grammar after all, one with many states. Such a characterization may satisfy the experimental evidence, but it would require an exponential number of states, impractical for a natural system acquired with ease by children.

Alternatively, the entire language faculty may be characterized more simply as the interaction of a more complex grammar, such as a context-free grammar, and a processing component with limitations imposed by resources such as short-term memory, or a finite-state transducer. These limitations produce the difficulty observed with processing sentences like (2.1).

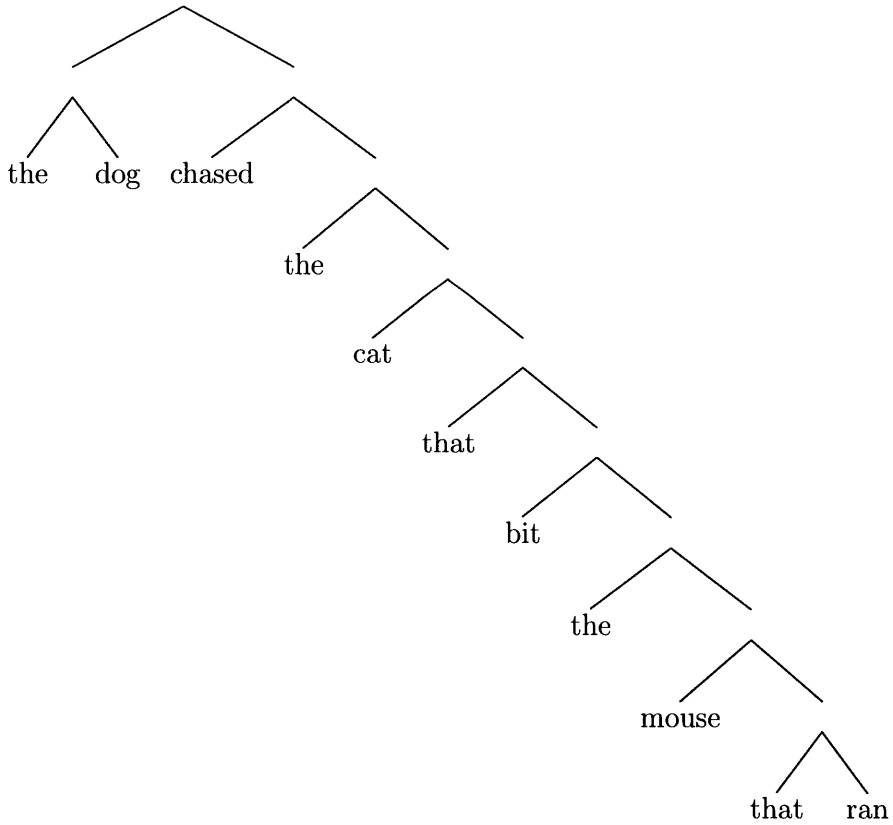
One may compare the complexity of (2.1) with an analogous tail-recursive structure like (2.2):

(2.2) The dog bit [the cat that chased [the mouse that ran]].

(2.2) is much easier to process than (2.1), and by drawing (simplified) trees to represent these structures, we can characterize the difference:


ran
the
mouse
bit
the
cat
chased
the
dog

(2.3)

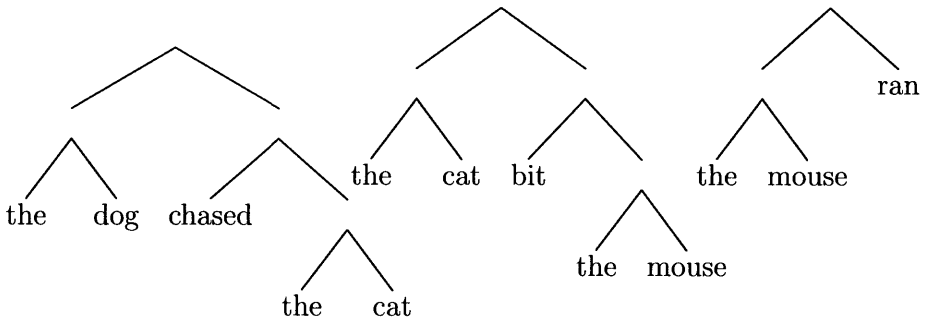

the
dog
chased
the
cat
that
bit
the
mouse
that
ran

(2.4)

In (2.3), dependencies between nodes can grow without bound, but all dependencies in (2.4) are local, limited by a constant number of intervening nodes. We may conclude that a constraint imposed by short-term memory limits the ability to process long-distance dependencies, producing the disparity observed.

However, both (2.3) and (2.4) share the characteristic of involving embedding to the same degree. If we suppose that the depth of a structure itself poses a considerable cost to a resource like short-term memory, as Langendoen (1975) notes, then even a structure like (2.4) will exceed the capacities of short-term memory, which has to store the entire depth of the embedded structure. We may wish to find a structure that limits the depth of an expanding structure rather than simply directing the expansion to one side. In short, we would like an ideal structure resembling the following:

(2.5)



In (2.5), the longest vertical path of successive edges is bounded by a constant, regardless of the number of additional clauses added. If we suppose that short-term memory only needs to handle the depth imposed by each disconnected clause individually, conserving its resources to process each one in turn, then there is no limit to how many clauses can be assembled together.

Langendoen (1975) proposes that while the underlying grammar of a language generates embedded structures of the type seen in (2.3) and (2.4), the processing component applies readjustment rules to produce intermediary structures like (2.5). These readjustment rules transform the underlying embedded structures into individual components like the ones in (2.5) so that short-term memory limitations apply not to an entire structure like (2.4) but to the intermediate structures produced by

the processing component of the language faculty. The readjustment rules in question cannot apply to a structure like (2.3) due to its long-distance dependencies, so the processor cannot conserve short-term memory by applying a rule. This would explain the difficulty of processing a sentence like (2.1) but not one like (2.2).

Do the readjustment rules then allow the grammar to generate embedded structures without any concern for their processing? We may conclude this only if the readjustment rules themselves place no demands on processing resources. Since the rules are an operation of the processing component of the language faculty, we cannot reasonably make such an assumption, so let us assign a cost to the readjustment rules themselves, C_{RR}. For simplicity, let us assume that this cost is the same for every individual readjustment of a structure.

To return to the question of whether human language structure should be characterized only by recursion, let us suppose that language is indeed characterized only by recursion and that every sequence of n clauses is generated by a grammar that constructs an underlying structure of n embedded clauses. To meet the constraints imposed by short-term memory, the processing component must then apply $\Theta(n)$ readjustment rules to flatten the entire structure, if the structure uses no nested embedding.

Ultimately, adding readjustment rules still requires a processing overhead that grows with the size of a structure. This overhead is limited at any point in time because rules can be distributed and spread out over a period of time, but the net result is that the processor must meet a necessary cost imposed by the readjustment rules.

Alternatively, let us suppose that the underlying grammar does not require completely embedded structures for every possible grammatical sequence of language, but that some structures may be reflected by underlying forms similar to (2.5): separate structures (themselves formed from embedding) that are linked together in a fashion other than embedding.[2] Given these underlying forms, and assuming that

[2]See Langendoen (2008) for a more recent approach that formally relates embedding structures to flat structures accessible to finite state transducers. Langendoen argues that bounds on center embedding follow from more natural bounds on zigzag embedding. Our approach differs, however,

a representative form does not have a depth of embedding in any of its constituent structures that exceeds the limitations of short-term memory, there is no reason to apply readjustment rules because the limitations enforced by short-term memory are not in danger of being violated. The total cost to the processor is thus reduced with respect to a tail-recursive structure like (2.4).

Is there experimental evidence that such a process is going on? As a simple example, let us suppose that conjunctions like *and* are a way of connecting separate structures without embedding. Then we would have, as an analogous structure to the embedded example (2.2), the following:

(2.6) The dog bit the cat, and the cat chased the mouse, and the mouse ran.

To test the processing requirements of (2.6) versus (2.2), let us extend each example into an infinite stream:

(2.7) The dog bit the cat [that chased the mouse that ran after the dog that bit the cat] ...

(2.8) The dog bit the cat, [and the cat chased the mouse, and the mouse ran after the dog, and the dog bit the cat,] ...

Given that readjustment rules demand an additional processing cost that can be spread out over time, we may ask what happens when we shorten the available time. The total processing cost would remain the same, but the processor is not permitted to spread the cost over a long period. The cost per unit time must increase to compensate. However, the limitations of short-term memory are themselves time-dependent since they arise from a limited capability of handling multiple items at once rather than multiple items at separate points in time. By compressing the time window in which a number of readjustment rules must take place, we must eventually force the readjustment cost to exceed the short-term memory limitations.

---

in that we do not allow any level of embedding among coordinands in a coordinate structure while Langendoen does allow coordinate embedding to a finite degree. Ultimately this may be a question of whether the information content of coordinate grouping has a syntactic representation; see section 5.7 for a discussion of logical hierarchy and flat syntactic structure. Langendoen also does not address the possibility of coordinating different categories, while we offer a tentative explanation for observed multiple-category coordinate constructions in section 5.7.

By this reasoning, we would expect that if someone tries to parse (2.7) as fast as possible, repeating the bracketed section indefinitely, the structure would unravel into incoherence very quickly. However, by our assumption that (2.8) imposes no requirement to apply readjustment rules, we would expect that someone trying to parse (2.8) at the same rate at which (2.7) becomes incoherent, (2.8) will still remain considerably coherent and will only become incoherent at a faster rate at which another processing resource common to the processing of both (2.7) and (2.8) becomes exhausted.

Psycholinguistic experiments may be devised to determine how well people perform on such a task and compare results for structures we propose to be grammatically flat versus those with embedding structure; see chapter 6 for a brief discussion of possible experimental approaches.

2.5 The Cost of Node Construction

In arguing for a grammatical representation of flat structure, we must distinguish between n-ary branching tree structures, where $n \geq 2$, and flat structures that do not have a node dominating a number of others. One can argue that this is merely a choice of representation, and that grammatically the two are equivalent. However, the two structures do have distinct characteristics that lead to distinguishable consequences in grammar.

In syntax, complex objects that are used in a derivation do not expose their entire contents to syntactic operations; rather, a complex object has a point of access which represents the object to operations that apply to it. Accordingly, we observe locality effects such as internal, unprojected features of a phrase remaining invisible to operations applied to the phrase.

In a tree, the most natural point of access is the root, a single node that is representative of the whole tree in some fashion, and allows the tree to form a part of a larger tree simply by making the root a node in the larger tree. A non-root node cannot be made a node in a larger tree without structural rearrangement.

In a flat list, on the other hand, there is no single node that is representative of the entire list, so it is not obvious how to isolate a single point of access. Structurally, there are two types of nodes within a flat list: endpoints and interior nodes. Interior nodes are connected to two other nodes each, but endpoints are only connected to one. The simplest way to form a flat list into part of a larger flat list is identify it with a node in the larger flat list. This does not change any of the connections of the interior nodes, but it introduces new connections to one or both of the endpoints. So we can think of flat lists as having two points of access for syntactic operations. Since the two endpoints are distinguished by their relative order, we may expect to find grammatical differences between operations that apply at the front of a flat list and ones that apply at the end of a flat list. In chapter 5, we suggest that this may explain certain peculiar features of coordinate constructions.

Chapter 3

Characteristics of Coordination

3.1 Basic Features

On the surface, a coordinate construction contains at least two phrases (*coordinands*) along with *coordinators*, particles or affixes that express a relationship among the phrases (Haspelmath 2004). *Syndetic* coordination occurs when at least one overt coordinator appears, while *asyndetic* coordination juxtaposes the coordinands without an overt coordinator, as in this example from Lavukaleve, a Papuan isolate spoken in the Solomon Islands:

(3.1) *nga-bakala* *nga-uia* *tula*
1SG.POSS-paddle(M) 1SG.POSS-knife(F) small.SG.F
"my paddle and my small knife" (Terrill 2004: 431)

When an overt coordinator appears in a coordination of two coordinands, it may appear once (*monosyndetic* coordination) or twice (*bisyndetic* coordination), as in these respective examples from Iraqw, a Southern Cushitic language of Tanzania, and Upper Kuskokwim Athabaskan, an Athabaskan language spoken in Alaska:

(3.2) *kwa/angw* *nee* *du'uma*
hare and leopard
"the hare and the leopard" (Mous 2004: 116)

(3.3) *dineje ʔɬ midzish ʔɬ*
moose with caribou with
"moose and caribou" (Kibrik 2004: 539)

More than two coordinands may appear, in which case bisyndetic coordination has one coordinator for every coordinand, and monosyndetic coordination one fewer. These analogous examples to (3.2) and (3.3) demonstrate coordination with multiple coordinands:

(3.4) *Kwermuhl, nee Tlawi, nee Dongobesh, nee Haydom nee Daudi*
Kwermuhl and Tlawi and Dongobesh and Haydom and Daudi
"Kwermuhl, Tlawi, Dongobesh, Haydom, and Daudi [places]"(Mous 2004: 113)

(3.5) *maladija ʔɬ jamena ʔɬ denk'a ʔɬ ɬeka mama? ʔɬ*
tent with stove with gun with dog food with
"a tent, a stove, a gun, and dog food" (Kibrik 2004: 539)

In monosyndetic coordination in many languages, omission of all coordinators except the last may occur, as in the English glosses of (3.4) and (3.5), but this option is not available in all languages, as in Hakha Lai clauses (Peterson and VanBik 2004). Omission of coordinators in bisyndetic coordination seems impossible in general (Haspelmath 2004).

Many languages that have monosyndetic coordination also allow the option of bisyndetic coordination for the same structures, as in French and Japanese, respectively:

(3.6) *Jean connaît et Paul et Michel.*
Jean knows and Paul and Michel
"Jean knows Paul and Michel." (Kayne 1994: 58)

(3.7) *John to Mary to ga kekkonsita.*
John and Mary and NOM married
"John and Mary married." (Kayne 1994: 58)

The coordinator is usually associated with one of its adjacent coordinands, but not both, and intonational phrasing can be used to demonstrate that a coordinator is associated with one of the adjacent coordinands. For example, in English, coordinators attach to the coordinands that follow them, which we can determine by separating the coordinands:

(3.8) John left, and he didn't even say goodbye.

(3.9) John left. And he didn't even say goodbye.

(3.10) *John left and. He didn't even say goodbye. (Ross 1986: 100)

This is an example of a *prepositive* coordinator. *Postpositive* coordinators attach to coordinands that precede them, as in Japanese:

(3.11) *remon iro-to miruku t'i*
lemon color-and milk tea
"lemon color and milk tea" (the title of a 2004 song by Morning Musume。)

3.2 Properties and Peculiarities

Coordinate structures exhibit a number of interesting properties, some of which differ greatly from structures that can easily be accommodated in a hierarchical framework.

One example is c-command, a relation that holds between a node and any child of its parent. As an example, in domains licensing polarity items, a negative polarity item (NPI) must be c-commanded by an NPI licensor while a positive polarity item (PPI) cannot be:

(3.12) *He chased nobody and/or any dogs.

(3.13) He chased nobody and no dogs.

(3.14) Nobody chased any dogs.

(3.15) *Nobody chased no dogs. (Progovac 1998: I/3)

In these examples from Standard English, the NPI *any* must be licensed by *nobody*, but the PPI *no* cannot appear in the domain of *nobody*. We can observe this distribution in (3.14) and (3.15). When a coordinate structure is introduced, however, as in (3.12) and (3.13), we find that there is no c-command relation between the coordinands, so that *any* and *no* are not licensed, resulting in *any* being unacceptable and *no* being acceptable.

This observation would be a problem for any model of coordination in which the first coordinand heads the larger coordinate phrase. As Hornstein (2009) argues, c-command arises naturally as a consequence of having primitive operations that combine to generate hierarchical structure, so c-command must be present in any structure that employs the same recursive combining operation that yields structures with typical c-command effects.

A common feature of coordinate structures is the similarity in category across coordinands. Coordinands usually must be of the "same" category, e.g. a noun phrase (NP) with another noun phrase or an adjective phrase (AP) with another adjective phrase, though a precise definition of "same" is unclear. This is known as the Coordination of Likes Constraint (CLC) (Chomsky 1957). However, the constraint appears to be violated in certain kinds of coordinations:

(3.16) You can depend on [my assistant] and [that he will be on time].

(3.17) *You can depend on that he will be on time. (Progovac 1998: I/5)

Here, a determiner phrase (DP) and a complementizer phrase (CP) are coordinated despite being of distinct categories. Furthermore, the ungrammaticality of (3.17) shows that the CP cannot appear by itself but is somehow acceptable within a coordinate structure. However, it cannot appear as the first coordinand:

(3.18) *You can depend on [that my assistant will be on time] and [his intelligence]. (Progovac 1998: I/4)

Another example of different categories being conjoined is the following, in which a DP predicate and an AP predicate are conjoined:

(3.19) Pat is [a Republican] and [proud of it]. (Sag et al. 1985: 117)

Coordinate structures also exhibit some unusual agreement characteristics. A conjunctive phrase of two singular conjuncts can agree with a plural verb, when either conjunct by itself would be unacceptable:

(3.20) A teacher and a student are/*is here discussing linguistics.

(3.21) A teacher *are/is here discussing linguistics.

(3.22) Two teachers are/*is here discussing linguistics.

However, the position of the verb with respect to the coordinate phrase reverses this effect:

(3.23) There *?are/is a teacher and a student here discussing linguistics.

(3.24) There *are/is a teacher here discussing linguistics.

(3.25) There are/*is two teachers here discussing linguistics.

The asymmetries with respect to relative verb position also appear in a number of other languages, as in Arabic:

(3.26) *el-walad* *we-l-banaat* *gataluu* *el-bisse*
the-boy and-the-girls killed-3PL/MASC the-cat
"The boy and the girls killed the cat."

(3.27) *el-banaat* *we-l-walad* *gataluu* *el-bisse*
the-girls and-the-boy killed-3PL/MASC the-cat

(3.28) *gatal* *el-walad* *we-l-banaat* *el-bisse*
killed-SG/MASC the-boy and-the-girls the-cat

(3.29) *gatalen* *el-banaat* *we-l-walad* *el-bisse*
killed-PL/FEM the-girls and-the-boy the-cat
(Progovac 1998: I/4)

Case mismatch also occurs in coordinate structures. Accusative Case may appear on a DP in subject position or nominative Case on a DP in object position; the expected Case of a DP may fail to be acceptable once it is conjoined with another, and Case may differ across coordinands in the same structure (Progovac 1998):

(3.30) A notice arrived for him and I.

(3.31) Me and her went to the store.

(3.32) *?I and she avoided going outside.

(3.33) She and him are best friends.

(3.34) Him and I found the solution.

This phenomenon occurs cross-linguistically and shows systematic patterns. Describing data from 32 languages, Johannessen (1998) demonstrates a correlation between verb-object order and the position of a deviant conjunct; out of 14 VO (verb before object) languages, all place the deviant conjunct in the second position, while 11 out of 12 OV (object before verb) languages place the deviant conjunct in the first position. The remaining 6 languages have mixed or unclear word order. This shows that in general case-checking is stronger between the verb and the closer conjunct. In coordinate constructions in subject position, however, the judgments are less clear. Progovac (1998) notes that judgments may differ across speakers, that the level of acceptability may be unclear even to a single speaker, and that Johannessen's analysis may be falsified by differing judgments.

Another parallel between verb-object word order and coordination is the attachment of coordinands. Progovac (1998) observes that the relative order of conjunctions and their associated conjuncts generally matches the relative order of verbs and objects, suggesting that coordinators occupy a head position in a phrase.

In addition to these differences, the Coordinate Structure Constraint (CSC) prohibits the movement of coordinands within a coordinate structure and movement of parts of coordinands outside of the coordinate structure (Ross 1986). An exception

to the constraint is Across-the-Board (ATB) movement, which allows extraction only when it takes place out of all coordinands:

(3.35) Which pancake did Airi prepare ___ and Koharu devour ___ ?

(3.36) *Which pancake did Airi prepare ___ and fry potatoes?

(3.37) *Which pancake did Airi fry potatoes and prepare ___ ?

Acceptable examples similar to (3.37) exist, such as in (3.38) and (3.39):

(3.38) How many pancakes can you eat ___ and still have an appetite?

(3.39) How many pancakes can you sit at the table and eat ___ ?

However, as Postal (1998) argues, examples like this one are not true coordination, but rather an example of modification by an adjunct. If we assume the CSC to hold with only the exception of ATB movement, then it would be preferable to have a theory that can account for this fact.

Chapter 4

Previous Analyses

4.1 Problems with Binary Branching

As argued by Kayne (1984) and subsequently adopted in Chomsky's (1995) formulation of Minimalism, binary branching arises as a consequence of a requirement for unambiguous paths to exist between any pair of distinct nodes. Kayne (1984: 132) defines a *path* as a sequence of adjacent nodes, and an *unambiguous path* in a tree T as a path $P = (A_0 \ldots A_i, A_{i+1}, \ldots A_n)$ such that

(4.1) $\forall i, 0 \leq i < n$:

(a) if A_i immediately dominates A_{i+1}, then A_i immediately dominates no node in T other than A_{i+1}, with the permissible exception of A_{i-1};

(b) if A_i is immediately dominated by A_{i+1}, then A_i is immediately dominated by no node in T other than A_{i+1}, with the permissible exception of A_{i-1}.

Informally, an unambiguous path is one that never has to choose among possible branches extending in the same direction. Since multiple upward branches are not possible in a binary tree, there is always an unambiguous path from a node to an ancestor of the node. In addition, immediate children of an ancestor are reachable by an unambiguous path, but only if the branching is binary, since having three or more

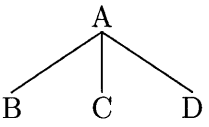
branches allows for more than one possible alternative at an ancestor node. Kayne treats this as a simpler reformulation of the c-command relation, and concludes that there can only be binary branching in phrase structure as a consequence.

While this argument is reasonable, it assumes that phrase structures exhibit a feature that trees must possess for this definition to apply: directionality. Without directionality, one cannot determine which of two adjacent nodes dominates the other. Hence the argument relies on an assumed property that may not actually characterize the structure of phrases.

If we remove the requirement of directionality, this allows two nodes to be adjacent without having one dominate the other. A sequence of adjacent nodes in which this property does not hold can be represented as a flat connected list.

Thus, while Kayne's argument holds for tree structures like (4.2), it does not hold for list structures like (4.3). In fact, as formulated, the unambiguous path constraint does hold between nodes B and D, since there is only one branch to choose from at any node.

(4.2)



(4.3) B—C—D

Following the definition strictly, however, we see that the existence of an unambiguous path follows vacuously since no node dominates another in this flat list. On the other hand, for every pair of adjacent nodes in a tree, one must dominate the other, so one of the two cases in the definition must hold. So how the concept of an unambiguous path applies to a flat list is unclear, and the apparent lack of c-command between coordinands suggests that there is no unambiguous path in the same sense as for a tree.[1]

Assuming that Kayne's argument holds for tree structures but is not sufficiently defined for flat lists, we can conclude that if the property of directionality is not

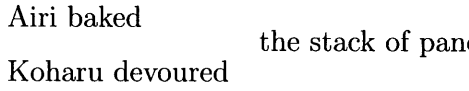
[1]See Progovac (1998) for a discussion of approaches arguing for and against c-command in coordinate structure.

assumed, phrase structure can exhibit binary-branching structures and flat structures, but not n-ary branching trees where $n > 2$.

4.2 Multidimensionality

A number of authors have treated phrase structure as exhibiting multidimensionality, having a third dimension by which nodes can be connected, independent of the dominance relation between a node and its parent and the adjacency relation between a node and its sibling. Goodall (1987) characterizes coordinate constructions as being a set union of phrase markers: a coordinate phrase is then an overlaying of one phrase on top of another and a merging of identical elements.[2] For example, a corresponding diagram for (4.4) would be (4.5).

(4.4) Airi prepared, and Koharu devoured, the stack of pancakes.

(4.5)


This analysis conveniently explains Right Node Raising and the lack of c-command between coordinands since they remain on separate levels, but it fails to sufficiently characterize the nature of the operation that overlays one coordinand on the other. An analysis by de Vries (2008) characterizes this operation as *behindance*, which assembles phrases along a third dimension; de Vries distinguishes two primitive operations: *dominance-Merge* and *behindance-Merge*. The former assembles hierarchies in which dominance-based conditions hold, and the latter performs an analogous operation but along an independent dimension immune to c-command. Each operation, however, is a form of Merge that connects two nodes into a larger node. Resulting structures in this formulation are still binary trees, but with two kinds of nodes: the type of each node determines the dimension along with its children extend.

This approach does raise some questions. It is not clear how this differs from dominance-Merge, or how properties like the CLC and CSC can be explained. If

[2]See also Lasnik and Kupin (1997). Fong and Berwick (1985) provide an implementation of these ideas in Prolog and further discussion.

situated within a Minimalist framework, questions arise concerning the economy of behindance-Merge: what economic considerations motivate the availability of behindance-Merge, and why is dominance-Merge unsuitable?

Furthermore, strict binary branching still exists in this model, and there are resulting problems outside of c-command.

Chapter 5

Primitive Computations

5.1 Beyond Merge and Move

The primitive Merge and Move operations of Chomsky (1995) have not gone unchallenged. A copy theory of movement has emerged in which the Move operation is regarded as involving an underlying primitive operation of Copy, which copies elements in the working domain so that they can be accessed in future stages of a derivation. Move would then result from applying Copy to a phrase in the working domain, then later on applying Merge to connect the copy to the larger structure that contains the original copied phrase (Nunes 2004).

Hornstein (2009) takes this basic idea a step further and breaks down Merge into more primitive elements: Concatenate and Label. As a result, the three primitives Copy, Concatenate, and Label combine to produce Merge and Move. Merge arises when Concatenate links two objects and Label combines the two into a larger object. Move occurs when Copy duplicates an object and Merge later combines the duplicate with a larger object that contains the original. An advantage of treating Merge and Move in this manner is that one can separate processes that appear in the cognitive systems of non-human animals from those unique to humans. The simpler the processes that make humans unique, the easier it is to explain how such an evolutionary leap could arise, since organisms evolve by developing often gradual changes that are retained across generations if beneficial.

Hornstein suggests that Copy and Concatenate are simple enough processes that may already appear in the communication systems of non-human animals, whose communication patterns exhibit features of connecting utterances in an ordered fashion (Concatenate) and duplication of utterances (Copy). Leaving the Label operation as the sole invention of the human language faculty, Hornstein suggests that this simple operation of constructing atomic units out of smaller ones emerged as an evolved trait specific to humans and is the basis for the complexity of language.

The nature of Label and what mental processes underlie it is a more challenging question. Concatenate is a simple enough operation; its only effect is to introduce order to a pair of previously unordered objects. So is Copy; its sole effect is to make a copy of the input object available for further derivations. What is more interesting is how Label operates. Labeling carries restrictions on what inputs can be operated on and depends on additional factors to determine the appropriate output. It is not clear that Label can operate so simply as to just project the head of its input Concatenated object; for one thing, what exactly constitutes the output is unclear, and if the theory is to correspond to actual processing operations, relevant to determining minimal computing costs. Another issue is what out of the input projects; the operation would be useless if it only projects the entire input, so some kind of selection mechanism is necessary.

Departing from Hornstein's approach, we allow the possibility that Label may be a class of primitive combining operators rather than a single unique one. These operators may apply in different circumstances, and they have differing costs that may make one preferable to another, depending on interface conditions that need to be satisfied.

5.2 Three Levels

In light of our earlier argument that flat lists are more useful in some circumstances, the simplest approach to deconstructing Label would be introducing two combining primitive operators, one that builds trees and one that builds flat lists. Concatenate

by itself already builds flat lists, but it cannot construct atomic units, so we need another operator that forms an atomic unit out of a Concatenated list while still retaining a flat structure. However, this kind of atomic unit must be distinguished from the one that results from the tree-building operation; otherwise it would be structurally equivalent to a multiple-branching tree. So we have a need for three distinct object types: an atomic unit with tree structure, an atomic unit with flat structure, and a list of atomic units.

Let us call the tree-building operation Project and the list-building operation Append, and let us refer to the tree-structure atomic units as *nodes*, the flat-structure atomic units as *frames*, and the non-atomic objects as *lists*.

More rigorously, we take nodes to be basic and define frames and lists in terms of nodes:

(5.1) **Node:** A syntactic object consisting of a collection of features.

(5.2) **Frame:** An ordered sequence of one or more nodes.

(5.3) **List:** An ordered sequence of one or more frames.

In the definitions for *frame* and *list*, the ordered sequence characterizes only the relationship among objects in the sequence; when there is only one object, the sequence is equivalent to the object itself. Hence a frame is a list of unit length, and a node is a frame of unit length and by extension also a list of unit length. In these terms, we no longer need a separate definition of *atomic*, as frames and only frames are atomic.

All lexical items are nodes, though new nodes can also be constructed during the course of a derivation by the operation Project. Frames and lists of more than unit length are constructed by Concatenate and Append.

We define the operations Concatenate, Append, and Project as follows. In our notation, we mark the boundaries of a node with parentheses $((\cdots))$ and the edges of a frame with square brackets $([\cdots])$. List boundaries are not explicitly marked, but unconnected lists are separated by commas. Adjacent nodes and frames are simply

placed next to each other in order. Since a node is a collection of features, we denote the features of a composite node as a subscript after the opening parenthesis. For example, if the node (XY) has the same features as X, we can notate this as $({}_{X}XY)$.

(5.4) **Concatenate:** Given two input lists X and Y, join them in order to produce a larger list. If $X = [x_1] \cdots [x_n]$ and $Y = [y_1] \cdots [y_m]$, where all x_i, y_j are frames, then

$$Concatenate(X, Y) \longrightarrow [x_1] \cdots [x_n][y_1] \cdots [y_m].$$

(5.5) **Append:** Given two input frames X and Y, join them in order to produce a larger frame. If $X = [(x_1) \cdots (x_n)]$ and $Y = [(y_1) \cdots (y_m)]$, where all x_i, y_i are nodes, then

$$Append(X, Y) \longrightarrow [(x_1) \cdots \ (x_n)(y_1) \cdots \ (y_m)].$$

(5.6) **Project:** Given two input nodes X and Y, join them in order to produce a larger node. If X and Y are nodes, then

$$Project(X, Y) \longrightarrow (XY).$$

Parallelism abounds in these definitions, and indeed, starting from our definitions of frames and lists, we can see that Append is simply a species of Concatenate (since all frames are lists), and Project is simply a species of Append (since all nodes are frames). To avoid reduplication of tasks, we can then reason that Append has Concatenate as a component, to connect lists together, along with any additional frame-specific computations that need to take place to connect frames together. Hence the application of Append involves an application of Concatenate, along with possible additional computations. Likewise, Project incorporates Append, so it makes use of Append to connect frames together, and then applies additional computations that need to take place to create a new node. We will discuss the nature of these additional computations shortly.

We assume that these operations consume objects in the working domain and introduce new objects to the working domain. However, the inputs to an operation are not necessarily maximally connected objects. If Project acts on two nodes, and the nodes are part of larger frames and lists, then the Append and Concatenate instances within Project operate on frames and lists (the two single nodes themselves) that are connected to other nodes. We therefore allow Append and Concatenate to operate on non-maximal frames and lists, as long as the inputs are prefixes and suffixes of any larger objects they are part of; specifically, if X and Y are inputs to Concatenate or Append, then X is not connected to anything to its right, and Y is not connected to anything to its left.[1]

Taken together, Concatenate, Append, and Project can be regarded as a general operation Merge that can apply to different data types, with more restricted behavior arising for more specific subtypes. This is somewhat analogous to class inheritance and method overriding in object-oriented programming.

(5.7) **Merge:** Given two input objects X and Y of type Z, join them in order to produce a larger object of type Z.

This can be viewed as a three-level analysis of Merge. Hornstein's proposal, which lacks Append, can be considered a two-level analysis. Let us examine properties that arise from the different kinds of structures, and check whether a three-level approach is viable. To do so, we need to distinguish Concatenate from Append, and Append from Project.

[1]If we relax this assumption, then a frame is no longer a totally ordered sequence of nodes, but only a partially ordered connected set of nodes, and likewise for a list. While we do not explore this possibility in the current study, it may be an appropriate direction in which to proceed, one that may explain phenomena which appear to involve constituents that are disconnected on the surface, as in flexibility in adjunct grouping (Hornstein 2009) and Right-Node Raising (Bachrach and Katzir 2009).

5.3 Concatenate vs. Append vs. Project

As formulated, there are no constraints on Concatenate. Any two lists can Concatenate to form a bigger list.[2] However, the result may or may not be grammatical or anywhere in the stages of becoming grammatical. We need an operation that can make a Concatenated list be useful. In the current approach, the available primitive operations are a three-level Merge and Copy. The former can apply to all data types, so it cannot differentiate among them. This leaves Copy.

Since Copy is used to duplicate an object for future use in a derivation, constraints introduced by Copy may select between Concatenated lists that can be Copied and ones that cannot. What characterizes the input to Copy? If X is an object in the working domain, and Copy's output is a copy of X, then the input cannot be a copy of X; otherwise, we would need another operation that does the same thing as Copy, and we have the same problem. So the input to Copy must be a piece of information that is not itself a copy of X, but can be used to construct a copy of X. If we assume that Copy has the ability to scan objects in the working domain, then it must be able to know when to start and stop copying. We can give Copy the first and last nodes and have it check successive nodes that it comes across, but the possibility of identical nodes makes this impractical. We can also specify exactly how many nodes to copy, but this information must be updated every time lists are extended, and as far as we are aware, no other part of grammar employs a counting operation. Alternatively, a simple and efficient solution would be to introduce markers into a list to identify contiguous sublists that are suitable for copying, assuming that Copy may impose its own constraints on what may or may not be copied. This results in a partitioning of lists into contiguous groupings, which we can distinguish as frames. Since Copy can

[2]In Hornstein's (2009) formulation, Concatenate operates on atomic units and produces a non-atomic list of units. Since in our terms a node is by definition a list, and a Concatenation of two nodes is the same as the Concatenation of any lists the nodes are in, the two formulations are equivalent. We can alternatively formulate Append and Concatenate as taking input *nodes* and producing frames and lists, but we favor the current approach on the grounds that the three-level construction of Merge allows the operations at each level to apply to all objects at that level rather than just nodes; any action that applies to nodes specifically cannot be a part of Concatenate or Append. This approach places stronger restrictions on what Append and Concatenate can do, and makes it easier to distinguish among the three levels.

only copy frames, not lists in general, we need an operation to construct composite frames, which we call Append.

The precise nature of frame markers is unclear, but there are two simple and equivalent characterizations. One possibility is to make frame markers denote the edges of a frame. Thus whenever a node is introduced into the working domain, it comes with enclosing frame markers. Select and Project must add frame markers as part of the process of introducing a new single-node frame, and Append joins existing frames by removing frame markers between them so that Copy does not stop between the old frames. The other possibility is to characterize frame markers as "bridges" between adjacent nodes that tell Copy to keep going until it reaches a node without an outgoing bridge. These two characterizations are equivalent because the connection between every pair of nodes in a list is either a separation between frames or a connection between nodes in the same frame. We can choose either type of connection to mark; the other follows from the absence of a marker. Since it is simpler to introduce an object only when it is needed rather than introducing objects and then deleting them, we adopt the bridge characterization.[3]

As a duplicating operation, Copy needs to allocate memory to correspond to each node being copied. If we imagine that nodes come in different shapes and sizes, as reflected in their features, then it may be more efficient to copy a group of similar items than a group of dissimilar ones. Using shape as an analogy, suppose that the copying machinery can accommodate different shapes, but it has different settings for different shapes, so that it can copy a group of triangles quickly rather than having to readjust if the successive inputs are an eclectic mix of triangles, squares, and circles. We can suppose that some features, such as shape, describe characteristics that allow Copy to operate efficiently, and that other features, such as color, characterize differences among objects with the same shape; these features are copied over to the output object corresponding to each input object. Following Chomsky (1995), we distinguish *categorial features* like shape and *ϕ-features* like color, though with the

[3]The notation we have adopted marks frame edges rather than bridges, but this is only for notational convenience, not a reflection of actual representation.

caveat that this classification is only for the purposes of Copy and may not relate to other classifications of features. So the ideal input to Copy would be a sequence of nodes with the same categorial features and possibly different ϕ-features.

These constraints on Copy make it advantageous for the operation that assembles frames to check whether the nodes in each frame have the same categorial features. If we assume that this notion of sameness constitutes an equivalence class among sets of categorial features, then the property of transitivity allows Append to check matching categorial features only between adjacent nodes. If all pairs of adjacent nodes in a frame share the same categorial features, then by transitivity, all nodes in the frame do. This property allows Append to operate on two input frames in general without compromising efficiency to check all nodes in the two frames. If transitivity did not exist, then this kind of feature checking cannot be generalized to input frames and must be restricted to input nodes; such a restriction violates the input conditions on Append, so we conclude that non-transitive feature-checking cannot be a function of Append.

Adjacency is not a transitive relation. Any relation that requires adjacency would motivate the inclusion of a third operation beyond Concatenate and Append. If there is a binary relation R such that for two nodes X and Y, $R(X, Y)$ must hold to yield a convergent derivation, and R is true only if X and Y are adjacent, then R cannot be computed by Append, but only by an operation that operates at the level of individual nodes. We thus turn to feature checking between individual nodes. It is well known that adjacent nodes can satisfy feature-based relations. For example, the transitive verb *find* requires an adjacent DP as an argument. We can state this criterion as a relation between *find* and any adjacent node with the right features.

To compute this relation, we need an operation that applies to individual nodes. We call this operation Project and allow it to compare features between two adjacent nodes and construct a new node whose features are determined by a function of the features of the input nodes.

Constraints on Copy and the need to compute non-transitive relations thus neces-

sitate the inclusion of at least three levels of syntactic operations.[4]

5.4 Processing Costs

With a three-level construction of Merge, where a joining operation at each level is a specification of joining operations at higher levels (regarding the list level as the highest), it is reasonable to separate computations into three levels, where computations that take place at higher levels are used as part of operations that take place at lower levels. This avoids unnecessary reduplication of tasks and allows more complex operations to build upon simpler ones. For example, Concatenate links lists together so that Append does not have to do so separately in order to check categorial feature matching, and Append accesses at least some features so that Project does not have to access them independently.[5]

This naturally makes Project more costly than Append, as it involves computations that cannot take place at the frame level, in addition to processes that form part of Append; likewise, Append becomes more costly than Concatenate. We may expect that the human language faculty, as a computational system that prefers efficient operations, to use Append only when Concatenate fails to meet necessary requirements, and to use Project only when Append is not sufficient.

We can formulate the three operations algorithmically as follows:

(5.8) CONCATENATE(X, Y):

1. If X is not connected to any object to its right and Y is not connected to any object to its left:
2. Join X and Y into the same list by connecting the rightmost node of X to the leftmost node of Y.
3. Else:

[4]While we allow the possibility of four or more levels, we do not pursue such an analysis here.

[5]Accessing features may impose a considerable cost if the encoding of features in labels is not the representation that Project and Append work with, so that features must be decoded from labels. Perhaps labels store features in a compressed form, and they have to be decompressed for Project and Append to use them. We merely raise this point as a possibility without addressing it further.

4. Return a Concatenate error.

(5.9) APPEND(X, Y):

1. If CONCATENATE(X, Y) succeeds:
2. If the categorial features of the rightmost node of X and the leftmost node of Y are equivalent:
3. Join X and Y into the same frame by inserting a bridge between the rightmost node of X and the leftmost node of Y.
4. Else:
5. Return an Append error.
6. Else:
7. Return a Concatenate error.

(5.10) PROJECT(X, Y):

1. If APPEND(X, Y) does not return a Concatenate error:
2. If the features of X and Y satisfy selection conditions:
3. Create a new node Z to dominate X and Y. The label of Z is a function of the labels of X and Y.
4. Else:
5. Return a Project error.
6. Else:
7. Return a Concatenate error.

While all three operations, if successful, result in a Concatenation of nodes, Project and Append introduce additional structure not found if Concatenate applies by itself.

We note that Append does not have to succeed in order for Project to use it. Append applies in order to construct a single frame from its inputs, and this characterizes its success. If it fails to construct a single frame, however, Project can construct a

new node, which by definition is a new frame, so the output of Project also meets the requirements for Copy. The difference is that the new frame is not formed from the joining of existing frames, but by the creation of a new node that dominates two single nodes within existing frames; other nodes in those existing frames do not directly participate in Project, so we can predict that structures exist in language which satisfy categorial equivalence conditions imposed by Append but not other conditions imposed by Project. We will try to explain some of the peculiarities of coordination as a consequence of having three levels of types and operations.

In the definition of Project, the function that constructs a new label is not defined. The details of such a function are outside the scope of this study, but we allow that such a function need not "project" the entire label of one of the two input nodes to form the label of the new node. It may assemble a new label from features present in both nodes, and it may even construct features not present in either of the nodes. The last possibility may be ruled out as being inefficient since introducing new information not present in the inputs can conceivably be more costly than copying existing information. We do not pursue this possibility, but we do allow the construction of composite labels from features of both input nodes as a simple explanation for coordinators.

5.5 Coordination as a Product of Append

We argue that in its most basic form, coordination is the application of Append to coordinands to form frames. Since coordinate structures can move, they must be frames rather than lists, due to constraints imposed by Copy. While this condition can be met by either Project or Append, Project is the more expensive operation and should be avoided if possible.

Under the current analysis, Project can operate to serve one of two purposes. If Append fails, then Project can serve as a backup since it too constructs a frame. The other purpose is to check features, following Chomsky (1995)'s Full Interpretation principle, in which uninterpretable features must be checked during a derivation and

unchecked uninterpretable features at the interfaces cause a derivation to crash. Thus, if uninterpretable features are present, Project can serve the purpose of checking them, even if Append does not fail.[6]

Since in general coordinands exhibit more categorial similarity than head-argument pairs, we may suppose that Append succeeds, and Project, if it occurs, must check features. The lack of c-command across coordinands, however, suggests a configuration of coordinands distinct from hierarchical structures where phrases move to higher positions to check their features. We conclude from the absence of expected consequences of Project that Project does not actually apply in joining coordinands.

Coordinate structures, however, do feature coordinators that appear between coordinands and exhibit asymmetry in that they are primarily associated with one coordinand each rather than multiple coordinands. Since cross-linguistically a coordinator-coordinand pair has the same order as a verb-object pair, the coordinator appears to behave as a head taking the coordinand as a complement. At the same time, the categorial features of the coordinand are accessible outside of the coordinator-coordinand pair and are used by Append to join coordinands. So the result of joining a coordinator and a coordinand manifests features of both components.

If we allow Project to construct a label from the labels of both input nodes, then it seems reasonable that Project would apply to join a coordinator and its associated coordinand. The need to use features from the complement suggests that the head does not carry all the features needed for Append to apply. We can think of coordinators, then, as a kind of "partial head" whose features form part of, but not all of, the label of a node constructed by Project.

Since Append compares categorial features, we can suppose that each coordinator-coordinand complex has categorial features. These seem to arise from the coordinand rather than the coordinator since in many languages, the same coordinator can be used to coordinate a number of types; for example, the English *and* can conjoin two noun phrases, two verb phrases, two prepositional phrases, and so on.

[6]This possibility raises interesting questions as to whether any empirical constructions are of this form.

While this is a cross-linguistic trend, it is by no means universal. In Japanese, for example, the conjunction *to* ("and") can conjoin noun phrases but not verb phrases. Thus some categorial selection must take place.

There are two levels to distinguish here. One is the level at which a coordinator and a coordinand Project. The other is the level at which the output of Project can undergo Append. Both levels can perform some kind of categorial selection. Distinguishing these two levels can lead to a simple explanation for examples in which coordinands of seemingly different categories are coordinated. We will address this in section 5.7, along with other curious phenomena like apparent case and agreement violations. But first let us give examples of the different kinds of structures that can arise from the interactions between Append and Project.

5.6 Examples

We now give examples of types of structures that can arise under a three-level conception of Merge. At one extreme, we have structures with absolute binary branching that arise out of repeated applications of Project. At the other end, we have completely flat structures that arise out of repeated applications of Append. In between are examples that make use of both operations. We derive examples according to our framework, and identify consequences in terms of processing costs.

As an example of pure binary branching, let us consider a sequence of n terminals (5.11), and examine how applications of Project can construct a complex structure, specifically argument structure. The initial input is the numeration, an unordered set of n lexical items; Project combines these into a tail-recursive nesting of arguments.

(5.11) History of philosophy of history of philosophy of ... of history of philosophy.

The meaning of this construction is an academic discipline whose main concern is the history of another academic discipline, which itself has a similarly recursive definition.[7] The derivation proceeds as follows, selecting successive lexical items starting

[7]Other interpretations of (5.11) are possible, such as a discipline whose main concern is the

from the right.[8]

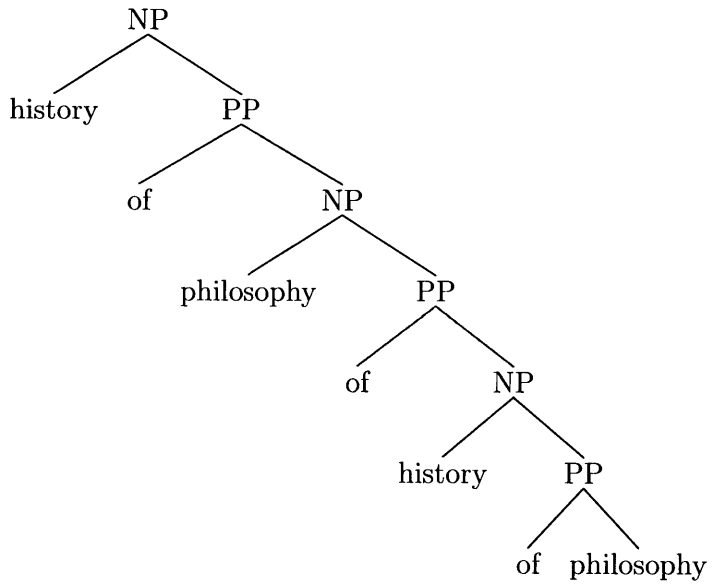
(5.12) Derivation of (5.11):

1. Select *philosophy* (the rightmost instance).
2. Select *of*.
3. Merge *of* and *philosophy*. Since *of* must check selection features, Project constructs a new node dominating *of* and *philosophy*. The features of *of* form the label of the new node.
4. Select *history*.
5. Merge *history* and *of philosophy*. The node *of philosophy* must be an argument of *history*, not an adjunct, so Project once again applies to satisfy the selection criteria.
6. Perform steps 2–5 in an analogous fashion to construct *philosophy of history of philosophy*.
7. Repeat steps 2–6.

This process continues until all lexical items have been incorporated. The result is a tree resembling the following (for $n = 7$ terminals):

"history of philosophy" of another discipline. We adopt the strictly tail-recursive interpretation for explanatory purposes, to illustrate how argument structure can arise from the available operations.

[8]While it is also possible to select lexical items starting from the left, the successive applications of Merge must happen in right-to-left order because the rightmost terminals are the most deeply nested. Selecting lexical items from the left leaves n leaves by themselves at the point at which the first Merge operation occurs, and it may be less efficient for the processor to maintain a set of unconnected objects instead of selecting objects from the numeration as needed.



(5.13)

At the other extreme, consider a similar construction in which *and* substitutes for *of*:

(5.14) History and philosophy and history and philosophy and ... and history and philosophy.

A plausible context for this construction would be a speaker unpacking a box full of history and philosophy books and identifying the genre of each book as it emerges.

The derivation proceeds as follows, selecting nouns in left-to-right order and selecting each instance of *and* immediately following the noun it precedes.[9]

(5.15) Derivation of (5.14):

1. Select *history* (the leftmost instance).
2. Select *philosophy.*
3. Select *and.*
4. Merge *and* and *philosophy.* Since *and* must check selection features, Project constructs a new node dominating *and* and *philosophy.* A

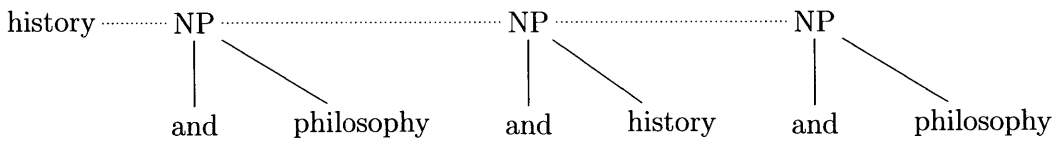
[9]We characterize the derivation as proceeding from left to right rather from the bottom up, but the Append operations can be applied in any order. We choose the left-to-right order as a convenience since any parser of a construction like (5.14) receives the input in that order. While we might wish to do the same for (5.11), Project must proceed from the bottom up to construct the requisite hierarchy, in (5.12).

combination of features from *and* and *philosophy* form the label of the new node.

5. Merge *history* and *and philosophy*. Since the categorial features are equivalent, and no selection features need to be checked, Append introduces a bridge between *history* and *and philosophy*, thus joining their frames.
6. Perform steps 2–5 in an analogous fashion to construct *history and philosophy and history*.
7. Repeat steps 2–6.

The resulting structure is not a tree but a single frame connecting the coordinands together. Each coordinator-coordinand pair forms a two-leaf tree within the frame. We can visualize the structure as the following graph (for $n = 7$ terminals):

(5.16)



Comparing the costs of these two types of extensions, we find that the completely nested structure of n leaves derived through Project has a processing cost of $(n-1)C_P$, where C_P is the cost of applying Project once. In addition, the height of the resulting tree is $\Theta(n)$. On the other hand, the flat structure of n leaves derived through Append has a processing cost of $(C_P + C_A)(n-1)/2$, where C_A is the cost of applying Append once. Since $C_A < C_P$, we have that the flat structure has a lower processing cost.

It is also possible to construct flat structures without coordinators, such as an enumeration of terms, which would have an even lower processing cost, but how this is distinguished from ordinary coordination is a matter for future inquiry.

5.7 Consequences

Our formulation of Merge as a three-level operation leads to natural explanations for a number of observed coordination phenomena.

The Coordination of Likes Constraint follows from conditions on the application of Append. While we have not established a direct mapping between categorial features as compared by Append and categories of phrases that are coordinated in coordinate constructions, some form of a mapping seems to exist. While the details are not entirely clear, the general categorial similarity of conjoined phrases suggests that it is compatible with and is a logical result of categorial equivalence conditions imposed by Append.

In a coordinate structure, there are two levels at which categorial selection takes place. The first is selection of a coordinand by a coordinator head. The second is the Appending of adjacent nodes. We can distinguish these two levels by introducing coordinands that are not of the same category, for example a determiner phrase and a complementizer phrase in these examples copied from (3.16) and (3.17):

(5.17) You can depend on [my assistant] and [that he will be on time].

(5.18) *You can depend on that he will be on time. (Progovac 1998: I/5)

The unacceptability of (5.18) shows that the preposition *on* in this sense cannot select a CP as its complement. In (5.18), however, it is not a preposition that is selecting the CP as its complement; it is the conjunction *and*. This observation can be explained by allowing *and* to select for a CP as a complement but project the same categorial features as a DP in order for Append to take place. Thus *and* possesses at least some categorial features, though not necessarily all the categorial features of a DP. At a minimum, it must have all categorial features of a DP that are not present in a CP.

In addition, the Coordinate Structure Constraint follows naturally from input constraints on Copy: since Copy must copy a frame, and it does so by duplicating all consecutive nodes connected by bridges, a non-maximal sequence of nodes within a frame cannot be copied. Hence no proper subsequence of a frame can be

moved. Across-the-Board movement is not movement directly out of a frame, but out of constituent nodes within a frame. This suggests that frame-like properties may hold across parts of nodes in adjacent frames. The exact mechanism remains to be explained, but returning to the shapes analogy, if we suppose that triangles can be joined in a frame, then the memory allocated to hold triangle representations may also allocate memory to hold representations of triangle corners and sides in the same configurations; the parts of memory holding information about analogous corners may align and form a frame-within-a-frame that may be readable by Copy and thus participate in movement.

Apparent mismatches in agreement and case also naturally arise from this line of analysis. Consider the distribution of *is* and *are* in the following examples:

(5.19) There is a cake and a pie in the fridge.

(5.20) *There are a cake and a pie in the fridge.

(5.21) *A cake and a pie is in the fridge.

(5.22) A cake and a pie are in the fridge.

We assume that *a cake and a pie* originates after the copula and optionally rises to satisfy the EPP condition, which may also take the expletive *there* instead. We assume features need to be checked in both the original position and the position to which *a cake and a pie* can move. For simplicity purposes, we collapse everything between the two positions, including tense and other functional heads, into a single verb head, and ignore the PP *in the fridge.*

The coordinate phrase *a cake and a pie* is constructed by applying Project to *a* and *cake*, yielding a DP, and to *a* and *pie*, yielding another DP. Project applies again to combine *and* with *a pie*, producing an object with the categorial features of a DP. Append then connects *a cake* and *and a pie.* The resulting structure can be represented as

(5.23) $[(_{DP,-Pl}(_{DP,-Pl}\text{a})(_{NP}\text{cake}))(_{DP,+Pl}(_{+Pl}\text{and})(_{DP,-Pl}(_{DP,-Pl}\text{a})(_{NP}\text{cake})))]$

Here we suppose that *and* has a [+ Plural] feature that allows a coordinate structure to have plural agreement in some settings. When *and* combines with *a pie* under Project, the [- Plural] feature of *a pie* is overridden by the [+ Plural] feature of *and*. Number is not a categorial feature, so Append can conjoin *a cake* and *and a pie* despite their differing plurality features. The result is a frame with two DP nodes: the first node is singular, and the second is plural.

After the coordinate structure is constructed, it can participate in subsequent operations. We now need to merge the preceding copula with the *a cake and a pie*. Since selection features need to be checked, we need to apply Project. Since Project only operates on individual nodes, it is applied to *is/are* and *a cake* (not the entire *a cake and a pie*). Since *is* selects for a DP with a [- Plural] feature, this operation succeeds. On the other hand, *are* selects for a [+ Plural] feature, so it fails. If the derivation stops here and fills the frontal position with *there*, then there are still unchecked features, so (5.20) crashes, and we have an unacceptable derivation. In (5.19), however, the analogous features are checked, so the derivation converges.

Now suppose that instead of using *there*, we move *a cake and a pie* to the front. To satisfy feature selection here, we must apply Project, which again operates on individual nodes. Thus, due to the change in relative position, it combines *and a pie* and *is/are*. Since *and a pie* is plural, Project succeeds with *are* but not *is*. If the derivation stops here, we have that (5.21) is unacceptable while (5.22) is acceptable.

Our theoretical framework leads naturally to a prediction that concurs with observed empirical distributions.

Case mismatch can be analyzed in a similar manner. Consider these examples in some varieties of informal English:

(5.24) Him and her ate some cake.

(5.25) ?Him and she ate some cake.

(5.26) He and her ate some cake.

(5.27) He and she ate some cake.

(5.28) A present arrived for him and her.

(5.29) ??A present arrived for him and she.

(5.30) *A present arrived for he and her.

(5.31) *A present arrived for he and she.

While my judgments on these are not completely clear, (5.24) through (5.29) are definitely much better than (5.30) and (5.31).[10]

If we take (5.24) to be grammatical, then nominative case is needed in the last node of the coordinate frame. Since *her* has accusative case, we suppose that *and* has nominative case that overrides it.

A similar argument to the earlier analysis for number features holds for these examples as well. (5.30) and (5.31) crash because *for* selects for accusative case and gets nominative case in *he* instead. (5.28) and (5.29) are both acceptable because *for* only checks features with *him*. In (5.25) and (5.27), *she* has nominative case which is not overridden in the construction of *and she*, so these pass. In (5.24) and (5.26), *her* has accusative case, but it is overridden by the nominative case of *and*, making these acceptable as well.

Why *and* has nominative case is unclear, but this assumption is consistent with the results, at least in my judgment. If another speaker's *and* does not have nominative case, and Project simply raises the case of *and*'s complement, we would expect that (5.24) and (5.25) fail to converge, a pattern that is similar to the distribution of analogous constructions with *or*, in my judgment:

(5.32) *?Him or her ate some cake.

(5.33) ?Him or she ate some cake.

[10]Different judgments are possible and have been attested. We attribute this to natural variation among speakers of a language, and predict that for speakers whose *and* lacks a case feature, a distribution similar to (5.32) through (5.39) would be observed, and that for speakers whose *and* has accusative case, only (5.28) and (5.29) would be acceptable, with (5.29) possibly appearing questionable, perhaps for independent reasons, as in my judgment. Additional factors may be involved that rule out otherwise acceptable derivations, or force derivations that crash to appear acceptable on some level; for example, the construction "I and my friend" seems to be unacceptable in any configuration.

(5.34) *He or her ate some cake.

(5.35) He or she ate some cake.

(5.36) A present arrived for him or her.

(5.37) ??A present arrived for him or she.

(5.38) *A present arrived for he or her.

(5.39) *A present arrived for he or she.

This distribution is consistent with an assumption that *or* has no case feature.

We tentatively conclude that our theoretical framework makes predictions that correspond with a variety of empirical characteristics of coordinate constructions.

A possible problem with our current approach, however, is its failure to account for logical hierarchy in coordinate structure. For example, a coordinate structure may have several logical groupings of coordinands:

(5.40) (A cake and a pie) and (a pancake and a waffle).

(5.41) A cake and (a pie and a pancake and a waffle).

(5.42) A cake and (a pie and a pancake) and a waffle.

In these examples, the bracketing could indicate the layout of these items on a table, with items in the same grouping lying closer to each other. These structures can be distinguished by prosodic information like intonation and spacing (Steedman 2000; Wagner 2009):

(5.43) A cake and a pie | and a pancake and a waffle.

(5.44) A cake | and a pie and a pancake and a waffle.

(5.45) A cake | and a pie and a pancake | and a waffle.

If we require that prosodic information have a syntactic representation, then our current approach fails to capture this prosodic information without further introducing syntactic objects that correspond to prosodic units. However, it may not be necessary to connect prosody and semantics through syntax. One can alternatively suppose that prosody and semantics are connected directly, as in Selkirk's (1984) proposal that a level of intonation structure connects phonetic structure and information structure separately from syntax, or Steedman's (2000) proposal that the appearance of a level of syntactic structure emerges only as a by-product of a processor mapping between phonetic and logical form. If so, a characterization of syntactic structure need not suppose that all relations between logical and phonetic form have a syntactic representation.

5.8 Three Merges or Three Nested Levels?

Our conception of a tripartite Merge bears some similarity to a proposal by Langendoen (2003), who divides Merge into three separate operations: Set Merge, List Merge, and Pair Merge, building upon Chomsky's (2001) distinguishing between Set Merge and Pair Merge operations. Ranking the three operations in order of "simplicity, elegance, and economy"—from greatest to least: Set Merge, List Merge, and Pair Merge—Langendoen argues that higher-ranked operations are employed unless they are inadequate. By analogy, we can suppose that our Concatenate, Append, and Project resemble set merge, list merge, and pair merge, respectively.

The two accounts differ, however, in that Langendoen argues that coordinate structure arises from Pair Merge applied both between a coordinator and its associated coordinand and between the resulting complex and additional coordinands, while in our approach, we use Project to produce the coordinator-coordinand complex and Append to connect the resulting complex and other complexes or stand-alone coordinands. Langendoen (2003) does not provide an explanation of how peculiar characteristics of coordination can be accounted for, so it is difficult to see how employing Pair Merge explains these empirical observations. Also, by employing Pair Merge for

coordinate structure and the simpler List Merge for pure argument structure, Langendoen's analysis predicts that the former is more difficult to process, while in our approach, the latter should be more difficult. Future experimental testing may be able to distinguish the relative applicability of these two approaches.

Furthermore, Langendoen's analysis treats the three operations as distinct alternatives, while we argue that simpler operations form subroutines of more complex ones. In our formulation, each operation applies to a separate syntactic data type, yielding the same data type, and the data types are related to each other in a class membership relation, while in Langendoen's analysis, it is not clear how the output of the three operations can be used for subsequent operations if they are of different types.

In addition, all three of our operations introduce order to objects that may in general be unordered, while Set Merge does not introduce an order, but rather a grouping among objects. Thus, over the objects on which they operate, Concatenate is associative but not commutative, while Set Merge is commutative but not associative. We may ask which of the two operations is preferable to have as part of a linguistic computational system.

While this question merits a deeper exploration, it is worth noting that if individual objects are associated with distinct spatial regions in a physical device that can store information, then the spatial arrangement of the objects' representations implies at least a partial order. Suppose we want to represent sets in a limited amount of memory, and we assign physical locations to two objects A and B. If we switch the locations of A and B, this results in a distinct memory configuration, but the result is still a valid representation of the set $\{A, B\}$. On the other hand, if we are representing ordered lists in the same memory architecture, then switching the locations of A and B yields a distinct ordered pair: (B, A) instead of (A, B). As a result, the number of ordered lists that can be represented in a given amount of memory is higher than the number of sets. For this reason, we find that many commonly used programming languages offer a primitive ordered list or array data type, while unordered sets are less often available, and where they are implemented, are often not a primitive data

type. For example, C++ has built-in arrays, but set implementations are only available in auxiliary libraries. So while the conventional set-theoretic representation of an ordered pair—$\{A, \{A, B\}\}$—requires more symbols than the corresponding representation for a set—$\{A, B\}$—this does not imply that sets are easier to represent than ordered lists in the memory architecture employed by the human language faculty. Indeed, Langendoen (2003: 317) observes that "set merge is not adequate for any such purpose" of "provid[ing] the basis for interpretation of linguistic objects at the interfaces." But if it is inadequate, is there any reason to hold a place for it in the human language faculty?

Chapter 6

Conclusion

6.1 Future Directions

This preliminary study has raised a number of questions that are worth pursuing further. Subsequent work can address other theoretical issues, implement computational models, and make use of psycholinguistic experimentation.

A number of theoretical questions remain to be addressed. In addition to strengthening our characterization of primitive operations, we may examine other characteristics of coordination as well as structures that exhibit similarity to coordination, including parenthesis, apposition, and parataxis (de Vries 2008). Other structures like comparatives and nonsententials may benefit from this analysis as well.

In addition, the nature of adjunct structure as opposed to argument structure has formed a core part of Hornstein's (2009) analysis. While we have not examined adjunct structure in detail in our current study, applying the current tripartite Merge approach to adjuncts, in a similar fashion to Hornstein's two-level conception, may yield insight into the primitive operations that underlie adjunction and the resulting differences among adjunct, argument, and coordinate structure. Furthermore, we may relate this approach to Kayne's (1994) model, in which all arguments originate after their head and all adjuncts originate before their head, both of which can subsequently undergo movement to yield the variety of head-complement orderings found in natural languages. Since in our formulation, Concatenate, Append, and Project all

apply to ordered pairs of objects, this order-based characterization of adjunct and argument structure may be a natural consequence of characteristics of primitive joining operations.

Another topic to explore is the syntax-prosody and syntax-semantics interfaces with respect to these primitive operations, and the nature of interface conditions that constrain syntactic structure.

The empirical applicability of the model can be improved by considering a larger variety of constructions from numerous languages.

As an approach to syntax motivated by considerations of economy and processing cost, this study would benefit from psycholinguistic experimentation to determine actual performance in parsing and production tasks. To highlight the difference in processing costs among different primitive operations, we could construct example sentences consisting of multiple applications of a single kind of joining operation, possibly sentences that repeat the same sequence of words an arbitrary number of times. For example, a possible contrastive set could include the following types of structures:[1]

(6.1) (Nested arguments with no adjuncts)

History of philosophy of history of philosophy of history of philosophy ...

(6.2) (Nested arguments with adjuncts)

A box in a house in a box in a house in a box in a house ...

(6.3) (Multiple adjuncts in succession)

A box with a cubic shape made of wood weighing one kilogram containing five apples ...

(6.4) (Flat coordination)

A box and a house and a box and a house and a box and a house ...

[1]We looked at examples analogous to (6.1) and (6.4) in chapter 5. (6.2) and (6.3) involve adjuncts, which we have not covered here in detail but may be analyzed in a manner similar to Hornstein's (2009) approach. (6.5) contains hierarchical logical structure within a flat syntactic structure. If our analysis is correct, we would expect similar syntactic processing effects to appear for (6.4) and (6.5) but possibly differing semantic processing effects.

(6.5) (Coordination with hierarchical logical structure)

(Either) a box or (both) a house and (either) a box or (both) a house ...

By varying the speed at which the sentences are presented, we can observe how well people can process the given sentences. Since processing resources are limited, those operations that are more costly would be less manageable if needed at a fast rate, so we would predict that structures formed from more demanding operations end up crashing the parser and registering as ungrammatical. It is a challenge to construct comparable constructions, since factors such as the words used in each example, the frequency or presence of repetition, and the possibility of ambiguous readings can make a direct comparison difficult. Nevertheless, at a first glance, constructions which remain essentially flat and avoid the use of Project (6.3, 6.4) seem much easier in my judgment to process than ones that require an operation like Project (6.1, 6.2), especially when read at a fast rate. In addition, examples that involve consecutive applications of Project (6.1), with no room for combining operations of any other kind, seem significantly more difficult to process than ones that interleave opportunities to combine phrases using less costly alternatives (6.2). Perhaps we observe this effect as a result of the possibility of an alternate reading, as (6.2) can be parsed as having the same structure as (6.3), and the available cheaper alternative may force the syntactic processor to adopt it, even though it conflicts with the intended semantics. In (6.5), the compositional meaning of the sequence is difficult to deduce at a fast rate, but the effect seems different from that of parsing (6.1). If the goal of exact semantic comprehension is disregarded, an attempt to parse (6.5) only to obtain a general idea of the phrase becomes much easier, almost as easy as (6.4), while (6.1) still remains difficult. This observation would agree with our theory that on a syntactic level, hierarchy in coordinate constructions is encoded in a flat manner and that a full hierarchical representation is only needed at the conceptual-intentional interface.

Research on event-related potentials (ERPs), electrical potentials measured at the scalp in conjunction with the presentation of a stimulus, may yield insight into how the brain processes different kinds of structures. The N400 effect, in which a strong negative peak is observed about 400 ms after a stimulus, has been shown to appear

when semantically incongruous words are presented (Kutas and Hillyard 1980). For syntactic anomalies, there is a similar P600 effect, in which a positive peak appears after a 600 ms delay (Osterhout et al. 1997). The P600 effect has been shown to arise even for grammatical sentences that are difficult to process, such as garden-path sentences in which a preferred resolution of an initial ambiguity turns out to be an incorrect choice (Kaan et al. 2000). Measuring ERPs in response to different types of structures may indicate which operations are more costly as well as which domains (syntactic or semantic) meet with difficulty in processing certain structures; this may illuminate processing differences between constructions like (6.1) and (6.5).

Another direction to pursue would be computational modeling of the primitive operations proposed to see how well they perform on a corpus. Artificial parsers that produce structured representations from a string of tokens have encountered difficulty particularly with coordinate structures, in part due to structural ambiguity among possible pairings of coordinands (Park and Cho 2000). In some cases, performance can improve if coordinate structures are distinguished; for example, Charniak's (2000) probabilistic parser performing on the Penn *Wall Street Journal* tree-bank corpus gains a 0.6% improvement in precision/recall after explicitly marking coordinate constructions. It is possible that existing parsers may be improved by recognizing the possibility that the joining processes underlying coordination may be subject to a different set of conditions than those behind subordination.

6.2 Closing Thoughts

Our exploration of processing efficiency as a factor affecting the choice of primitive operations has yielded fruitful results and appears to be a promising pursuit within the Minimalist framework. In addition to considering the ways in which basic operations can apply to generate syntactic structure, a focus from the start of the Minimalist Program, we can also ask *which* basic operations are preferable in a given setting, a question in line with Minimalist goals. The unique characteristics of different operations result in different kinds of structures that are reflected in empirical observations.

We can strengthen a theory of language if we recognize that the tools available for constructing language may be a set of options that are optimal for different conditions, developed and retained in the human language faculty through the process of evolution. In part, the complexity and variety of human language naturally arise from the availability of multiple operations, and by examining the characteristics of each, we may come to a fuller understanding of linguistic diversity.

References

Bachrach, Asaf and Roni Katzir. 2009. Right-node raising and delayed spell-out. In Kleanthes K. Grohmann (ed.), *InterPhases: Phase-theoretic investigations of linguistic interfaces*, 283–316. Oxford: Oxford University Press.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, 132–139. San Francisco: Morgan Kaufmann.

Chomsky, Noam. 1956. Three models for the description of language. *IRE Transactions on Information Theory* 2(3): 113–124.

Chomsky, Noam. 1957. *Syntactic structures.* The Hague: Mouton.

Chomsky, Noam. 1995. *The Minimalist Program.* Cambridge, Mass.: MIT Press.

Chomsky, Noam. 2001. Beyond explanatory adequacy. *MIT Occasional Papers in Linguistics* 20.

Everett, Daniel. 2005. Cultural constraints on grammar and cognition in Pirahã: Another look at the design features of human language. *Current Anthropology* 46(4): 621–646.

Fong, Sandiway and Robert C. Berwick. 1985. New approaches to parsing conjunctions using Prolog. In *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics*, 17, 118–126. Chicago: Association for Computational Linguistics.

Goodall, Grant. 1987. *Parallel structures in syntax.* Cambridge: Cambridge University Press.

Hale, Kenneth L. 1982. Some essential features of Warlpiri verbal clauses. In Stephen Swartz (ed.), *Papers in Warlpiri grammar: In memory of Lothar Jagst*, Work Papers of SIL-AAB, Series A vol. 6, 217–315. Berrimah, N.T.: SIL-AAB.

Haspelmath, Martin. 2004. Coordinating constructions: An overview. In Martin Haspelmath (ed.), *Coordinating constructions*, 3–39. Amsterdam: John Benjamins.

Hauser, Marc D., Noam Chomsky and W. Tecumseh Fitch. 2002. The faculty of language: What is it, who has it, and how did it evolve? *Science* 298(5598): 1569–1579.

Hornstein, Norbert. 2009. *A theory of syntax.* Cambridge: Cambridge University Press.

Johannessen, Janne Bondi. 1998. *Coordination.* Oxford: Oxford University Press.

Kaan, Edith, Anthony Harris, Edward Gibson and Phillip Holcomb. 2000. The P600 as an index of syntactic integration difficulty. *Language and Cognitive Processes* 15(2): 159–202.

Kayne, Richard S. 1984. *Connectedness and binary branching.* Dordrecht: Foris.

Kayne, Richard S. 1994. *The antisymmetry of syntax.* Cambridge, Mass.: MIT Press.

Kibrik, Andrej A. 2004. Coordination in Upper Kuskokwim Athabaskan. In Martin Haspelmath (ed.), *Coordinating constructions*, 537–553. Amsterdam: John Benjamins.

Kutas, Marta and Steven A. Hillyard. 1980. Reading senseless sentences: Brain potentials reflect semantic incongruity. *Science* 207(4427): 203–205.

Langendoen, D. Terence. 1975. Finite-state parsing of phrase-structure languages and the status of readjustment rules in grammar. *Linguistic Inquiry* 6(4): 533–554.

Langendoen, D. Terence. 1998. Limitations on embedding in coordinate structures. *Journal of Psycholinguistic Research* 27(2): 235–259.

Langendoen, D. Terence. 2003. Merge. In Andrew Carnie, Mary Willie and Heidi Harley (eds.), *Formal approaches to functional phenomena: In honor of Eloise Jelinek*, 307–318. Amsterdam: John Benjamins.

Langendoen, D. Terence. 2008. Coordinate grammar. *Language* 84(4): 691–709.

Lasnik, Howard and Joseph Kupin. 1997. A restrictive theory of transformational grammar. *Theoretical Linguistics* 4(3): 173–196.

Miller, George A. and Noam Chomsky. 1963. Finitary models of language users. In R. Duncan Luce, Robert R. Bush and Eugene Galanter (eds.), *Handbook of mathematical psychology*, vol. 2, 419–491. New York: Wiley.

Mous, Maarten. 2004. The grammar of conjunctive and disjunctive coordination in Iraqw. In Martin Haspelmath (ed.), *Coordinating constructions*, 109–122. Amsterdam: John Benjamins.

Nevins, Andrew, David Pesetsky and Cilene Rodrigues. 2009. Pirahã exceptionality: A reassessment. *Language* 85(2): 355–404.

Nunes, Jairo. 2004. *Linearization of chains and sideward movement.* Cambridge, Mass.: MIT Press.

Osterhout, Lee, Judith McLaughlin and Michael Bersick. 1997. Event-related brain potentials and human language. *Trends in Cognitive Sciences* 1(6): 203–209.

Park, Jong C. and Hyung Joon Cho. 2000. Informed parsing for coordination with combinatory categorial grammar. In *Proceedings of the 18th International Conference on Computational Linguistics*, 593–599. Morristown, NJ: Association for Computational Linguistics.

Peterson, David A. and Kenneth VanBik. 2004. Coordination in Hakha Lai (Tibeto-Burman). In Martin Haspelmath (ed.), *Coordinating constructions*, 334–356. Amsterdam: John Benjamins.

Postal, Paul M. 1998. *Three investigations of extraction.* Cambridge, Mass.: MIT Press.

Progovac, Ljiljana. 1998. Structure for coordination. Parts I and II. *Glot International* 3(7): 3–6; 3(8): 3–9.

Ross, John Robert. 1986. *Infinite syntax!* Norwood, N.J.: Ablex.

Sag, Ivan, Gerald Gazdar, Thomas Wasow and Steven Weisler. 1985. Coordination and how to distinguish categories. *Natural Language and Linguistic Theory* 3: 117–171.

Sauerland, Uli, Jeanette Sakel and Eugenie Stapert. 2009. Syntactic vs. semantic embedding (in preparation).

Selkirk, Elisabeth O. 1984. *Phonology and syntax.* Cambridge, Mass.: MIT Press.

Steedman, Mark. 2000. *The syntactic process.* Cambridge, Mass.: MIT Press.

Terrill, Angela. 2004. Coordination in Lavukaleve. In Martin Haspelmath (ed.), *Coordinating constructions*, 427–443. Amsterdam: John Benjamins.

de Vries, Mark. 2008. Asymmetric merge and parataxis. *Canadian Journal of Linguistics* 53(2): 355–385.

Wagner, Michael. 2009. Prosody and recursion in coordinate structures and beyond. *Natural Language and Linguistic Theory* (forthcoming).