GRAMMAR CONSTRUCTION IN THE MINIMALIST PROGRAM

Joshua Herring

Submitted to the faculty of the University Graduate School

in partial fulfillment of the requirements

for the degree

Doctor of Philosophy

in the Department of Lingusitics,

Indiana University

December 2016

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

Markus Dickinson, PhD

Steven L. Franks, PhD

Thomas Grano, PhD

Lawrence S. Moss, PhD

15 November 2016

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, Tom Grano, Larry Moss, and in particular Steven Franks and Markus Dickinson for their patience and support throughout this unusual process. Additionally I thank Damir Cavar for mentoring me through my early years in graduate school, Alexis Lanham for unwavering support, and of course my family, for believing it could be done.

Joshua Herring

GRAMMAR CONSTRUCTION IN THE MINIMALIST PROGRAM

The Minimalist Program is in principle nothing more than a set of guidelines for cognitive syntactic research. Because of its historical pedigree and foundational assumptions, in practice it functions and is perceived as something closer to a formal theoretical framework. This opens the door to implementational possibilities. Though it is not possible to strictly delineate "minimalist" and "non-minimalist" linguistic theories by choice of theoretical device alone, it is possible to identify, and make concrete, shared theoretical assumptions and formal devices that minimalist theories draw from. This project surveys the recent minimalist literature and catalogs the most important such devices, unifying them where possible, to build a set of implementational primitives capable of accurately representing a large section of recent proposals in Minimalism. The utility of this approach is demonstrated through the development of a grammar development software toolkit for the Minimalist Program which makes these primitives available to researchers. By implementing theories in this system, it is possible to validate their empirical claims and adjudicate disputes over empirical coverage between competing theories. Sample implementations relevant to the ongoing dispute over the Movement Theory of Control are given.

Markus Dickinson, PhD

Steven L. Franks, PhD

Thomas Grano, PhD

Lawrence S. Moss, PhD

CONTENTS

Appendices

CHAPTER 1 - MINIMALISM

MINIMALIST ASSUMPTIONS

Minimalism is a program and not a theory. It makes no claims about which formalisms are best suited to represent the structure of linguistic knowledge. Strictly speaking, it doesn't even make specific claims about what constitutes linguistic knowledge. "There are minimalist questions, but no minimalist answers, apart from those found in pursuing the program: perhaps that it makes no sense, or that it makes sense but is premature." (Chomsky 2000) It is merely a set of guidelines for asking questions, leavened with some speculation about what lines of inquiry are likely to yield interesting results.

Taken at face value, this would seem to rule out any attempt to formalize it. If it is not a theory, and if it does not make any testable claims about how language works, it's not clear what kinds of objects any hypothetical formalization would operate over, let alone how to evaluate any attempt to specify them. As the founder himself notes, "It is a misunderstanding to contrast 'minimalism and X,' where X is some theoretical conception (Optimality Theory, Lexicalism, etc.). X may be pursued with minimalist goals, or not." (Chomsky 2000) Minimalism *per se* has no truth value.

And yet in practice, The Minimalist Program's pedigree has ensured that it strongly resembles a coherent theory. It finds its roots in the *Government and Binding Theory* framework proposed in (Chomsky 1981) and in subsequent work under the rubric of *Principles and Parameters* (Haegeman 1994). This foundation is both conceptual and empirical (see (Epstein and Hornstein 1999) for one confirming account).

Conceptually, the crucial shift involved dispensing with traditional, rule-based grammar systems in favor of very general, universally available and invariant operations which transform (syntactic) structures that, in contradistinction to the more familiar explicitly-stated, pattern-based structures of formal grammars with construction-specific rules, are *projected from* lexical items (more accurately, their categories) and combinations of lexical items. Expressions generated by the transformational component are well-formed or not according to how well they conform to a set of universal constraints. (See (Chomsky 1995) for elaboration.)

On the empirical side, the tendency has been for Minimalist researchers to build on work done in this tradition, taken to have been highly successful at capturing important, non-obvious truths about human language, and as such to be a good set of guideposts for uncovering the principles of Universal Grammar. Minimalism treats *Principles and Parameters* research generally as descriptively correct but ill-stated: formulated in ways that distract from, or at least are not particularly revealing about, *why* human language should be the way it is. (Chomsky 1995; Hornstein, Nunes, and Grohman 2005)

This question of *why* human language should be one way and not some other animates minimalist inquiry. Minimalism is a program in the sense that it fixes on no particular answer to this question, but rather seeks to generate a family of theories that propose competing answers. In practice, researchers have adopted the same one, however, formulated succinctly as the **Strong Minimalist Thesis** of (Chomsky 2000):

> *Language is an optimal solution to legibility conditions*

It may be more accurate to say that this has been a guiding principle of the program than to call it a thesis. To the extent that it is a thesis, one would need (a) some sense of how *language*, for the purpose of this statement, is bounded in the cognitive domain, (b) some specifics about how these *legibility conditions* are constituted pursuant to an evaluation function ranging over

solutions, and (c) a definition of "optimal" for this purpose, since any hypothetical evalution function will depend on it. None of these particulars are currently forthcoming in anything but the vaguest of terms.

Neither were they promised. In an important sense, Minimalism "works backward" compared with more familiar scientific approaches. It knows what theory it would like to propose, but it is certain neither of the specific formulation of this theory, nor even whether it is viable. The pursuit is nevertheless thought worthwhile on the strong hunch that *something like* the target theory will prove viable. If this turns out to be true, the deviations from the original target may well be illuminating in their own way. Human language is the way it is either because it is an optimal solution to legibility conditions, or because it would be if it could, but for something preventing it from being so, and as we have a scientific curiosity about what that "something" is, it will have been fortuitous that we phrased the question in just this way.

The "hunch" itself is based in a series of empirically grounded, if not conclusively established, assumptions about the origins and nature of natural language (Chomsky 2005). First, that it is a biological system in the same sense that the visual or circulatory systems are, consisting perhaps of some dedicated parts that are specific to language, but mostly using and interfacing with organs that additionally serve other purposes. Second, that it emerged late in human evolution, and quite rapidly compared with other biological morphology. Third, that it is reasonably modular - that is, that a "faculty of language" exists fairly independently of other systems of the body.

If all this is true, it is reasonably clear in the abstract both what "interface conditions" are and why they would be the arbiters of linguistic output. Since the newly-evolved language system came late to the game, comparatively little of its structure is specific to its operation. It makes use of existing systems - such as arms and digits for sign language, or the tongue,

mouth and larynx for spoken language, as well as the propositional apparatus of the brain for logical interpretation, etc. Linguistic output must come in a form that these systems can convert into useable (mental) objects, and with minimal disruption of their preexisting mode of operation; the onus is squarely on the language faculty to conform to the external systems. It is also reasonably clear why we might expect the language faculty's "solution" to the problem of interfacing with the systems it has coopted for external expression to be something like optimal. If it is true that language's appearance was rapid, it stands to reason that only a *minimal* number of changes to preexisting primate biology are involved. Explaining the outsized effectiveness of language is much easier if this minimal number of changes were particularly well-adapted to the niche they evolved to fill - i.e. that they "got it (mostly) right the first time."

> *Imagine some primate with the human mental architecture and sensorimotor apparatus in place, but no language organ. It has our modes of perceptual organization, our propositional attitudes . . . insofar as these are not mediated by language, perhaps a "language of thought" in Jerry Fodor's sense, but no way to express its thoughts by means of linguistic expressions . . . . Suppose some event reorganizes the brain in such a way as, in effect, to insert FL. To be useable, the new organ has to meet certain "legibility conditions."* (Chomsky 2000)

This "evolutionary fable" has consequences for how concepts like "minimal" and "optimal" are to be understood in the program. It is tempting, and a common mistake, to understand these exclusively in their traditional guises as guidelines to reduce, as far as possible, theoretical and operational complexity. Indeed, both of these senses are operative in the Minimalist Program (and inherited from the *Principles and Parameters* approach - see (Chomsky 2008) and (Gallego 2010)): theorists should not multiply entities (in the sense of *lex parsimoniae* - "Occam's Razor"), and simpler operations that impose a lower computational burden (involving fewer or less expensive "steps") are to be prefered to ones that are more involved. But there is the additional *desideratum* that the operations and objects composing any minimalist theory be independently biologically justified, a point which creates some tension

between the two traditional understandings of scientific parsimony. If a proposed linguistic operation is complex or burdensome compared with another, but is also better-grounded in independently-motivated biological processes, the traditional preference for the simpler operation is not decisive. Minimalism, therefore, is not wedded to computational efficiency in the sense a computer scientist might use the term. It is first and foremost a program of cognitive biology, however contentious it may be for a group of researchers which by and large lacks biological and psychological training to assume that mantle. Consequently, computational linguistic researchers should not necessarily expect minimalist theories to yield machine-efficient results. The claim is rather that their results will be cognitively accurate compared with other approaches to machine-based language implementation (see (Hornstein and Idsardi 2014) for elaboration).

MINIMALIST MACHINERY

At least as regards the backbone architecture of the human language faculty, agreement among minimalist researchers is near-universal. There is assumed to be a set of linguistic features **F** which is universally available and language-independent. These are features in the "Bloomfieldian" sense of marking parameters along which linguistic objects form oppositions with one another - the idea being that the collection of features marks "what is special" about a particular item. Individual languages **L** choose from among these features, so that human languages will select a subset of those available in the universal inventory, but never the entire set. There are no theories at present about how this selection takes place, nor what the limits are, but there is plenty of empirical work capturing apparent dependency patterns/implicational hierarchies found in the resulting selection. Features are bundled into lexical items - LI - which are collected into a Lexicon - **Lex** - available to the language

faculty. The language faculty - FL - comprises this lexicon and a computational engine - sometimes called *narrow syntax* - which applies operations from a set of universally-available and invariant computational processes - C_{HL} - to generate a set of *expressions* - **{Exp}**. Characterization of a language **L** as defining membership in a set of expressions is common in formal language theory (Hopcroft, Motwani, and Ullman 2001), and has been integral to Chomskyan approaches to natural language since at least (Chomsky 1957), and arguably since (Harris 1951). It is indeed characteristic of most competing approaches as well.

Where Minimalism begins to distinguish itself from current approaches and its immediate predecessors is in the way it characterizes C_{HL} and evaluates membership in **{Exp}**.

To the extent language is "an optimal solution to interface conditions," **{Exp}** should contain primarily - preferably *only* - material which makes sense to the interfaces. Put differently, FL should output information which is useful to other cognitive systems, and ideally *nothing more*. Since, by assumption, input to narrow syntax is composed of bundles of features from **F** (i.e. of *lexical items*), Minimalist **{Exp}** consists only of arrangements or sequences of members of **F** that other cognitive systems can use. A sequence of features that is acceptable to a given set of external cognitive systems is said to *converge*, and all others *crash* (Chomsky 1995) - terms which can be taken as jargon for marking membership, and lack of same, in the (infinite) set of expressions that characterizes any particular given language.

Unfortunately for the progress of Language Science, it is not yet possible to open up a human mind and, with any precision, take inventory of its cognitive subsystems or say in any detail how they interact with each other. "The Interfaces" must necessarily remain, for the foreseeable future, theoretical constructs. Minimalism naturally takes the conservative approach and tries to assume as little as possible about them, postulating only two: the **A**rticulatory-**P**honetic (A-P) interface - sometimes called **PF**, and the **C**onceptual-**I**ntensional (C-I) interface - somtimes

called **LF** (Chomsky 1993). These constructs are conscious oversimplifications, stand-ins for what are possibly arrays of cognitive subsystems of varying degrees of applicability to language proper. They are conceptualized in this way simply because of general agreement, spanning the entire history of the study of language, that at a minimum grammars pair "sound" (i.e. expression/articulation) and meaning (Chomsky 2005).

In principle, little more is required. The faculty of language could simply consist of a set of linguistic features and a random sequencer that *selects* all possible arrays of all possible lengths (the *power set*) from its members, throws them at the interfaces and sees which ones stick. But such a "solution" is unsatisfying for a number of reasons. In addition to generating a bunch of linguistic "junk," it offers no insights into *why* some sequences of features work and others don't. It is in this space - the space between the existing set of features and the interfaces that make use of them - that Minimalism hopes to be illuminating.

Naturally, therefore, the lion's share of the action of Minimalist theorizing plays out in placing constraints on what operations comprise C_{HL} - i.e. constraints on what narrow syntax can do. It is important to understand that "placing constraints" is not perfectly reducible to "eliminating machinery." If it were, the "random generation" theory would be optimal. Minimizing machinery is important, but it takes a back seat to minimizing the load on cognition in general. Generating endless amounts of junk just to extract the useful subparts is therefore out of bounds.

At least two aspects of the "random generator" baseline will necessarily be retained: **Select** and the idea of a **Lexical Array**. The need for some procedure that selects subsets of **Lex** is clear: the individual natural language utterances we observe do not make use of the full panoply of lexical items or features. Therefore, there is a mental process that sections them off - call it **Select**. The existence of **Lexical Array**s, subsets of lexical items selected as input

to individual firings of C_{HL}, follows analytically (any selection from a set is automatically a subset, which minimalists - somewhat at odds with standard practice in formal language theory - call an "array").

Leaving the "random generator" approach behind requires something more, however. With just **Select** into **Lexical Array**s that are input to the operations of C_{HL}, any operations of C_{HL} would be suboptimal departures from simply letting the ultimate arbiters of convergence - the interfaces - have at the **Select**ed arrays directly. Either **Select** must be more *selective*, or **Lex** must have some structure over and above just being a repository for features bundled into lexical items, or else narrow syntax does more work than simply arranging features.

And in fact, Minimalism makes use of all three possibilities - the first explicitly in the form of *lexical subarrays* which form the basis for derivation in *phases* (see (Chomsky 2001), (Chomsky 2008)), and the second implicitly with the assumption that certain lexical items, called *core functional categories* (CFCs), are specially marked in the Lexicon to form the nucleus for such subarrays, determining in part which (classes of) other items can be present (see (Chomsky 1995)). What items constitute the set of CFCs is a matter of some dispute, but the idea that at least C (= the clause), T (= tense) and (various flavors of) v^* (= the so-called "light verb") are in the set is generally accepted.[1]

The third - that narrow syntax adds interface-necessary information not directly present in the lexicon - is universally accepted but a bit contentious to state, as it flirts with violating an important proposed economy condition, the **Inclusiveness Condition**:

> *Any structure formed by the computation is constituted of elements already present in the lexical items selected for N[umeration]; no new objects are added in the course of computation, apart from rearrangements of lexical properties (in particular, no indices, bar levels in the sense of X-bar theory, etc.)* (Chomsky

[1] It should be noted here that not all minimalist researchers fully adopt or are entirely comfortable with phases. See (Epstein and Seely 2006) for one skeptical approach.

1995)

In other words, narrow syntax isn't allowed to add any *lexical* information that wasn't already there. In some important sense, all it does is select and arrange feature bundles from the lexicon into configurations that the interfaces can use. But there is at least one important reason to believe it is a bit more than that: the ubiquitous *displacement property* of natural language.

Possibly the only fundamental way in which natural languages differ in form from artificial languages, such as programming languages, and a factor present in all known human languages, the *displacement property* is a fact that any theory of natural language must account for (Hornstein, Nunes, and Grohman 2005). Put simply - unlike in artificial languages, where sequential position uniquely determines interpretation, in natural languages interpretation can be disjoint from sequential position. In fact, elements can be interpreted in *multiple* positions (though in the general case they are articulated in only one). The target minimalist conclusion would obviously be that something about the tension between the two interfaces - some difference in intelligibility requirements between A-P and C-I - necessitates this. And in the ideal case, not only is displacement born out of the tension between these two systems, it further turns out to be the computationally optimal out of all available ways of resolving such tension.

At present, there is of course no clear sense of why an "optimal" solution to differing legibility requirements would require displacement, but that is to be expected. Again, it is a target conclusion, one the Minimalist Program would like to reach, not one for which there is currently any compelling evidence. It takes the form, for now, of a working hypothesis, one that guides the inventory of proposed operations in C_{HL} in important ways.

An obvious place to look for such a solution is in the fact that conceptual relations seem naturally hierarchical while articulatory relations are sequential. Perhaps, then, something about the need to express hierarchical relations in sequential form tailors C_{HL} in a way that makes displacement a computational "good fit".

Indeed, the core operations assumed by the vast majority of minimalist theories fit that bill. These are: **Merge**, **Probe**, **Agree**, **Value**, **Move**, and, with some mild controversy, also **Copy** and **Delete**.

Of these, **Merge** - the operation that combines two[2] syntactic objects into a composite syntactic object - is fundamental. No theory of syntax which is even mildly lexicalist can exist without a version of it. Typical minimalist theories take **Merge** to be technically unconstrained (see (Gallego 2010)): **Merge** simply takes two inputs and returns a single output. Consequently, it is not by itself the locus of whatever constraints keep the system from overgenerating (see (Boeckx 2008b)).[3] What it does do is establish a number of relations that are variously believed to have importance at the interfaces. The simplest of these is *sisterhood*, which obtains between any two objects that have been **merge**d. Another is *c-command*, which is established between one object and all the subparts of an object with which it has **Merge**d. Since **Merge** is containment-based - that is, it constructs syntactic objects out of syntactic objects - the hierarchy that seems important for legibility at the C-I interface comes as a by-product.

Constraints on **Merge** are realized - directly or otherwise - by **Agree**, paired with either **Value** or **Delete** (or, perhaps suboptimally, both). Since the program already assumes

[2]In minimalist practice there are always two; in principle, any number are possible

[3]Some theorists, notably (Chomsky 2008), are loathe to allow any operations without a featural motivation and have stipulated *edge features*, a special kind of feature that simply allows an item to undergo **merge**. Empirical motivation for this is somewhat unclear.

the existence of features, it is natural to suppose that features are what determine the subset of arrangements of lexical items from **Lex** that are in **{Exp}** By further assumption (the **Inclusiveness Condition**), the operations of C_{HL} are forbidden from creating lexical information. Therefore, featural constraints will either take the form of **Value**-ing features - that is, determining which subclass a set of features representing the parent class actually instantiates in the particular expression - or **Delete**-ing them.

Either choice represents a mild violation of the spirit of the **Inclusiveness Condition**. In the former case, this is because determining that what was an abstract class is a concrete subclass at least *reveals* information that wasn't strictly present in the lexicon. In the latter case, this is because it seems to violate the optimality assumption. There is something *prima facie* suspect about the idea that a lexicon optimized for communicating with external cognitive interfaces encodes information those interfaces cannot use. Using at least one of **Value** or **Delete** seems unavoidable, however, for, as noted above, if narrow syntax were simply in the business of arranging features, the random generator approach would be enough, and the theory not very explanatory.

In truth, the violations in both cases are mild to the point of irrelevance. Determining that a featural superclass has such-and-such a value in a given expression is an operation that itself has an economy interpretation: why should the lexicon create multiple lexical items when a single one can be leveraged for a class of related purposes? Likewise, the idea that the lexicon would encode information that one or the other of the interfaces can't use is, from a certain point of view, not different from saying that operations of C_{HL} are necessary to appease the interfaces - an assumption that underlies any justification for the existence of C_{HL} *a priori*.

If **Agree** is to explain displacement, it is natural to assume that it operates at a remove - i.e. is not strictly local - and determining its boundaries is arguably the core driver of

minimalist research. So, subject to the conditions imposed by an individual minimalist theory, **Agree** is an operation that obtains between two lexical items - or perhaps features specified on them - which are in some relationship with each other resulting from a previous **Merge** operation. This could be simple *sisterhood* or some more "distant," perhaps *c-command*ing, configuration.

Most accounts take **Agree** to be asymmetric: one half of the pair is a *probe* and the other is a *goal*. The distinction is motivated in equal parts by conceptual necessity and economy considerations. On the conceptual side, regardless of whether one chooses the **Value** or **Delete** route, the effects of the operation affect each party differently, implying an asymmetry. On the economy side, the asymmetry is useful in limiting the scope and number of operations: **Agree** operations can be triggered only upon **Merge** of a *probe* (an item with requirements to satisfy), for example, or perhaps are limited to searching for goals among their lexical subarray.

As mentioned, it's not too much of a stretch to say that what bounds are placed on **Agree** relations are the real meat of any minimalist theory. These are not exclusively configurational. Cyclic considerations are also central, and it is precisely this function that *phases* serve.

Phases were introduced as a way of constraining the choice of **Lexical Array**s, but this is only one of their purposes, and not really the most important one. In the course of the derivation, they additionally realize bounding conditions, delimiting the range over which syntactic operations can apply, and also act as a means of ordering operations, by requiring that operations involving members of the subarray apply to them as a group before any member of the group is allowed to interact with members outside. Implementational details of *phases* can vary quite a bit among minimalist theories, but it is not necessary to survey all of them (see (Citko 2014) for a fairly comprehensive survey). The purposes are always the

same: to constrain the domain of application of a given operation in some particular way.

The final operation, and the one most directly implicated in explaining *displacement*, is **Move**, sometimes called **PiedPipe**. This process happens as a reflex of **Agree**, but not all instances of **Agree** result in **Move**. When **Move** occurs, a syntactic category that occupies one position in the assembled hierarchy is attached at a second position determined by the *probe* member of the **Agree** relation that triggered it. The word "attached' is used advisedly, to emphasize that the source instance of the **Move**d item is still represented in its original position, just not overtly realized in any articulatory sense - at least not in the general case (see (Boskovic and Nunes 2007) for evidence that articulation of instances of objects that have subsequently been re-attached by **Move** is sometimes grammatical). Syntactic items can be *attached* at multiple points in the hierarchy provided there is some motivation to do so, where"motivation" typically means marking that an **Agree** relation has occurred that **Delete**d or **Value**d features that would otherwise have caused a crash at one or the other of the interfaces[4].

Minimalist theories can, of course, add to this inventory as needed, provided the proposed operations are empirically motivated and conceptually deferential to minimalist concerns, but these are the basic builing blocks. To review:

1. There is a *lexicon* which is a repository of linguistic features bundled into (atomic) lexical items. The lexicon is specific to a given language, but the features it bundles come from a universal inventory.
2. There is a set of operations C_{HL} - assumed to be universally available and invariant across languages - which can be applied to items from the lexicon

[4]Alternately, an item is copied and re-**Merge**d, with an independent interface process responsible for **Delete**-ing one of the other instance of a duplicated item. See (Nunes 2004) for a prominent implementation of such a system.

3. Operations of FL begin by **Select**ing items from this inventory into a **Lexical Array**.
4. **Select**ion is organized around special lexical items that comprise the class of *core functional categories*.
5. "Organized around" means that individual examples of CFCs guide **Select** by identifying numbers and categories of elements to choose from. Items so chosen end up in a *subarray* with the CFC instance.
6. *Subarray*s of the **Lexical Array** identify *phases*. *Phases* serve to limit the range over which C_{HL} operations can apply and, in an indirect way, to constrain the order in which operations must apply in the formation of a longer utterance.
7. **Merge** is the fundamental structure-building operation, responsible for establishing relationships between items in the **Lexical Array**. It is in principle unconstrained, but some choices of pairs of items to **Merge** may end up blocking, as a side-effect, **Agree** operations between other items that need to happen for the derivation to converge.
8. A language consists of all the expressions constructable from a *lexicon* under these constraints that *converge* at all the interfaces.
9. An expression *converges* if it contains only information which is legible at the interfaces. In practice, this means all of its uninterpretable features have been eliminated, or unvalued features have been valued, at some point before the end of the dervivation.
10. The primary method of restricting *convergence* among the model objects to those that correspond to grammatical utterances in a real language is by devising ways to block **Agree**. These blocks typically take the form of limiting search space - i.e. are implemented through conditions established by *phases*, or are established through intervention effects - configurations where an unsuitable *goal* blocks a *probe* from finding a better match.

THE PURPOSE OF THIS PROJECT

Perhaps the most striking feature of all of this machinery to people unfamiliar with it is how imaginative it seems. While it is true that nothing presented here is without conceptual motivation, it is equally true that there is no direct way of verifying the cognitive reality of any of it. It is all speculation about how the faculty of language might operate, and that only if certain guiding assumptions turn out to be correct, which they may well not.

The ontological status of things like *phase*s and **Merge** is in an important sense left up to the individual reader. They could be blueprints - plans for how to build the eventual theory. They could be scaffolding - temporary fixtures which enable construction of the real theory. They could be placeholders - ways of talking about families of mechanisms yet to be determined. Or they could even, in the fortunate case, be direct analogues of cognitive processes that will turn out to be only slightly messier than these idealized versions.

What they cannot be is directly observed.

That's alright. The lack of direct access to the object of study is no secret, and the hypothetical nature of the object of study is hardly a unique condition in the sciences. Many things about the world that we think we know have essentially the same status. What is available is a vast amount of easily-obtainable lingusitic data. Minimalism's theoretical constructs are useful to the extent they account for that data, and that is all.

What perhaps does set Linguistics a bit apart is the comparative lack of a concrete toolset. Physicists work with constructs that are easily as imaginatve - and certainly at starker odds with everyday experience - as those syntacticians devise. If Physics has a better reputation

for objectivity, then because physicists can typically be more explicit about what it means for a theory to fail, and that often because the predicted effects are isolable.

In Syntax, comparatively little is isolable. The object of study itself - FL - is mediated through layers of cognitive machinery of varying degrees of linguistic specialization. On top of that, every utterance, by hypothesis, is formed from the same narrow band of operations on features. There is the very real possibility of unintended consequences, therefore: minor changes in assumptions about these devices to account for one observed phenomenon may well (and often do) silently disturb assumptions made previously to account for other phenomena.

For that reason, internal consistency is the best yardstick for success a minimalist researcher has. Theories are successful to the extent that they account for large amounts of data with minimal amounts of side effects (undesireable results requiring stipulative repair). The trouble is that there is no easy way to test for "side effects" short of mentally going through the motions of deriving fleets of expressions with the proposed new device.

The purpose of this project is to add to the syntax researcher's working toolset a way to automate such evaluations, so that they can be applied quickly to larger and more varied sets of data than an individual researcher typically thinks to include in his own mental checks. This brings benefits on a number of levels. For one, by building a standard set of data sentences, researchers working in parallel can quickly determine what the relative strengths and weaknesses of their individual proposals are, especially beyond the sets of data with which they are directly familiar. In the minimalist world, broader coverage isn't the only metric for the success of a theory, but even in cases where researchers trade coverage for perceived cognitive accuracy, it can be useful to know what areas are not covered as a jumping off point for further investigation. More importantly, though, the exercise of specifying a theory in enough detail that a machine can "run" it over data is an invaluable clarifying

device. Programming environments are less forgiving than a human observer, as programming environments are *tabula rasa*. They don't - can't - share intuitions or assumptions. Forcing oneself to lay bare intuitions and assumptions can by itself lead to insights. Pointedly, it is exactly this kind of exercise, applied to *Government and Binding Theory* results, that was the bread and butter of the early Minimalist Program.

Of course, the open-ended nature of the Minimalist Program - the fact that it is a program and not a theory - makes full realization of this project perhaps impossible. But like the program it augments, full realization is maybe also not the point. If the exercise itself brings insights and gets the program closer to being able to state the target theory in the appropriate detail, it is a success.

ROADMAP

The remainder of this presentation is divided into two sections.

The first of these, comprised of chapters 2 and 3, surveys core mechanisms. As noted above, it is not possible to properly delineate the domain of grammatical operations that are allowed for minimalist analysis. For this reason, this project is in an important sense impossible. What is possible is to justify the choice of core operations that any conceivable "minimalist" toolkit should include, and that is what these chapters aim to do.

Each are based in historical survey. Because Minimalism is not a formal theory, it is, in a sense, its lineage. Not unlike the "undocumented constitution" of the United Kingdom, it has a few foundational texts augmented with years of precedent, practice and referenced authority, rather than any definitive set of formal commitments. Understanding how it came to be the

way it is is critical in characterizing the field.

In its current iteration, Minimalist analysis is organized primarily around Phases and Agree. This section therefore deals with each in turn.

Chapter 2 deals with Phases - where they came from, where they stand, and what purposes they serve. It is ultimately concluded that *phase* in the current understanding is a bit of a hodgepodge. The category bundles together ideas that are conceptually separate: (operation) ordering, grouping, and locality. The current system provides mechanisms for separating these concerns.

Chapter 3 deals with Agree. Though Minimalism is, on the surface, a fully lexicalist program, it retains traces of its heritage in the Gen-Eval tradition of GB. In contradistinction to most formal theories of grammar, combinations of items are in principle unrestricted in Minimalism. What guards against "illicit" combinations are not so much prohibitions on combinations themselves as the consequences that arise from them. Deciding what those consequences are is the purview of Agree, making Agree the primary determinate of "grammaticality" in Minimalism. Chapter 3 motivates a value-based concept of Agree with a survey of current emprical arguments and concludes by outlining what Agree mechanisms have to be present to capture the insights of modern Minimalist theories.

These motivational sections are followed by a concrete introduction to the implemented toolkit. Chapter 4 outlines the programatic objects and mechanisms involved, and which form the core of the tools available to users. Chapter 5 then follows with a high-level example of the system's utility as applied to longstanding disputes around the Movement Theory of Control.

More concrete examples of system operation can be found in a series of appendices. Appendix

A provides step-by-step recreation of one of the analyses in Chapter 5 in sufficient detail that curious readers can duplicate system functionality in a competing program. Appendix B provides examples of ways derivations can fail and discussion of how to repair them. Appendix C supplies the lexicons and starting lexical arrays used in many of the examples presented, including the two main ones.

This writeup then concludes with some discussion of insights into the Minimalist Program gained over the course of the project and an outline of how the project should be expanded in the future.

CHAPTER 2 - AN OVERVIEW OF PHASES

MOTIVATION

One of the side-effects of Minimalism's reimagining of syntactic derivation as a pair-wise, bottom-up process was the introduction of a "timing" problem: certain independently licit operations had to be constrained from applying at certain points in the derivation to prevent ungrammatical results. The canonical example (modified slightly from (Citko 2014), citing (Chomsky 2000)) involves **Merge** of an expletive with an expression involving a small clause:

1. Many people are likely to be at the party.
2. There are likely to be many people at the party.

Both expressions are grammatical and express the same idea. They differ only in the presence of the expletive *there*.

Introducing the expletive has some interesting consequences. These include (a) that the expletive intervenes in any attempt to move *many people* over it (in (2)) to the left edge of the expression and apparently also (b) that it must be **Merge**d in the **Spec** of *to* (**T**).

Motivation for (a) comes from examples like (3):

3. *Many people are likely there to be at the party.

Although licit at the left edge of the clause in (1), *many people* is barred from this position when the expletive is present.

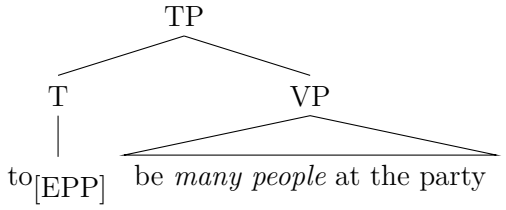
More interesting is that *many people* seems to be barred from moving at all under these circumstances. That is, even to the right of *there*, moving *many people* seems to violate some constraint of the system:

4. *There are likely many people$_i$ to be t_i at the party.

It is a bit surprising that this should be the case. By assumption, *many people* lands in **Spec-T** - that is, just to the left of *to* - on its way to the left edge of the clause in (1). Furthermore, this cannot be intrinsically motivated (such as for case checking), or *many people* would never be licit in its original small clause position, as it is in (2). *many people*, it seems, either makes it "all the way to the top," or it doesn't move at all. But if the intermediate movement is not motivated by anything intrinsic to the phrase - i.e. is in some sense "optional" - something must prevent it from happening in cases like (2) where the expletive is present.

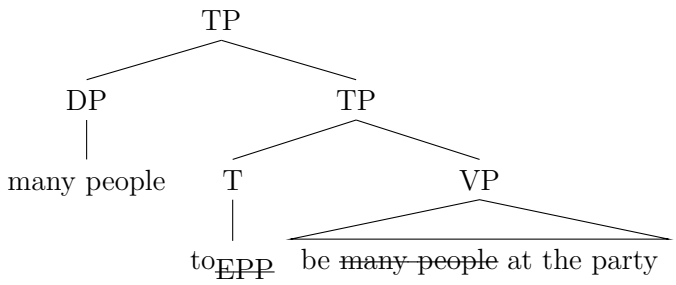
The addition of *phase*s to the Minimalist toolkit begins with (Chomsky 2000)'s solution to this problem.

If the intermediate movement step is real in (1), then the most straightfoward way to rule it out in (2) (thus avoiding the ungrammatical (3)) is to somehow require *there* to enter the derivation at that point, blocking it at the source. This is the approach (Chomsky 2000) takes, proposing a **Merge over Move** principle that, all else equal, requires the derivation to proceed with a **Merge** operation at junctures where both **Merge** and **Move** are available and motivated. The tree diagram below shows such a juncture:

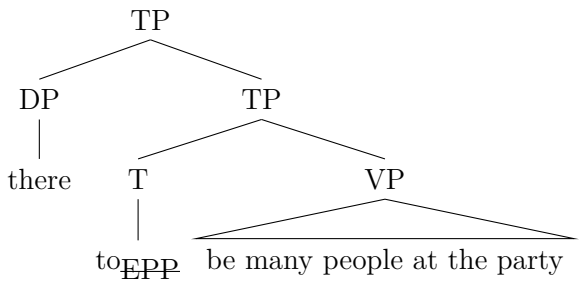


After the **Merge** of **T** *to* and **VP** *be many people at the party*, assume there is an *EPP*[5] feature on **T** that must be checked. This can be done either by **Move**-ing *many people* there, or by **Merge**-ing in *there*. **Merge over Move** requires the derivation to prefer the latter. Each is schematized below:

MOVE *many people*



MERGE *there*



By **Merge over Move**, the derivation schematized by the first tree is licit only if there is no available expletive to **Merge**, as in (1). If one is available, moving *many people* is ruled out, and only the derivation schematized by the second tree is allowed.

That solves the immediate problem, but at the cost of introducing others. Computationally,

[5]**EPP** is a bit of a problematic category in Minimalism. It is essentially a placeholder - a feature that implements the fact that some syntactic projections seem to require - or at least allow - specifiers for reasons unknown or not fully understood. *EPP* simply requires that the projection have a specifier, often without much in the way of discrimination about what types of objects fill it.

Merge over Move has the potential to massively complicate the grammar by requiring the comparison of parallel derivations; it requires **Lookahead**. At the point in the derivation where **Merge over Move** must come into play, it is not yet known if **Merge**-ing *there* will be required. *There*, for all the derivation knows, could end up put to some other use later, one where it doesn't block *many people*'s path to the left edge, obviating the problem that the principle is enlisted to solve. For example:

5. There is reason to believe that many people will be at the party.

Having **Merge**d in *there* at the first **Merge over Move** point would have yielded[6] the ungrammatical:

6. *There is reason to believe that will be many people at the party.

In (5) and (6), the facts are reversed from (1) - (4): **Move** seems *required* here.

There are, in essence, two ways to solve the problem without giving up the principle: either **Merge over Move** admits of exceptions, or there is some reason why *there* is not available at the relevant point in this derivation.

(Chomsky 2000) opts for the latter. The **lexical array** that limits access to the **Lexicon** for the purposes of a given derivation is given more articulation. It is subdivided into **Lexical Subarray**s, operations on each of which must complete *independently* of the others as far as possible before being **Merge**d together. Access to elements in (structures formed from) one subarray by (structures formed from) elements of another is not completely ruled out, but it

[6] Of course, there are other mechanisms (e.g. case-related considerations) to either prevent *there* from **Merge**-ing at this point in the derivation - or else to prevent it from moving to the left periphery if it is **Merge**d here.

is sharply limited in a manner to be detailed. The solution to the immediate problem is that **Merge over Move** holds *only for elements of the same subarray.*

Lexical Array = {{C, T},
{v, be, reason, there, to, believe},
{that, will},
{many, people, v, be, at, the, party}}

The composition of the various subarrays given here will not be entirely uncontroversial, but they serve to illustrate the proposed mechanism, which is that at the juncture in the derviation when **Merge over Move** could force **Merge** of *there* in place of **Move**-ing *many people*, the derivation does not have access to *there*.

Of course, nothing *prevents* inclusion of *there* in the relevant subarray:

7. There is reason to believe that there will be many people at the party.

Lexical Array = {{C, T},
{v, be, reason, there, to, believe},
{that, will},
{there, many, people, v, be, at, the, party}}

But it is telling that including it doesn't obviate the requirement for the second, independent *there* in the higher clause. The higher and lower clauses are in an important sense syntactically independent of each other - constructed separately from independent pools of lexical resources. Contentful lexical items which contribute to the meaning of the expression may be shared between segments, but items introduced for purely grammatical functions - such as expletives - are included only to satisfy the needs of their domain. Defining such domains precisely is the aim of phase theory.

TRANSFER

Phases were introduced to explain an ordering constraint, but that is not the only work that they do. They additionally serve to limit access by later operations to portions of the syntactic object already assembled, and in this capacity they can be leveraged to explain otherwise stipulative reconstruction effects.

It is well-known that anaphoric binding interacts with displacement in expressions like the following (examples from (Boeckx and Grohmann 2007)):

8. *John$_1$ said that Sue likes pictures of himself$_1$.
9. Which pictures of himself$_1$ did John$_1$ say that Sue likes?

Example (8) is straightforward: *himself* should refer to *John*, but it cannot because *Sue* intervenes as a "closer" potential binder. The ϕ-feature mismatch between *Sue* and *himself* (*Sue* is a feminine binder, but *himself* must be bound by a masculine antecedent) renders the binding impossible, and the derivation crashes.

(9), by contrast, is a bit of a puzzle. By assumption, an antecedent must stand in some kind of superiority relation to its bindee - c-command, dominance, or something of the sort. However, the bindee *himself* would seem to be in the superior position with regard to *all* its potential binders. The only way to save the assumption would be to allow binding to resolve at some point in the derivation before *Which pictures of himself* has moved to the left edge of the expression (or else allow binding to resolve after movement - perhaps at the interfaces - but over traces/unpronounced copies of the moved item). But at first blush, this kind of approach merely reintroduces the problem in (8) as the best-motivated "trace" position for *Which pictures of himself* is as the **complement** of *likes*.

10. Which pictures of himself$_1$ did John$_1$ say that Sue likes ~~which pictures of himself$_1$~~?

Sue will again be the closest potential binder, and the binding will fail as before.

What seems to be required is for *which pictures of himself* to also exist in some intermediate position in the derivation - somewhere lower than *John* but higher, in some sense, than *Sue*. And indeed, this seems intuitively correct. The central puzzle here is, after all, what it is about *displacing* the *wh*-phrase that permits a binding resolution that was otherwise impossible; the explanation should involve movement.

10. Which pictures of himself$_1$ did John$_1$ say ~~which pictures of himself$_1$~~ that Sue likes ~~which pictures of himself$_1$~~?

The trouble is motivating such movement. However well-aligned with the empirical facts, it isn't very conceptually satisfying to simply note that intermediate landing sites exist. Minimalism is founded on the notion that the computational system shouldn't do anything it doesn't strictly have to, and intermediate landing sites would seem to fall into that category.

"Would seem to" but for the fact that there is already some independent motivation for phases. The system needs some mechanism to account for **Merge over Move** facts; if that same mechanism also helps explain intermediate landing sites, all the more reason to adopt it.

(Chomsky 2000) does just that, motivating phases in part by appeal to notions of economy. If phases are the transfer points - points where assembled material is "shipped" to the interfaces - then they serve to restrict the domain of syntactic operations. A given operation need not consider the entire derivational tree, as the parts of it that are "known" to be complete can be removed from the workspace. This hypothesis is made concrete in the **Phase**

Impenetrability Condition (as defined in (Chomsky 2001)):

> **Phase Impenetrability Condition**: The domain of H is not accessible to operations at ZP; only H and its *edge* are accessible to such operations.

ZP, in this context, is meant to be the [next] smallest strong phase - that is, the next phase head that c-commands H in the course of the derivation. So, the assumption is that anything that is **complement** to a phase head is inaccessible to further syntactic operations after a higher phase head has been merged in. If there are unvalued uninterpretable features in this domain at this point, the derivation crashes. This has the effect of keeping the derivation immediate: by narrowing the search space, and thus the range of options, the computational system can more quickly determine whether a derivation is worth proceding with or should be terminated than would be the case if it had to consider the full range of already-**Merge**d lexical items.

The relevant point for the present discussion is that it is the *domain* of the phase head that is transfered to the interfaces. The "edge" and the head itself remain available. If this view of the system can be maintained, the need for intermediate landing sites is clear: without them, movement to the operator position at the left edge of the expression would be impossible. Put differently, it is a requirement of the system that syntactically "active" items remain in narrow focus, and short-move displacement to intermediate landing sites along the edge of the phase to escape transfer are the mechanism by which they are kept visible.

Of course, it remains to be explained why the system should be constructed this way. It seems correct to say that the otherwise mysterious existence of intermediate landing sites gains some cachet once there is independent reason to believe in (something like) *phases*, but this does nothing to explain why *phases* operate the way they do - permitting access to their left edge, but not their complements - or why they should exist at all.

THE LEFT PERIPHERY

Broadly speaking, researchers have come to terms with the special active status of items in the left periphery of a *phase* in three ways: denial, acceptance and reanalysis.

The **denial** approach argues that the special status of the left periphery is an epiphenomenon: items **Merge**d to the left will be structurally higher than items in the **Complement**, and as such are naturally more visible to outside operations anyway. It's not that access to the **Complement** is completely ruled out so much as just computationally more difficult, as there are, almost by definition, intervening elements between any **Probe** and a potential **Comp**-situated **Goal**.

The standard-bearer for this approach is (Fox and Pesetsky 2005). This work takes the view that there is nothing to explain about why the periphery should be more accessible than any other part of the *phase* for the simple fact that *everything* in a lower *phase* is in principle accessible. The traditional distinction between **SpellOut** - as a PF-only sequentialization process - and **Transfer** - as an LF process that reads semantic relations from the assembled structure - is maintained, and any computationally salient effects of *phases* exist to ease the burden on **SpellOut**.

Syntactic tree structures are hierarchical in nature and do not by themselves establish precedence relations. Indeed, the existence (and ubiquity) of displacement phenomena in natural language puts tree structures into tension with precedence relations, as items can be and frequently are attached to multiple positions in the hierarchy, making the mapping to any strict sequential precedence relation problematic. The physical utterances that represent tree structures, however, adhere to strict sequentiality between realized items, with an item

articulated only once in the general case no matter how many places it occupies in the hierarchy (see (Felser 2004) for documented exceptions, however), and typically in the sequential position naturally corresponding to the *highest* hierarchical position it occupies (but see (Boskovic and Nunes 2007) for counterexamples). **SpellOut** is the PF process responsible for establishing these relations.

Phases ease the burden on **SpellOut** by "pre-compiling" precedence relations. At the end of a phase, ordered pairs are established between each of the items involved, according to where they sit in an *in-order traversal*[7] of the tree. Once established, these relations are inviolable - circumventing them causes chaos at the articulatory interface, and illegibility at an interface is Minimalism's analogue to "ungrammaticality" in other frameworks. This constraint is codified in a **Linearization Preservation** principle:

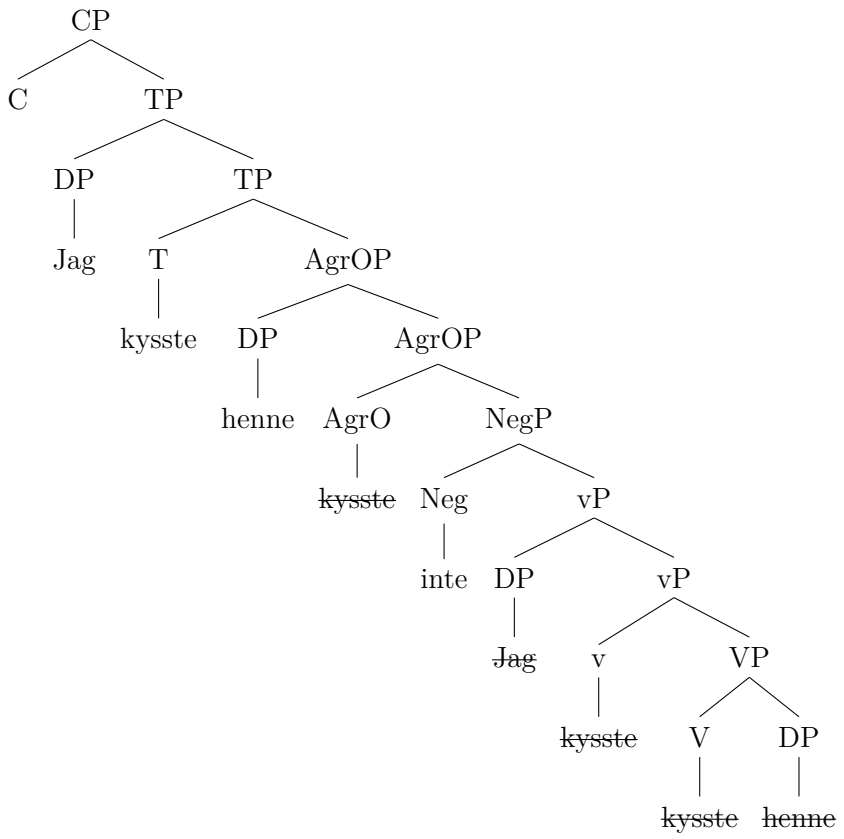
> The linear ordering of syntactic units is affected by Merge and Move within a Spell-out Domain, but is fixed once and for all at the end of each Spell-out Domain.

The authors believe this principle is what underlies Holmberg's Generalization (Holmberg 1986):

> Object shift of an element α from the complement domain of a verb β occurs only if β has moved out of VP

Concretely:

[7] An *in-order traversal* of a binary tree starts at the root of a tree structure and then recursively orders all elements of the tree by ordering elements in the left branch of each node ahead of elements in the right. (See (Sedgewick and Wayne 2011) for a more detailed explanation.) Though the syntactic literature tries to avoid explicit encoding of concepts like *left branch* and *right branch* in its discussion of hierarchical structures, they can generally be mapped onto purely hierarchical relations like C-command. See (Kayne 1994) for the most influential example of such a mapping process.



Jag kysste henne inte ("I didn't kiss her")

If the object is going to "shift" out of **VP** - to a hypothetical "object agreement phrase" **AgrOP** or whatever the analysis in question requires - it can only do so if the sequential ordering relations established at the last phase are preserved. Since v**P** is a phase by hypothesis, the relative SVO ordering between subject, object and verb established by that point in the derivation has to surface in the final **SpellOut** of the expression. The subject must precede the verb and the object, and the verb must precede the object. The negation head *inte*, by contrast, is not ordered with respect to the subject, verb and object at the (v**P**) phase level. It will be ordered at the **SpellOut** of the next phase up: **CP**. Items are free to reorder with respect to *inte* at this point, but they may not reorder relative to each other. In this system, everything in VP is fully visible and accessible at the **CP** phase level, even though **VP** is

Complement to a phase head (and as such would be transfered to the interfaces in more standard phase systems, rendering it inaccessible for further operations). There is therefore no problem with **AgrO** "seeing" *henne* or attracting it to its **Spec**. The only constraint imposed by the lower phase on the ordering of *henne* is that by the end of the present phase (**C**), *henne* must still be lower in the hierarchy than *kysste* and *Jag* - as indeed it is.

The crucial point is that this system makes no codified distinction between the subject *Jag* and the object *henne* with respect to this requirement. The fact that *Jag* appears in the periphery and *henne* in the core of **VP** is of no formal consequence. The phase encompasses the entire phrase.

The **acceptance** approach takes the opposite tack, treating the phase edge as indeed special and leaving explanation of this fact as a bit of an open question. To the extent explanations are offered, they tend to be conceptual. For example, (den Dikken 2014) justifies it with a (somewhat circular) appeal to necessity: if there were no escape hatch at the periphery, no derivation would ever succeed. If a phase, in essence, is a unit domain for the purposes of interaction with the interfaces, then (nearly) every derivation would crash for want of a way to eliminate remaining uninterpretable features. Any uninterpretable features which cannot be eliminated in the phase proper - and there are nearly always such features, as they are the source of the derivational engine - will need to be eliminated in the extended derivation. That phrases seem to organize themselves into a core - **Head** and **Complement** - and a periphery - (a variable number of) **Specifier**(s) - is simply optimally compliant with these independent requirements. If the system weren't set up in this or some similar way, it wouldn't work.

This sort of approach is by far the most popular, and it tends to embrace - again, out of necessity - what has come to be called the **PIC**$_2$ or the **Weak Version** of the **PIC**. In the original formulation of the **Phase Impenetrability Condition** of (Chomsky 2000), the

Complement to a phase head was spelled out at the merger of the next phrase head:

> In phase α with head H, the domain of H is not accessible to operations outside α, only H and its edge are accessible to such operations

This proved to be overly strict, however. Among other things, it would seem to rule out the object shift derivation given above. At the point where *inte* is merged in, items in v**P** are transfered and no longer available for syntactic operations, making it impossible for **AgrO** to attract the object *henne*, since it is not at the phase edge[8]. For this and other reasons, the definition was modified in (Chomsky 2001) so that the **Complement** remained accessible until the merger of the next **strong phase head**:

> The domain of H is not accessible to operations at ZP; only H and its edge are accessible to such operations.

With this definition, where **ZP** is the next *phase*, the object shift derivation above is possible: *henne* is visible to all operations up to the point where **C** is **Merge**d (with **TP**). Since **AgrO** is **Merge**d below **C**, *henne* is still available for attraction to its **Spec**.

It is really only the conceptual emphasis which is different between **denial** and **acceptance**. The Fox and Pesetsky version avoids a lot of technical complications, but at the expense of committing itself to a philosophical claim which might not pan out. The claim here is that the conceptual and articulatory interfaces are highly independent, with the conceptual interface operating over the entire assembled structure (which is why narrow syntax retains access to structures assembled in previous phases), and only the articulatory interface requiring incremental computation. This is insightful if it turns out to be true, but there are reasons to be skeptical. Phases seem at least as conceptually separable as they are phonetically, after

[8]It should be noted that there are many proposals for the landing site of the object in object shift constructions and that not all of them are problematic for $\mathbf{PIC}_1$. Many involve adjunction to vP, for example, an analysis which treats the object much like the subject - it has adjoined to make itself available to syntactic operations beyond the phase that contains its θ-position. See (Vikner 2005) for a good survey of analyses of object shift.

all. It seems plausible the *v* phrase and the **C** phrase are grammatically/conceptually special, functioning as syntactic "kernels" around which the derivation might organize itself. The idea that they would be isolable in any articulatory way that other phrases are not seems a bit less solid.

The more standard approach - the **acceptance** approach - has the baggage of a number of unexplained technical add-ons. It's not clear why the grammar should wait until the *next highest phase* to spell out the previous phase, nor why it should then split it in half and spell out only part of it. These feel like stipulations. But as a tradeoff for the more stipulative feel, the **acceptance** approach can remain agnostic about how the interfaces work, which seems advisable at this stage in the development of Linguistic science. Interfaces are, after all, a theoretical construct - a mere placeholder for something Minimalist researchers *expect* to find when Biological science has advanced far enough to confirm or deny their existence. For now, they remain necessarily vague, and speculation about exactly how they operate is perhaps best avoided until more direct examination is possible.

For researchers unsatisfied with the stipulative feel of the standard approach but unwilling to wander too far into speculation about the nature of the interfaces, the best option has been to try to clarify the definition of *phase* - a **reanalysis** approach. (Richards 2010) forms a prominent example.

The central question of that work is of what constitutes a *phase* - how does the system know where the boundary is? There are essentially two possibilities here, corresponding to early and late conceptions of the *phase* in the work of Noam Chomsky. In earlier work, Chomsky's phases were defined by the lexical subarrays the **Merge over Move** principle motivated. The phase was the subarray, and the subarray defined the phase. How subarrays came to be composed was a question for later research (Chomsky 2000; Chomsky 2001), an approach

heavily criticized as circular and unenlightening (see e.g. (Boeckx and Grohmann 2007)). As research progressed, Chomsky became more convinced of the idea that phase heads simply *are* the core syntactic engine, concretely realized as a condition by which *only* phase heads bear uninterpretable features. Since "all syntax" happens around phase heads in this system, lexical subarrays become almost redundant: a *phase* is the phase head and what it selects, and nothing more is needed (Chomsky 2008).

On close examination, this approach neither eliminates the circularity problem nor brings the field any closer to understanding the mysterious edge condition. As (Richards 2010) notes, there is no meaningful *conceptual* difference between defining a phase as whatever a phase head selects and organizing the lexical array into subarrays. Either way, the phase head is the defining kernel and is grouped with a (limited) number of items that either do or don't meet its specifications.

But if there is no meaningful difference on the conceptual level, it can still be the case that one or the other is more revealing as an implementational model. They may be mere notational differences, but notation that is well-suited to its target can be illuminating.

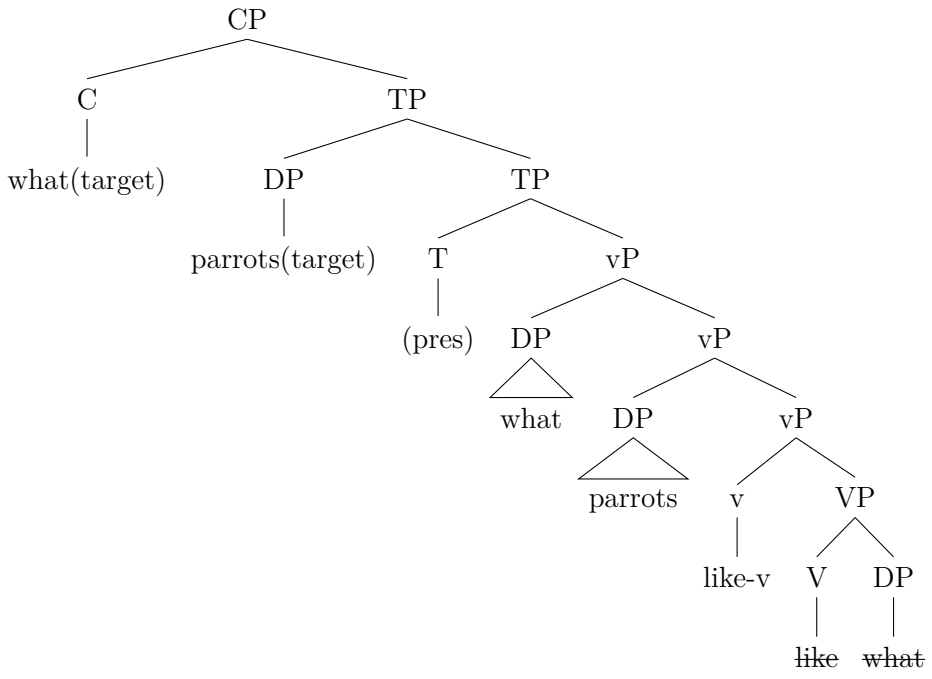
In this spirit, (Richards 2010) prefers lexical subarrays because they focus attention on the *grouping* aspect, more clearly raising questions about how the groups are chosen. The first insight this yields is that the difference between the two definitions of the **Phase Impenetrability Condition** given above reduce to a difference in how **T** is grouped.

Recall that in the **Strong Phase Impenetrability Condition** - sometimes called $\mathbf{PIC}_1$ - the complement to the phase head is spelled out as soon as the next *phrase* head is merged - that phrase be what it may:

In phase α with head H, the domain of H is not accessible to operations outside

α, only H and its edge are accessible to such operations

As noted above, this definition is empirically problematic in a number of ways. In addition to preventing certain approaches to object shift, it also blocks any chance that **C** would have to interact with the object in examples like the following (adapted from (Citko 2014)):



C seems to need to be able to have access to features on the object of the verb (*what*), and yet it can't do so as long as the object is buried in a spelled-out **Comp** - vP. One could get around this by having the object adjoin to the left of *vP* (as illustrated), but it's not clear what would motivate the adjunction, as this is not a canonical object position and so doesn't seem a likely target for A movement. Moreover - such adjunction blocks agreement between the subject (*parrots*) and **T**. Analyses that involve an *AgrO* do not overcome the difficulty: the complement of a spelled-out v phase is as opaque to one type of higher-up phrase as another. **AgrO** doesn't have any special privileges here.

But if the so-called **Weak Phase Impenetrability Condition** - sometimes called **PIC**$_2$ - holds, the problem vanishes:

> The domain of H is not accessible to operations at ZP; only H and its edge are accessible to such operations.

Here, **ZP** refers to the *next highest phase* - **CP** in the case where **H** is v. Since **T** is not a phase head, and since **T** is lower in the hierarchy than **C**, the **VP Complement** of v remains accessible to it - crucially including access to the object[9].

(Richards 2010)'s insight is to note that this is functionally equivalent to simply including **T** in the lexical subarray with v. Schematically, it amounts to a difference between this:

Strong PIC: {C, T}, {v, V}

and this:

Weak PIC: {...C}, {T, v}, {V...}

Furthermore, the idea that the only the "periphery" of the phase - that is, its **Spec** and the phase head itself - should be accessible to **Probe**s of higher phrases, but not the **Comp** falls out of this grouping. The **Comp** of v here is the result of the completion of operations on **V** and whatever else finds itself in the same lexical subarray. **VP** is merged as **Complement** to v only after its cycle has completed; v selects this assembled object as its internal argument, and the internal argument remains active/accessible so long as v is - that is, until the v phase completes. This provides a kind of answer to why it should be the **Complement** of a phase head that is spelled-out: it is because the **Complement** always originates in a separate

[9] Of course, *what* still blocks movement of the subject *parrots* to **T**. This issue is typically resolved with appeals to the idea that items in the same *phase* periphery are *equidistant* goals for the purposes of locality. Some details are given later in the chapter.

lexical array.

FEATURE INHERITANCE

Especially conceptually attractive in ruminations about *phases* is the idea that *all and only* the phase heads have *uninterpretable/unvalued* features. The philosophical appeal is clear: to the extent that phases are (relatively) syntactically autonomous, it is implementationally risky to place syntactic drivers outside of the heads that mediate these boundaries. If phases are demarcated by phase heads, and if phases are the drivers of syntactic operations, and if syntactic operations are driven by *unvalued* features, then phase heads alone should determine what "gets driven" in their domain. The only way to implement this in a system driven by lexical features is to confine the driving features to phase heads. For once, definitional circularity pays off: if phase heads are the drivers of syntactic operations, and if uninterpretable features are what drive syntactic operations, then phase heads must be the heads that come with uninterpretable features.

Conceptually appealing as this is, however, it leaves the program without an account for **T**. **T** pretty clearly drives syntactic operations, responsible as it is for verb positioning and subject marking. **T** seems to need to be a phase head in practice but not in concept.

(Chomsky 2008) proposes **Feature Inheritance** as a way of having this particular cake while still eating it. **T** is not a phase head, but it *acts like* one as soon as **C** is merged insofar as **C** transfers its syntactically active features to it.

While the details of the mechanism for this transfer remain muddy, it does have a number of empirical points in its favor. Since **T** does not acquire its driver features until **C** merges in

and completes the phase, there is no question of *ordering* between **C**-driven and **T**-driven operations. Rather, all operations happen "simultaneously" in "phase time." And if this is so, some nagging timing questions can be put to rest.

Holmberg's Generalization

For one, the countercyclicity in Holmberg's generalization vanishes. Holmberg's Generalization was a fact about object shift that seemed to be ordered "backward." From the original (Holmberg 1986):

> *Object shift of an element α from the complement domain of a verb β occurs only if β has moved out of VP*

That's all very well for a framework like *Government and Binding Theory* which allows free movement, but the Minimalist Program's grounding in pairwise **Merge** entails that structure-building operations be ordered. The paradox, then, is this: object movement is allowed only after verb movement has succeeded, but once verb movement has succeeded it's too late for the lower object to move.

With the implication of feature inheritance that phases form some kind of domain of simultaneity for otherwise-ordered operations, however, this concern can be laid to rest, and Holmberg's Generalization ceases to be a countercyclic misfit.

Subject Marking

Another formerly vexing ordering paradox that need no longer concern us after the move to feature inheritance is the question of how the subject makes it to **Spec**-TP at all. On the

assumption that only the edge of a phase head - and not its interior - is visible to operations at the next phase level, both the subject and any **C**-bound *wh*-phrase object must occupy **Spec**-v**P** at the time **T** is merged. Moreover, on the assumption that **Spec**-v**P** is the subject's θ-position, the *wh*-phrase object is likely higher, as it was adjoined later, having been moved from **Comp**-V[10] after the (soon-to-be) subject's initial **Merge**. If true, this would imply that it blocks any **T Probe** hoping to receive ϕ-valuation from the external argument - as it is itself a nominal category - which in turn means that the external argument's Case features will never be valued, and the derivation will crash. This may even be true even under systems that treat all **Spec**s as equidistant as it's not clear that the **T** probe can "pick and choose" from an array of similarly-situated items.

Under **feature inheritance** this problem can be made to go away in one of two ways.

First, on the assumption that the *wh*-object and the external argument were not already equidistant by virtue of both being specifiers (and on the further assumption that equidistance obviates intervention), the effective collapse of **C** and **T** into a single category through feature inheritance makes the entire phase domain a "drop in time" for the purposes of these operations. If **C** and **T** act like a single lexical item, as per (Chomsky 2008), then all syntactic drivers of the phase are unordered relative to each other, and operations can proceed as makes sense. Whether or not the (soon-to-be) subject and *wh*-object are ordered relative to one another, the **C-T** complex can try its probes in whatever order works out.

Second, it could instead be that the **Probe**s on **C** and **T** are still ordered, but by virute of **feature inheritance** the order is the opposite of what their relative hierarchical positions

[10] Motivated, presumably, either as a reflex of Case agreement (on the assumption that v is responsible for **Acc** case assignment), or by so-called *agnostic movement* (Franks and Lavine 2006), whereby an item that bears unvalued features at the end of a phase may move to the edge of the phase as a **Last Resort** to save the derivation from crash.

would imply. It all depends on the ordering of **Inherit** relative to **Probe**. If **Probe** immediately follows **Merge**, as seems reasonable, then **C** may **Probe** for a *wh* match *before* **Inherit** passes its ϕ and Case features to **T**. **C**'s **focus** probe finds and attracts the *wh*-object, which is displaced to the left edge of **CP**. This leaves no intervener between **T** and the external argument of v**P** after **Inherit** completes.

As a motivation for the otherwise stipulative **Inherit**, this is of course appealing, but it does beg the conceptual question of how to *prevent* the active ϕ probe from attracting the subject to **Spec-CP** before **Inherit** has a chance to work. The analysis is based in the idea, after all, that **Probe** precedes **Inherit** in operational order.

The answer is probably something like one or both of the following:

1. **Inherit is Probe**. Whatever *probe* goes in search of the subject will have to pass through **TP** anyway. Whatever mechanism entangles **C** and **T** to allow **feature inheritance** might as well fire at this point as on **Merge**.
2. On **Merge** and subsequent **Inherit**, **C** and **T** are no longer entirely separate entities, but instead behave as though they had been merged as a single lexical item[11]. If that's the case, there is no meaningful difference between the ϕ probe originating from **C** or from **T**. **Inherit** simply changes which part of the complex acts as the site of attraction for those features, and it doesn't actually matter where they originate from. Put differently, in **Inherit**-ing features from **C**, **T** also inherits whatever they attracted. It would be tempting here to say that this implies that the ordering of the subject below the *wh* landing site is yet another PF epiphenomenon. It isn't so much that **T**, *per se*, inherits ϕ and Case as that associating ϕ and Case with **T** in the complex instructs PF to order their associates after the associates of features on **C**. But this is probably not

[11] This seems to be how (Chomsky 2008) sees it.

right. Subject is ordered below **Operator** *in the hierarchy*, and not just for purposes of **SpellOut**. So, **Inheritance** means just that: **T** receives the features, and whatever is associated with them.

Summary

Phases are a bit of a problematic category in Minimalism. Though they play a large role in modern analysis, it is not always entirely clear what purpose they serve. More accurately, they seem to serve several seemingly disparate ones. In one role, they are the delimiters of *locality*. *Phases* define (relatively) independent syntactic domains. In another role, they *punctuate* interpretation. Interpretive functions in Minimalism are served by appeals to reconstruction effects, and *phases* both delimit where those effects are observed and, on some accounts, justify their existence. In yet another role, they are the drivers of syntactic operations. There is a conspicuous division of lexical items - and lexical features - into those that provide interpretive substance, and those that drive operations. Phases reflect this distinction by organizing the derivation into suboperations around syntactic drivers. In still another role, they play a part in accounting for the distribution of grammatical formatives like expletives. In a final role, they provide *ordering* constraints, requiring that some operations take place before others.

Implementation need only allow for building analyses compatible with those prominent in the field. So long as it provides the means, it need not share all of the philosophical commitments. In this spirit, it is probably best to separate the disparate concerns bundled in *phases*, as they seem ripe for separation. To the extent that things are conceptually separable, an implementation should allow them to be separated in practice.

As will be seen in Chapter 4, this system does not provide a *phase* primitive. Rather, it

addresses the implementation of (phenomena associated with) *phases* in two separate ways.

On the *locality* side, it leans on Agree in a manner to be detailed in Chapter 3. To give a preview, if Agree is conceptualized as a *valuation* (or, if prefered, a *specification*) process, rather than a checking process, probes can be bounded by choice of goal feature. A *phase boundary*, then, is simply a "universal goal" - a goal that matches (and therefore blocks) every conceivable probe, rendering items below the head bearing the feature in question inaccessible.

On the *system architecture* side, this implementation provides `lexical array`s which bundle lexical items for subprocessing. This allows implementors to enforce ordering constraints by grouping items in determined ways. Additionally, in a manner that gains clarity in Chapters 4 and 5, by taking feature inheritance - as outlined above - quite literally, the system can enforce domains of operations that span beyond the phrase boundary. Providing lexical groupings in the form of arrays is arguably a reimplementation of the notion of *phase* by other means, as these groupings *are* architectural primitives[12] in the system. If the reader prefers, it is not unfair to say that this system commits itself to a very weak notion of *phase*, in which every subgrouping (in most cases, every phrase) is a *phase*.

Phases, it will ultimately be argued, are an epiphenomenon - a bundling of disparate primitives into a questionably motivated category. As they pervade modern Minimalist analysis, however, it is important for any implementation to survey them, and the motivations for postulating them, and ensure that all associated phenomena are covered by other means. That has been the purpose of this chapter.

[12]Though not strictly required. As can be seen in the more detailed derivations in the appendices, implementors are free to include only a single lexical array in their `derivation` classes if they like. However, this frequently yields muddled results. The claim that *phases* are not a system primitive is based on part on the fact that use of grouped subarrays is not required.

CHAPTER 3 - AN OVERVIEW OF AGREE

MOTIVATION

Jargon is a common barrier to entry to any field. The inarguably indispensible economy and precision of communication it affords professionals keeps amateurs at bay, unable to even receive instruction in the field's latest insights without some re-education in how to use superficially recognizable words. Minimalism's reductionist heritage virtually guarantees that this problem is worse for it than for other branches of Lingusitics. The commitment to leveraging directly observable processes to power even the most hypothetical of un-observables (or at-most-indirectly-observables) necessarily results in putting formerly familiar terms to counterintuitive use. While **Agree** isn't the worst offender on this front, it's hard to deny that Minimalism uses agreement phenomena for things well outside their normal bailiwick.

Ask anyone, and he knows immediately what *agreement* means: it's that thing verbs do where they change their form based on the subject. If the subject is 3rd person, add an *-s*, or maybe an *-es*. Otherwise, don't. For speakers of morphologically richer languages than English, it might also be that thing adjectives do where they are pronounced differently based on the class of the noun they modify. Whatever the domain, in ordinary use, *agreement* is some flavor of pronunciation rule employed to reflect already-understood relationships between words. As such, it's perhaps useful in a communicative sense, but it's also ultimately superficial.

Like with so much else in Minimalism, the minimalist researcher sees this "backward." If there is a pronunciation reflex, there must be a process that results in it. And if pronunciation is superficial, the process is not. Since the process exists, attributing things to it is "free" - at least in the sense that we know we're not multiplying entities. Soon enough, by extended

analogy, *everything* that involves establishing relationships between words is attributable to (the process that underlies) *agreement* - now called **Agree** (perhaps to mark the departure) - even if there no direct alteration in pronunciation associated with the relationship. What started as a reflex is now the cause. If **Merge** underlies the assembly of complex syntactic objects from atomic ones, **Agree** mediates the consequences of such combinations: what the side effects are, and whether they are licit at all.

FEATURE CHECKING

In the early days of Minimalism, accounting for "whether [given combinations] are licit at all" was really all **Agree** did. The foundational principles of the Program allowed for nothing else. These principles included[13]:

Inclusiveness Condition No new features are introduced in the course of linguistic derivation

No-Tampering Condition No elements introduced by syntax are deleted or modified in the course of linguistic derivation

The **No-Tampering Condition** ruled out the idea of *valuation*. If an item - a verb, say - reflected *agreement* with a subject in its pronuncation, then only because it had been selected from the Lexicon with the features that specify that type already present. Anything else would be *tampering* - altering the makeup of an item by adding feature specifications that were not there before. As a consequence, a Lexicon could be expected to contain express

[13] Definitions are taken from (Narita 2011) but conform well to the principles as originally formulated in the foundational works (Chomsky 1993) and (Chomsky 1995)

entries for *every morphological alternation of a given item* - quite a storage memory burden for morphologically rich Lexicons.

Enforcing what (Preminger 2014) terms "The Obligatoriness of Agreement" - the observation that some morphological combinations[14] are illicit - was a matter of *checking*. **Agree** simply identified whether two items were structurally in a relevant relationship with each other and then *check*ed whether that relationship were allowed. It was not the case that, on encountering a third-person, singular subject, the verb would alter its form to reflect this relationship - the verb was either third-person and singular to begin with, or the combination with the subject failed to succeed.

The **Inclusiveness Condition** ensured that **Agree** had no meaningful side effects. Once a feature had been *check*ed, the best that could happen is that it be marked as such, perhaps through deletion[15]. Valuation of underspecified features - something that would become standard in Minimalist theories after (Chomsky 2001) - was rejected out of hand. Again, a verb agreeing with a subject did so not because establishing the relationship with its subject had altered its form in any way, but because it just so happened to be the variant of the verb from the Lexicon that is capable of combining with that subject.

This approach to **Agree** lives on in the project by Ed Stabler and colleagues to give a formal specification for Minimalist Grammars. Starting with (Stabler 1997) and continuing to

[14] e.g. **John think she's stupid* - at least in standard varieties of English the **3rd** person marker is required when the subject is singular.

[15] There was always a tension in early Minimalism between the proscription by the **No-Tampering Condition** on altering the form of participating items in a derivation and the founding idea that Syntax was motivated by the need to remove any material that is present in the Lexicon but illegible at one or both of the interfaces.

the present day[16], *Minimalist Grammars*[17] represent features in terms of two oppositions: between *select* and *category* features and between *licensor* and *licensee* features. Each pairing is divided according to polarity, with a *select* feature pairing with a matching *category* feature, and a *licensor* feature pairing with a *licensee* feature. Features that match in type and bear opposite polarity correspond directly to the checking relationship of (Chomsky 1995).

VALUATION - A RETHINKING

Feature checking is no longer, however, the standard approach among Minimalist researchers. Starting with (Chomsky 2001) and slightly before, the working paradigm shifted to one focused instead on *valuation* for a variety of mostly conceptal reasons. For one thing, it was recognized that a massively redundant lexicon, containing entries for potentially large numbers of slight variations on individual items, was not very "minimalist" in spirit. Instead, it should be reconceptualized in "Bloomfieldian" terms - meaning here as the smallest possible list of "exceptions," i.e. unique properties of an individual item that distinguish it from others.

> *In any case, we can think of LEX as in principle "Bloomfieldian," a "list of exceptions" that provides just the information required to yield the interface outputs, and does so in the best way, with least redundancy and complication.* (Chomsky 2001)

For another, there was some discomfort with the tension, mentioned earlier, between the **No-Tampering Condition** and the founding idea that Syntax is driven by the need to remove interface-illegible material. It seemed incongruous that the evolutionary development of human linguistic systems would proceed by producing featural objects that were specifically

[16] Or at least up to (Stabler 2011). The most recent, and explicit, variant in (Collins and Stabler 2016) does not expressly give a formalization of a linguistic feature system, but it is compatible with the system used in earlier works in this tradition, a fact mentioned in the paper.

[17] This program of rigorously formalizing core ideas and mechanisms common to most research in the Minimalist Program refers to grammars written using its objects as *Minimalist Grammars (MGs)*.

illegible at the interfaces to the presumably prelinguistic systems with which C_{HL} aimed to integrate.

> *It is as if formal features like case and agreement appear in syntax only to be wiped out before syntactic elements are interpretable* (Boeckx 2008a)

The idea that features are universally composed of a type or category and a value specification for that type, and that what Syntax needed to "repair" among lexical items before they could be interpretable in combination at the interfaces was underspecification in the value, was a much more comfortable fit. A thing that can in principle have a value but lacks one just seems more like the kind of thing that an interface would have trouble "reading" - as opposed to something expressly marked "unreadable" - and so "uninterpretable" came to mean "unvalued." Moreover, it fit in well with competing approaches to grammar, such as HPSG, which were driven by feature unification. Mechanisms available to those approaches that had enjoyed some analytical success thus became available to Minimalist researchers in their own program.

MECHANISMS - SPEC-HEAD AND PROBE-GOAL

Whether valuation or checking, however, both are mere end results of the operation that is actually under investigation. Having observed that Syntax is at heart about lexical item combination, that some combinations are licit while others are not, and that some licit combinations of a given bag of items have one interpretation and others another, the work to be done involves investigating these structure-building operations. If **Merge** exists to combine items and **Agree** to referee the combinations, then the enterprise of specifying the grammar of a given language means mainly getting **Agree** right.

The centrality of **Agree** to Minimalist Syntax has its roots in developments that began

before the program itself - specifically in the articulation of the functional core proposed in the seminal (Pollock 1989). To explain the differences in relative output positioning of verbs, objects and adverbs between French and English in parametric terms, Pollock found it convenient that there be a position between verbal inflection and the (position at which the) verb itself (enters the derivation) where the object sometimes was realized. Some of the differences in English and French object positioning could then be explained partly in terms of whether the object were in this position by the time PF-relevant information were assembled, or not[18]. The hitch came in motivating the movement[19], as there was no obvious compelling mechanism for it. The most plausible candidate seemed to be case assignment, and indeed, assuming it were case assignment lent the proposal an appealing elegance, as it managed to harmonize case assignment strategies. Up to that point, case had been assigned "under government," which was, under the hood, little more than a catch-all term for a variety of dissimilar structural configurations researchers found it convenient to treat as a natural class. Under government, subjects were assigned case in a **Spec-Head** configuration (via *m-command*) and objects as the **Comp** of a lexical head -the verb (under *c-command*, sisterhood, or perhaps simply initial **Select**). The lack of uniformity was vexing to a research program born of the need to reduce grammatical mechanisms to a minimum as an answer to Poverty of Stimulus problems in child language acquisition. With the introduction of the *Agr* node, this problem could be overcome: objects raised to a **Spec** position to get/check case just as subjects did, and only a single type of structural configuration need be implicated in case assigment.

[18] Given the commitment to the universality hypothesis (see (Baker 2003) and (Lightfoot 1979) for discussion) underpinning parametric approaches to language variation, there was never a suggestion that the object might raise to this position in French but not in English - the question was one of *when*.

[19] As of (Chomsky 1986b), which introduced the **Last Resort** principle to UG - a ban on operations which are not directly motivated - it was not enough to stipulate it.

This general approach flourished in what might be called the proto-Minimalist Era[20]. (Chomsky 1991) noted that the *Agr* head proposed in (Pollock 1989) might reasonably have been above **Infl** rather than below it, to account for subject-verb agreement in finite clauses[21]. And yet, Pollock's evidence was convincing that there needed to be a position of some kind *below* **Infl**. The solution was just to have two such *Agr* positions - one for subjects, positioned above **Infl**, and one for objects positioned below. Independent evidence seemed to confirm the conjecture: among other examples, (Holmberg 1986) had noted years before that Icelandic objects shift out of VP only then when they are case-marked NPs, and never when they are PPs, thus implicating (case) agreement in motivating movement. Examples[22] include:

(1) Nemandinn las bókinna ekki
student.the read book.the not
"The student didn't read the book."

(2) *Jón talaði við Maríu ekki
Jon spoke with Maria not
"John didn't speak with Maria."

Likewise, Lasnik and Saito pointed out in an influential presentation (Lasnik and Saito 1991) that the new *AgrO* position was compatible with evidence that had been marshalled a decade before to support a raising-to-object analysis of ECM constructions. From (Postal 1974):

(3) Joan believes that he_i is a genius even more than Bob_i does.

[20]The period between (Chomsky 1991) and (Chomsky 1995) when Minimalist principles were proposed and generally agreed upon, but there was not yet any explicit program or bevy of officially-sanctioned linguistic mechanisms which researchers were expected to strive to reduce their analyses to.

[21]If the subject agrees with the verb only then when Tense is finite, as seems to be the case, then whatever *Agr* head mediates the agreement has to already "know" the verb is tensed, which in technical terms means the *Agr* head selects *T* rather than the other way around. If **Select** is always **Head-Spec**, then it follows that the *Agr* head would need to dominate *TP* in the hierarchy.

[22]From (Boeckx 2008b).

(4) Joan believes him_i to be a genius even more than Bob_i does.

him would appear to c-command *Bob* in (4), ruling out coreference, but the same is not true of *he* in (3). The salient difference between the two is Case. *he* has **Nom** by virtue of being the subject of a finite clause, but *him* must be case-licensed by some other method. The fact that it appears to be in the matrix clause for the purposes of binding implicates displacement in its assignment of **ACC** under ECM.

Movement would seem to be correlated with case licensing. And once a position was established for object agreement, there was little objection to (Chomsky 1991) taking the short step of extending this to subject agreement as well.

This solution - separate *Agr* positions for **Nom**(subject) and **Acc**(object) case assignment - had the virtue of uniformity, but as a general program it was ultimately too powerful to be really explanatory. For a time, every problem looked like the same nail in need of the same hammer: to account for any sequential ordering phenomenon, it was all too easy to postulate a(n unpronounced) functional head in whatever position in the hierarchy would yield the correct output order and attract the necessary item to it with features stipulated for just that purpose.

(Chomsky 1995) recognized the problem and took some steps to reining it in. Functional heads assembled exclusively from features that cannot be interpreted and were not pronounced were declared suspect. The *Agr* heads had to go. Object shift was eventually reanalyzed as movement to **Spec**-*v*P. Furthermore, displacement for the purpose of realizing agreement was ruled uneconomical. If the agreement process can target subparts of an item - i.e. only those features implicated - why should movement (copy-and-remerge) be any different? The agreement relationship is only between a pair of features, not a feature and a lexical item *per*

se, so only the implicated feature should move. Cases where the whole category moved overtly as a reflex of **Agree** were relegated to the need to satisfy EPP features on the target, an explanation which is essentially a promissory note to report back when further investigation has uncovered the real cause. It did, however, have the effect of decoupling displacement from case/agreement, essentially speculating that such displacement is motivated by independent morphological or articuatory interface necessity.

As philosophically unsatisfying as jettisoning the uniformity afforded by tying agreement to **Spec-Head** relations seemed, the move came with a number of empirical virtues. Divorcing **Agree** from any local configuration allowed it to happen at a distance, which captured a number of other facts in a wide variety of languages. Prominently, it retired longstanding puzzles about the agreement of expletives with their associates and loss of inverse scope under VP-fronting as well as opened the door to more intuitive explanations for quirky case and long-distance agreement phenomena.

Expletive Replacement

Expletives agree with their associate NPs in existential constructions in a way that is difficult to ascribe to movement.

(5) a. There are/*is three men in the car.

b. They are/*is one and the same element.

The verb in these examples[23] agrees with something to its right and not with the expletive one finds in the canonical subject position. Early explanations offered in (Chomsky 1986b)

[23]From (Boeckx 2008b).

attempted to reduce this to covert movement - the associate NP raised to subject position and replaced the existential expletive covertly. However, this got the scope and binding facts wrong (see (Lasnik 1999) for a full explanation):

(6) a. Someone from New York is likely to be at the party.

b. There is likely to be someone from New York at the party.

(7) a. A man$_i$seems to himself$_i$to be doing something wrong.

b. *There seems to himself$_i$to be a man$_i$doing something wrong.

Only (6a) is ambiguous between wide and narrow scope for *someone* which should not be the case if the two are identical at LF after replacement. Likewise, *himself* should be bound for the purposes of Condition A in (7b) but is not.

Scope Freezing under VP-fronting

A similar example can be found in German, where the fact that a marginal scope ambiguity disappears under VP-fronting rules out any covert movement analysis of case assignment. Examples are from (Bobaljik and Wurmbrand 2005).

(8) a. weil mindestens einem Kritiker jeder Film gefallen sollte
because at.least one.DAT critic every.NOM film please should

"Since at least one critic should like every movie."

b. ?[jeder Film gefallen]$_{VP}$ sollte mindestens einem Kritiker
every.NOM film please should at.least one.DAT critic

"Since at least one critic should like every movie."

Example (8a) is mildly ambiguous between wide and narrow scope for "every Film." Though glossed here in the sense that there exists at least one critic who likes every movie, it can also be interpreted to mean (albeit marginally) that every movie has at least one critic who likes it. This is not true for (8b), which can only have the first interpretation. And yet, *jeder Film* has **Nom** case in both examples, casting doubt on the idea that covert movement can be involved in this assignment. (If it were, the marginal wide-scope reading for *jeder Film* should not be available in (8a).)

Quirky Case

In some cases, for some verbs, Icelandic subjects fail to agree with the verb, which instead manifests agreement with an item lower in the hierarchy - generally the object. Moreover, the target of agreement is marked - exceptionally - with *nominative* case, and the subject surfaces in some other "quirky" case - generally *dative*. The following example (from (Taraldsen 1995)) shows the contrast:

(9) Henni leiddust/?*leiddist *θ*eir
her.DAT.3sg bored.3pl/?*3sg they.NOM.pl
"She was bored with them."

The verb agrees with and values case on *θeir*, rather than *Henni*, the item that would actually appear to be in **Spec**-T (or whatever agreement position is postulated), clearly showing that *agreement* is not always coupled with the structural position canonically associated with it.

Long-distance Agreement

A number of languages seem to optionally allow agreement between embedded objects and matrix verbs, in possible (but this is controversial) violation of standard versions of the **Phase Impenetrability Condition**. The examples below are for Hindi from (Bhatt 2005):

(10) a. Ram-ne rotii khaa-nii chaah-ii
Ram.ERG bread.f eat.INF.f want.f.sg

"Ram wanted to eat bread"

b. Ram-ne rotii khaa-naa chaah-aa
Ram.ERG bread.f eat.INF.m want.m.sg

"Ram wanted to eat bread"

The optionality is notable, as is the fact that the embedded clause is non-finite, and therefore not a phase. But the problem isn't so much that this can happen across clause boundaries as that it can happen at all. The identical word order and interpretation between the two examples belies the idea that there are any consequential syntactic differences between these sentences: *rotii* - the source of the nonstandard agreement - would appear to be embedded in the subordinate TP in both cases. Moreover, there is a subject in the canonical subject position as well. Like with Icelandic quirky case, the most plausible explanation is that **Agree** need not confine itself to **Spec-Head** configurations.

Each of these phenoma is difficult if not impossible to model without decoupling **Agree** from its movement reflex.

In addition to its empirical attractiveness, the move to decouple agreement from **Spec-Head** configurations wasn't as much of a theoretical step backward as it initially seemed. After all,

on closer inspection, it's clear that **Agree**[24] needs to precede displacement of the feature-supplying item to the **Spec** of its attractor in any case, else how does the feature-seeking head know which item in its **Comp** to target?

Probe and Goal

The hypothesis that **Agree** required a **Spec-Head** configuration to be realized having been rejected[25], it was now necessary to give character to the alternative. Even with the locality requirement loosened, *agreement* remained a highly constrained operation. Only certain pairs of items matched for its purposes, and the intervention effects that had made movement analyses attractive remained.

Given these restrictions, the new metaphor was straightfoward: there would be an operation **Agree** in which a **Probe** - a head bearing (uninterpretable) features in need of a value - would go seeking a **Goal** - a syntactic obect able to supply what the **Probe** needs (Chomsky 2000). This metaphor captured the discrimination of the operation - not all objects encountered along the search path need be relevant - and of course there's no trouble imagining that *probe*s can be interrupted or blocked by elements that match enough to distract but not so much as to fulfill (so-called *defective intervention*). The latter aspect allows **Agree** to retain the insights of movement-based analyses of *agreement*, but it is the former that ultimately

[24]It has of course been argued that morphology does not drive syntax at all - perhaps most explicitly in (Bobaljik 2002), where agreement marking is taken to be a superficial PF phenomenon. If that is so, then morphological agreement marking is not necessarily evidence of a long-distance *syntactic* dependency between items. However, this should not be taken further than it goes. Rejecting morphology as a driver of syntax does not obviate the need for items to identify - and license the presence of - other items at a remove, *based on features of the involved items*. The present system need not take a position on how deep these processes go; it fulfills its purpose in supplying researchers who believe that morphological agreement drives syntax with the tools to model their systems.

[25]Not, of course, by everyone - the subject remains mildly controversial. Perhaps the most prominent attempt to maintain move-based analyses of agreement is (Chandra 2007)

preserves the derivational character of the system, as it has been argued (Abels 2003; Boeckx 2008c) that so-called "punctuated paths" - the ability for long-distance syntactic operations to "skip" intervening nodes - is the remaining salient operational difference between Minimalism and unification-based theories (LFG, HPSG) after GB's turn to lexicalism.

Despite the new metaphor and expanded empirical coverage, the implementational impact of the shift was minimal. The search domain remains the **Comp** of the **Probe**-ing head, and intervention effects are as they were before: anything of the right category can interrupt the probe by (1) being "closest" along the path to other potential goals and (2) lacking the requisite feature values to satisfy the probe. It's in the details of (1) and (2), mostly, that individual Minimalist approaches tend to differentiate themselves.

Of the two, (2) tends to have a much broader range than (1). While it is possible to imagine alternatives to hierarchical C-command as a measure of "distance," the binary branching structures that pairwise **Merge** imposes make them awkward. A **Head** selecting its **Comp** automatically c-commands it and all of the **Comp**'s constituents, and as these were likewise assembled by a **Head** selecting a **Comp**, the definition is recursive and applies all the way down the hierarchy. "Distance" falls quite naturaly out of the **Probe** metaphor: a **Probe**-ing **Head** inspects the constituents of its **Comp**, each in turn, looking inside them if necessary, until it hits a match or has exhausted the search space. Since this process proceeds stepwise, "closer" can be defined procedurally - whichever nodes come first in the algorithmically-defined search order are "closer" than those that come later. Because the fit is so natural, variants on distance algorithms generally take the tack of defining exceptions - and these exceptions, in turn, generally involve allowing something that is technically "further" by the baseline computation of distance to count as though it were "closer" than the hierarchy would suggest. These are things like counting **Spec**ifiers (and adjuncts) of a phrase as "equidistant" for

search purposes, even though they were assembled in some order by pairwise **Merge** and are therefore technically hierarchically represented. Another such modification is "restructuring," which has a long and varied history[26], but also tends to involve "shortening" distances and removing "barriers" to allow otherwise illicit operations to go through. See (Wurmbrand 2015) and (Wurmbrand 2001) for surveys of Minimalist approaches.

Specification of the features involved in **Agree** tends to offer more degrees of freedom to researchers seeking to capture new empirical discoveries, and so this is where most of the action will be in providing a grammar development tool for Minimalism. In the foundational texts, **Agree** is relatively simple.

A brief overview of **Probe-Goal** algorithms found in the literature follows. This is followed by a cursory overiew of several phenomena that help define the range over which feature specification has to work when **Agree** succeeds.

Derivation by Phase (Chomsky 2001)

A **Probe** P agrees with a **Goal** G if:

1. P c-commands G
2. Both P and G are *active*
3. P matches a feature of G
4. The matching feature on P is *unvalued* and the matching feature on G is *valued*

Worth noting here is that *both* **Probe** and **Goal** must be *active* in some sense. This is largely up to the grammar designer, but generally it means having an unvalued feaure.

[26]Foundational works date back to the 1970s, with (Rizzi 1978) and (Aissen and Perlmutter 1976) being prominent.

Feature Sharing (Pesetsky and Torrego 2007)

1. A **Probe** P is an *unvalued* feature F on a **Head**
2. Its **Goal** G is a *valued* instance of F in P's c-command domain
3. Replace the feature on the P with the one from G

This differs from the (Chomsky 2001) version in establishing a more direct connection between the two items involved: they actually share structure.

Reverse Agree (Zeijlstra 2012)

Agree relates a **Probe** P to a **Goal** G if:

1. P and G are in a proper local domain
2. P has an *uninterpretable* feature
3. G has a matching *interpretable* feature
4. **P is c-commanded by G**
5. There is no intervening object carrying the same feature between P and G

This differs from (Chomsky 2001) primarily in directionality: Goals (*valued/interpretable*) c-command Probes (*unvalued/uninterpretable*) rather than the other way around.

Parametric Agree (Baker 2008)

A language may implement only one of the following versions:

A feature F agrees with DP...

1. ... if F *c-commands* DP
2. ... if DP *c-commands* F
3. ... if DP *c-commands* F or F *c-commands* DP

In other words, the directionality of **Agree** is parametric.

Checking (Chomsky 1995; Adger 2003; Stabler 1997)

An *uninterpretable* feature F is *checked* on a syntactic object X

1. when X is in a *c-command* relation with another syntactic object Y
2. which has a matching *interpretable* feature F.

Though now out of fashion, this was the dominant paradigm in the early days of the Minimalist Program and must be implementable in any comprehensive Minimalist grammar building environment. Worth noting is that directionality is free as stated. System can choose to impose directionality requirements as they wish.

Polyvalent Agree (Merchant 2011)

1. There is a function **Val** that returns the value of a feature F: Val(F)
2. "Unvalued" features F' can have default values in some languages. In this case, Val(F') returns this value
3. If a syntactic object X *c-commands* an object Y and X has a *valued* feature F with value Val(F) and Y has a matching *unvalued* feature F' with "value" Val(F'), then let Val(F') = Val(F), Val(F')

This system is rather complicated and, by admission of the author, incomplete, as it was designed to deal with a puzzling phenomenon[27]. The general idea is that Feature values can form *nested sets* which respond in various ways to different environments. The nesting allows values to be hierarchical. An important point is that features - even unvalued features - retain a record of their past "values". This makes a nice fit with the Minimalist Program's founding principles, since feature values are never destroyed in the derivation, merely rearranged/reconfigued.

Feature Inheritance (Chomsky 2008)

Pursuant to the idea that *all and only* the phase heads of a derivation bear *unvalued/uninterpretable* features, it was necessary to add a new tool to the set. T, not being a phase head, could not be the locus of any derivation-driving features, and yet evidence that it was a locus of features associated with subjecthood was strong. The solution was to allow features that are realized in one place to originate in another: **Feature Inheritance** (as covered in the previous chapter). T would receive C's unvalued ϕ and valued $Case[NOM]$ features, allowing it to function as a **Probe** even though it isn't lexically equipped to be one. This system is best thought of as an extension to the **Derivation by Phase** approach of (Chomsky 2001). Everything that applies there applies here, with the following embellishments:

1. A functional head selecting another functional phrase can transfer features to it, provided they are not already specified on that head.
2. Once in this kind of relationship, the two functional heads act in many respects as though they are a single lexical item.

[27] See the original reference for a the full facts, but the gist is that in Aleut morphological marking on a subject argument can reflect when another (non-subject) argument is unrealized.

3. The apparent countercyclic movement this gives rise to (the subject presumably raises to **Spec**-TP only after the **Probe** appears on it) is not countercyclic at all since it happens *within the phase*.

It is the last point that is the innovation relevant to **Agree** because it forms such a radical departure from the structure-based concepts of syntactic distance that underlie most accounts of intervention effects. Everything within the same phase is equidistant, and subjects and objects both situated at the edge of v**P** do not intervene against **Probe**s targeting the other.

RELATIVIZED PROBES

To complicate matters, there is some evidence that **Agree** needs to be *relativized* - in the sense that **Probe** can skip, for the purposes of **Agree**, arguments that are of the matching category but do not have the "right" feature values. The motivating example is the Kichean *Agent-Focus* Construction (Preminger 2014).

Agent-Focus Constructions in Kichean

The phenomenon in question applies to constructions where the subject is focused and works by giving morphological preference to 1st/2nd person arguments over 3rd person arguments.

(11) ja yin x-in/*NULL-ax-an ri achin
FOC me COM-1sg/*sg.ABS-hear-AF the man
"It was me who heard the man"

(12) ja ri achin x-in/*NULL-ax-an yin
FOC the man COM-1sg/*sg.ABS-hear-AF me

"It was the man who heard me"

The focused subject is presumably higher than the object in both sentences, and yet the agreement probe only ever seems to find the 1sg argument, regardless of where it is in the hierarchy. In the second example, *ri achin* ("the man") should in theory block any attempts by the verbal ϕ probe to reach the more deeply embedded object *yin* ("me"), but this is not the case. Similar effects are seen when both arguments are 3rd person but one is plural and the other singular: the plural argument "wins" regardless of position in the hierarchy. The only time 3sg morphology surfaces in this construction is when there are no other types of arguments. When *both* arguments are 1st/2nd, by contrast, the construction patterns as one would expect and agreement is preferentially with the higher argument - the subject.

1/2sg/pl > 3pl > 3sg

This example and examples like it justify what Preminger calls *relativized probe*. Put simply, **Probe** can be sensitive to more than just the *class* of feature - it can be specified for actual *values* of features - or at least classes of values. The idea is that there is some kind of feature geometry - familiar from Phonology - where certain values of features are represented as subsets of the larger feature class. A probe can be specialized to look for only features of the subset. In this case, the **Probe** in AF constructions is specialized for a more specific subset than just ϕ. Simplifying somewhat, the probe looks for a subset of ϕ that is specified for [**author**] - the name chosen for "not 3rd person," more or less. If it does not find a match, it surfaces in its default form - which is to say, 3rd person agreement in this construction in this language marks not agreement but rather a *failure* to agree. The preference for plural over

singular is implemented by assuming separate **Probe**ing heads for **number** and **person**[28].

Summary

Minimalism admits of only one structure-building operation - **Merge** - and this operation is in principle unbounded. Any two syntactic objects can be freely combined into a new structure. Taken in isolation, this defeats the purpose of grammatical analysis: it is largely the job of a grammmarian to say what combinations are *not* allowed. This is the function played by **Agree**. If **Merge** allows any combination, **Agree** at least defines what the consequences of the combination are. In a great many cases, the consequences are such that they are not useable by the interfaces, and this is the locus of grammatical explanation in Minimalism.

This chapter has shown a progression away from a checking concept of **Agree** toward one concerned with valuation(/specification). In more general terms, it is a move away from licensing and toward one concerned with *effects*. **Agree** records the effects of an operation, and this, in turn, helps determine whether a given series of operations is ultimately licit.

Based on a necessarily broad empirical survey, it is clear that a modern Minimalist toolkit must include a collection of mechanisms for agreement. These mechanisms must:

1. Operate over *features* as a medium
2. Be constrained by *categories*
3. Involve *valuation* - including sensitivity to previously-defined values
4. Be able to operate at a distance - put differently, they must be able to skip interven-

[28]The idea that these are separate **Probe**s suggests that in the case where a 3pl argument is higher than, say, a 1sg argument, the agreement morphology on the verb should be 1pl rather than 1sg, but Preminger does not address this question directly.

ing/irrelevant items

5. Operate over binary pairs - as identified by polarity/valence specified on the features, and in the roles of *probe* and *goal* assumed as a consequence
6. Trigger further operations as "follow-on" effects or side-effects

Chapters 4 and 5 will demonstrate how all of these concerns are addressed.

CHAPTER 4 - IMPLEMENTATION

PURPOSE

As noted in the introductory chapter, the Minimalist Program lives in a kind of purgatory between theory and program. In principle, it is nothing more than a set of research guidelines for a particular domain of study we might term "Cognitive Linguistics." In practice, for reasons both conceptual and historical, it is somewhat more than that.

Conceptually, its foundational texts (Chomsky 1993; Chomsky 1995) present a rationale for its approach that is grounded in a particular (architectural) model about the position language occupies in the human cognitive system. However abstract, and no matter that it demurs from making concrete predictions about how language is *actually* situated in the brain, it is a model notwithstanding and as such constrains the range of objects and mechanisms that researchers working in the Minimalist Program are likely to propose in their theories. Historically, it is grounded in, and largely a rethinking of, previous work in *Government and Binding Theory*. This likewise strongly informs the range of topics that Minimalist researchers concern themselves with.

If Minimalism is not strictly a theory, calling it a "program" can sometimes feel like a technicality. It seems to actually be somewhere in between: too specific to be a program, but not constrained enough to be a theory. It would be more accurate to say that it is *aiming* at a particular theory that cannot yet be tested, given the nature of the domain it studies and the current level of technology available for observing objects in that domain. All that can be done for now is to try to imagine what the Human Language Faculty would look like if the theory were true, and then try to fit the lingusitic phenomena we observe to that hypothetical

set of machinery. It is believed that successes and failures are equally informative in this enterprise: successes in advancing understanding of what it would mean for the theory to be true, and failures in learning where reality is likely to deviate from the theory. "Is likely" is used advisedly: in the strange paradigm Minimalism operates under, where conjectures can be lent credence by data but never properly verified, a failure to fit data to the target theory can as easily mean an *actual* deviation from a "perfectly designed" language faculty as a misconception on the part of the researcher about what "perfect" would mean in this domain.

In an important sense, what Minimalism is doing is reverse engineering the brain. It looks at "output" (at least, a certain domain of "output") and tries to imagine how the machine that produced it would have to be constituted to get just this range of data. "How the machine is constituted" can be almost arbitrarily abstract. The Minimalist researcher need not concern himself with the actual neural mechanisms of human language - indeed, it would be misleading to do so[29]. It's more a question of high-level understanding. Even so, there is no better proof of understanding than the ability to make it concrete, and there would be no better test of Minimalist success than the ability to build a machine that produces exactly the same output, if not necessarily in exactly the same way. The purpose of this project is to take the first steps toward building a toolkit that allows researchers to do exactly that.

INTRODUCTION

The previous two chapters gave a necessarily cursory survey of conceptual/explanatory machinery that is in popular use in the Minimalist program and would be available in any

[29] *Misleading* because the brain does more than process language and uses the same neural-computational operations to implement every domain of knowledge it covers. Learning about language processing by studying neurons is like learning about spreadsheets by studying microchips: it's not that the knowledge is irrelevant so much as that it is at the wrong level of abstraction.

grammar-building toolkit if it is to be useful to researchers. As this project develops, more mechanisms may be added. The ones represented for now are chosen to achieve maximal coverage of theories currently popular in the literature. The following sections give a survey of how they are represented in the system and how researchers would go about assembling testing environments for their individual theories out of them.

ALTERNATE SYSTEMS

Before getting into that, however, it would be remiss not to mention alternate approaches to this problem. Broadly speaking, there seem to be two.

Minimalist Program Grammar Implementation

The *Minimalist Program Grammar Implementation* of (Fong and Ginsburg in preparation) is the closest in spirit to this project. Like this one, it is primarily concerned with building a publicly-available system that people can actually use in their research. Like this one, it considers computational implementation the "proof in the pudding" - a (minimalist) theory is on the right track to the extent that it *verifiably* produces linguistic output consistent with people's actual intuitions. Like this one, it is a perpetual work in progress, starting by implementing the core mechanisms that underly most or all minimalist theories and expanding from there. And like this one, it seems to have identified **Agree** and its implementations as the prime computational engine and locus of difference between various minimalist theories. The most recent work by researchers associated with this project seeks to provide a full accounting for and implementation of **Probe-Goal** agree systems.

It departs from this system in being focused on efficiency and parsing. Where this project aims to be a grammar-building toolkit that researchers can use to test their theories against a common database of example sentences, Fong and Ginsburg's system seems more focused on building an industry-applicable parser. For this reason, the focus is more on efficiency than range of coverage. The system is grounded in Minimalist research out of a concern for cognitive linguistic accuracy (over and above mere machine usability), but it still wants its accurate results in a manner that is maximally efficient for silicon processors. As a consequence, there is a different principle guiding the choice of implementational mechanisms, and it is aimed at a different class of users with a different class of concerns. Practically speaking, this means it offers a more limited range of implementational mechanisms. Where this system aims to offer an API[30] of known grammatical mechanisms, the *Minimalist Program Grammar Implementation* settles on a more specific version of Minimalism that it believes to be correct, at least in terms of parsing efficiency.

Minimalist Grammars

The *Minimalist Grammars* approach of (Stabler 2001) and (Harkema 2001) is somewhat more removed from this project. Whereas this project and the *Minimalist Program Grammar Implementation* are concerned with tool building, the *Minimalist Grammars* project is a research program in its own right.

Minimalist Grammars are rigorous formalizations of a subset of the theoretical mechanisms

[30] "**A**pplication **P**rogramming **I**nterface." An API is a collection of subroutine definitions and use protocols that "exposes" the functionality of one program for use by an outside user (which might itself be another program). The "outside user" anticipated in this case is, of course, a syntax researcher working in the Minimalist tradition who wants to (a) verify that his theory derives all and only what he claims, (b) expose any unintended consequences his ideas may have introduced for the theories of others and (c) explore the consequences of (seemingly) minor changes in his theory - i.e. what structures would be had some choices been made differently.

proposed under the Minimalist rubric. No attempt is made to formalize the entire field - rather, those mechanisms are chosen which lend themselves to formalization (and which are in common enough use to plausibly be called "core" operations), and they are given formal grounding. This is done in the hope that it makes the Minimalist Program easier to reason about. To the extent that such formalizations aid the creation of Minimalist tools, this is of course viewed as a success, but it is also not the main point. The main point is to give Minimalist conjecture theoretical footing so that it can be reasoned about like a theory. One might say that this approach is more concerned with depth than breadth.

OBJECT AND MECHANISMS

This project takes as its inspiration the LKB (Linguistic Knowledge Builder) system of (Copestake 2002) and its associated English Resource Grammar (Copestake and Flickinger 2000; Flickinger, Bender, and Oepen 2014). Its aim is to build an analogous system for Minimalism. In an important sense, this goal is unattainable - for where "unification grammars" are a formalism, Minimalism is merely a research program. Unlike with *Head-Driven Phrase Structure Grammar (HPSG)* - the research paradigm targeted by the LKB - there is no authoritative Minimalist guidebook one can look to for the core mechanisms. The best one can do is to read the prominent papers in the field and mimic their systems computationally. In what follows, these core objects and mechanisms are distilled from those presented in the overview given by the previous two chapters.

Lexical Features

(Chomsky 1995), and nearly all subsequent work, conceives of Lexical Items - the atoms of a derivation - as bundles of linguistic features. Any grammar implementation in Minimalism must therefore begin by laying out the inventory of features that will be used. In an important sense, Lexical Features are the primitives of any minimalist grammar.

This implementation represents lexical features with a class `feature`. `feature`s are defined by:

Category The class to which a feature belongs. This governs to a large degree what features it can interact with.

Polarity Features in Minimalism come in paired oppositions. For each category, there are two classes of instantiation.[31]

Value Features have values (or don't). It will be assumed here that all features are *potentially* valued in the sense that all features come with a place to store a value. Whether or not a value can be transmitted or received for a given feature is a degree of freedom in the system.

In the current implementation, this is associated with a paricular machine-readable notation that is standard in the literature:

$polarityCATEGORY[value]$

Polarity is a lower-case letter - usually u or i[32].

[31] EPP features form an obvious exception to this, as they are (apparently) unipolar. As will be seen shortly, however, it is possible to bring them under the standard bivalent paradigm, and this project will do so.

[32] To indicate **i**nterpretable or **u**ninterpretable, legacy terms from the early days of Minimalism

Category is string of letters at least the initial of which is upper-case.

Value is a string and is enclosed in brackets. An *unvalued* feature marks its status by having the empty string between the brackets.

Examples of possible representations include:

A valued (accusative) case feature: $uK[ACC]$

An unvalued ϕ feature: $u\phi[]$

An (inherently-)valued ϕ feature: $i\phi[3sg]$

Lexical Feature Specification File

The range of features allowed for a given grammar is laid out in a user-defined `feature specification file`.

To capture the insights in (Preminger 2014), (Nevins 2007) and (McGinnis 2005), among others, this file is arranged in a *feature geometry*. The file should be in JSON (Bray 2014) format and should be represented as a single root object. Categories are arranged hierarchically as the keys of objects both at the root level and of objects nested inside other objects. The value associated with a category key is either itself an object keyed by category, or it is a `specification object`. A `specification object` is an object with a single key: `value`. `value` is typically a list of `string`s. A category definition embedded in another category definition is a *specification* for the purposes of standard feature unification algorithms such as those used in HPSG (see

that have largely been replaced by *valued* and *unvalued*.

(Sag, Wasow, and Bender 2003)). The format of `feature specification file`s is best thought of as a tree that represents a *feature geometry* (a la (Clements 1985)), with `value` arrays as a shorthand method for spelling out the leaves/terminals.

An example `feature specification file` is shown below:

```
{
  "Syn": {
    "Phi": {
      "Person": {
        "value": ["1", "2", "3"]
      },
      "Number": {
        "value": ["sg", "pl"]
      }
    },
    "Case": {
      "values": ["ACC", "NOM", "DAT", ...]
    },
    "Selection": {
      ...
    }
  }
}
```

One thing that might stand out about this example file is that `Person` and `Number` are separate

features, begging the question of how one implements the ϕ probes popular in the literature. This is done for compatibility with analyses like (Preminger 2014) and (Béjar and Rezac 2003) in which these features are treated separately, but of course it is up to the researcher-user to make these kinds of decisions in the space the system allows. It is of course possible to collapse them into a single ϕ category by replacing the implementation object for "Phi" in the file above with a `specification object` that is the concatenated powerset of the two `value` arrays, like so:

```
{ ... ,
"Phi": {
  "value": ["1sg", "2sg", "3sg", "1pl", ...]
  }
}
```

Shortcut Values

To save space and tedium in `feature specification file`s, the system provides two shortcut values that stand in for a range of things that could have been added to the feature geometry to acheive certain effects. These are **wildcard** (*) and **sink** (#). **wildcard** automatically matches any value, including **category** specifications. **sink** automatically *fails* to match (is automatically *distinct from* any match, technically). These are meant to cover situations where achieving certain effects requires a large number of single-use entries at every point in the hierarchy. So, for example, implementing a *barrier feature* to block any probe is normally a lot of work, because it requires entering a special-use "stop value" - a value that represents no linguistic specification - as a leaf of every category branch in the hierarchy. Repurposing

the earlier `feature specification file` example, to implement a head that blocks all Case and ϕ probes, one might do something like this:

```
{
  "Syn": {
    "Phi": {
      "Person": {
        "value": ["1", "2", "3", "#"]
      },
      "Number": {
        "value": ["sg", "pl", "#"]
      }
    },
    "Case": {
      "values": ["ACC", "NOM", "DAT", "#"]
    }
  }
}
```

And then, on the head in question, there would have to be "sink" features - like so: $iCase[\#], iPerson[\#], iNumber[\#]$. This guarantees that any probe of these categories will match and meet with failure, since the sink feature is meaningless, but the probe is technically now valued and thus inactive. Suppose instead one wanted to block *all* probes (as is true at phase boundaries, for example - see chapter 2). In that case the shortcuts save a lot of typing. The "barrier feature," to coin a term, is just $i[\#]$. It matches any category and transmits a

useless value.[33]

As a sidenote, any applicability of the *barrier* feature to systems like (Chomsky 1986a) is not a mere coincidence. It has been widely noted that Minimalist *phases* capture a lot of the same intuitions, and the shortcuts were added to the system precisely to make *phases* easier to implement without having to build them in as primitives. The name reflects this heritage.

Lexical Item

A `lexical item` is nothing more than a selection of features (Chomsky 1995). Ideally, then, `lexical item`s in this system would just be arrays of features. For implementational convenience, they are a bit more articulated than that in the present system. `lexical items` consist of:

Tag a string that uniquely identifies the item[34]. No two items in a lexicon may have the same `tag`.

Phon a string representation of phonological features

Syn a set of syntactic `lexical features`

Sem a string representing a semantic denotation (if any)

`Syn` consists of features in the sense of `lexical features` defined above. At the time of writing, `Phon` and `Sem` are not fleshed out. **SpellOut** in the present system is just string

[33] One may well wonder why not just leave the value unspecified on the "barrier feature." That is certainly also an option. The trouble is that in some systems - especially multiple agree systems (Hiraiwa 2005) - one may wish to draw a distinction between valued and unvalued **Goal**s.

[34] *Uniquely identifies* with respect to other items in the lexicon. This is not the same as an *index* (or memory address), which uniquely identifies a *token* of the item participating in a derivation.

concatenation, so `Phon` for lexical items is its surface string. It remains for future work to articulate this further (see (Herring 2009) for a preview). `Sem` is also not articulated at the time of writing, but the intention is to leave this largely to user implementation. More details are given in later sections, but the system includes "attachment points" at which a user-defined semantic implementation can be "plugged in." To delegate maximal responsibility to the external semantic system, semantic features are currently expected to be strings. These strings can represent whatever objects the semantic plugin understands, and it is the responsibility of the plugin to parse and assemble them. It is anticipated that in the general case these will be semantic *denotations* to be combined in a manner determined by the algorithm that reads them from the output tree structure (see (Heim and Kratzer 1998) for a compatible system that is popular among Minimalists).

Lexical Item Token

A `lexical item token` is a syntactically independent copy of a `lexical item` that participates in a derivation. It differs from a `lexical item` only in terms of having an `index` - an `integer` that is incremented each time a uniquely-identified `lexical item` is **Select**ed from the **Lexicon** into a **Derivation**. This roughly mirrors the *numeration* of (Chomsky 1995). The indices serve to keep instances of `lexical item`s distinct[35].

Lexicon

A `lexicon` is a set of `lexical item`s. Ideally, it represents an entire language. In practice, it will typically be just as many items as needed to test the implementation of a particular

[35] *Distinct* from the user's point of view. The system keeps track of them by their address in memory - i.e. this is a *multiattachment* system by default.

theory.

`lexicon`s are represented by `lexicon files`. Like `feature specification file`s, `lexicon file`s consist of a single JSON object. In this case, the object has a single key - `lexicon` - the value of which is an array of JSON objects representing lexical items. The system will check to make sure there are no duplicates and that all items are correctly specified. Items loaded into memory via a `lexicon` are available to the system for use in **Derivation**s.

A sample lexicon file for a highly simple grammar that generates a single sentence[36]:

```
{
  "lexicon": [
    {
      "tag": "nuts_1",
      "phon": "nuts",
      "sem": "",
      "syn": ["cN[nom]", "iH[3pl]"]
    },
    {
      "tag": "parrots_1",
      "phon": "parrots",
      "sem": "",
      "syn": ["cN[nom]", "iH[3pl]"]
    },
```

[36]The individual feature representations here owe a great deal to (Stabler 1997) and subsequent work in the *Minimalist Grammars* tradition. Though they are more articulated than Stabler features in ways detailed previously, the choice of `=` to mark a `selectional` feature is, for example, borrowed from the work of Stabler and colleagues.

```
{
  "tag": "det_indef_pl",
  "phon": "",
  "sem": "",
  "syn": ["cD[indef]", "=N[]", "uC[]"]
},
{
  "tag": "like_1",
  "phon": "like",
  "sem": "",
  "syn": ["cV[like]", "=D[]"]
},
{
  "tag": "v_trans",
  "phon": "",
  "sem": "",
  "syn": ["cv[trans]", "=V[]", "uH[]", "iC[acc]"]
},
{
  "tag": "T_pres",
  "phon": "",
  "sem": "",
  "syn": ["cT[pres]", "=v[]", "=D[]", "iC[nom]"]
},
{
  "tag": "C",
```

```
      "phon": "",
      "sem": "",
      "syn": ["cC[decl]", "=T[]"]
    }
  ]
}
```

Lexical Array

A `lexical array` is an object that holds only those items that will participate in a given **Derivation** - i.e. is an array in sense of (Chomsky 2001). Items selected from the Lexicon into a `lexical array` are instantiated as `lexical item tokens`.

Syntactic Object

`syntactic object` is a type that subsumes `lexical item tokens` and combinations of `lexical item tokens` - i.e. `lexical item tokens` that have been **Merge**d (in a process to be defined below). `lexical item tokens` which are selected from the `lexical array` of a `derivation` are automatically converted to a `syntactic object` type.

The salient difference between `syntactic objects` and `lexical item tokens` is that `syntactic objects` are recursion-enabled. They contain, as part of their data structure, a `constituent list` of other `syntactic objects`, which in the case of a head obviously contains its **Comp** as well as any **Spec**s.

In the interest of keeping the system maximally general, **Spec** and **Comp** are not explicitly distinguished in this system the way they are in, say, the LKB, or more recent versions of HPSG. Rather, the system relies on the insight that in any pairwise **Merge** system it will always be the case that there is a sequence to the list of items that have been **Merged** with a particular `lexical item token` head. In principle, individual grammar implementations are free to ignore this, disregarding the ordering for any user-scripted functions. In this way, shallow-tree systems (of the kind detailed in (Culicover and Jackendoff 2005), for example) are implementable in the system - one need only ignore the ordering of the constituents associated with a phrase. But since the system is intended for use by researchers working in the Minimalist Program/GB tradition, it is expected that the overwhelming majority of use cases will choose to write rules that treat the first item in the internal constituent list - the **Comp** - considerably differently from the remaining items.

`syntactic object` also encapsulates - in defined interface methods - much of the functionality involved in combining with other `syntactic objects`. This list includes:

Immediately Contains an interface function that inspects the `constituent list` to determine whether the queried item is a member.

Contains the recursive extension of `immediately contains`: an item `contains` another item if it `immediately contains` it or a member of its `constituent list` does. C-command is implemented by restricting `contains` queries to the first item in the `constituent list` - the **Comp**.

Merge takes another `syntactic object` as an argument and returns a new `syntactic object` built from the one on which the method was called, and the argument[37].

[37] This is not a destructive operation. The constituent `syntactic objects` from which the new one is formed are contained within - implementationally as pointers, so in computational terms they are *referenced by* - the created object and are still available to the system for further operations.

Project determines the side-effects of **Merge**. Technically this is a system "hook" - a point where a user-scripted function can be inserted to specify the next action the system should take. But the obvious candidates are provided out of the box. These are `append` (put the argument on the end of the calling object's `constituents list`), and `donate` (a convenience implementation of feature inheritance per (Chomsky 2008) - see below for more details).

Workspace

This concept is adapted from (Collins and Stabler 2016) by taking the representational idea from that system and embellishing it with functionality.

A `workspace` in this system is an implementational primitive and should not be confused with the **Workspace**s of (Nunes 2004) or (Hornstein, Nunes, and Grohman 2005), though they can be leveraged for the same purposes[38]. At their core, though, `workspace`s provide a way of marking items as *active*[39] in a derivation as well as an interface for executing derviational actions.

A `workspace` in this sense consists of a list of `syntactic object`s and a system of interface functions related to their derivational status. In the general case, a `workspace`'s list will consist of either one or two objects. If two, then one that is the output of previous derivational

[38] Workspaces in (Nunes 2004) are theoretical constructs that allow for the possibility of parallel sub-derivations, which in turn allows for the possibility that items from one sub-derivation can participate in another in ways that would otherwise be countercyclic. There is not space to go into the complexities of this proposal here; suffice it to say that although Workspaces in this system do not directly model the Workspaces of (Nunes 2004), Nunes-style system coulbe implmented using them by expanding Stages to host multiple Workspaces (and multiple LexicalArrays).

[39] *Active* does not mean anything particularly special - the "active" item is simply the one the system will operate on next, allowing it to `select` and `probe` to find matches for its own active `feature`s, possibly `merge`ing with other items as a side-effect.

steps and one that is recently-**Select**ed - whether from the relevant `lexical array` or as a result of a **Probe** on the other object. If one, then either the initial **Select**ion from the `lexical array` or the (possibly convergent) output of previous derivational steps.

However, this assumes a fairly constrained algorithm for **Select**ing items from the `lexical array` - one that is informed by the features on the previously-derived or previously-**Select**ed member of the `workspace`, perhaps - and this need not be true of all implementations. Thorough checking for overgeneration in a grammar may wish to allow for random selection from the `lexical array`, just to see what the result is. In this case, it can happen that two in-principle-uncombinable items end up in the `workspace` at the same time, and it is necessary to **Select** again to proceed. In such cases, there will be more than two (perhaps many more than two) items in the `workspace`. It is really only true for convergent `lexical array`s that the "general case" involves `workspace`s of one or two items.

`workspace`s are not mere containers. They also implement a number of useful interface representations and functions.

Representationally, because `workspace`s are *ordered* arrays of items - that is, since **Select**ed `syntactic object`s are always appended to the end of the array, the first item in the array has a special status, which we'll call *active*. This means merely that whatever operation is attempted will start by calling the relevant interface function on the *active* (leftmost, 0-positioned) `syntactic object` and only proceed to the next if it fails on the first. The utility of this will become easier to understand after reading the derivational example section (Appendix A), but in simple terms, the current active head of a derivation should always be first-selected into the `workspace`. It is not that the derviation will fail if this is not the case, it is just that the system will waste a step determining that the left-most item in a given `workspace` can't actually select the item to its right, that this must be retried the other way

around, etc.

Functionally, `workspace`s provide interfaces for the derviational algorithms to use. Some which are exposed include:

is_root which determines whether a given item in the `workspace` is a root in the workspace (i.e. it is there because it is expressly in the list of contained `syntactic items`) or is contained within a root in the workspace (i.e. is a candidate for **Move**)

is_simplex which determines whether a `workspace` has a single member or multiple members

merge instructs the `workspace` to implement whatever **Merge** algorithm is defined on it. In the general case the most active item (the leftmost in the `workspace`'s array) goes through each of the other items in the array in turn (in the general case, there is of course only one) until it finds (or fails to find) something that it can **Merge** with. This returns an updated version of the workspace.

probe instructs the `workspace` to try to establish a relationship with another item. This amounts to passing a message on to the leftmost item in its own array to **probe**. In systems that adhere to the **merge-over-move** constraint, items should first **probe** in the relevant `lexical array` before attempting to look into their own arrays of items. **probe** passes a reference to the `probe`ing feature into each `syntactic object` considered's own `probe` method, and this method, in turn, allows for recursive `probe` (in the case where the target `syntactic object` is not simplex). As detailed in chapters 2-3, this is largely the way the system enforces locality constraints.

The reader may notice that **move** (**re-merge**) and **value** are not on the list of interface functions. That is because they are expected to occur only as side-effects of **probe** and so are not directly callable on a `workspace`.

Stage

A `stage` is another implementational primitive adapted from (Collins and Stabler 2016) and is essentially a "state manager" for derivations. Like with `workspace`s, where the Collins and Stabler version is mostly representational, `stage`s in this system also incorporate functionality.

`stage`s contain the `lexical array` for a particular section of a derivation[40] and at least one `workspace`. This could just as well be "at most one `workspace`" but for proposals by (Nunes 2004), (Bobaljik and Brown 1997) and others that movement of objects between "simultaneous" derivations - originally called "interarboreal movement," now more commonly refered to as "sideward movement" - is possible under certain circumstances. Additionally, it is convenient to assemble adjuncts in this way (as they are typically assumed to be fully-formed by a separate process before attachment to the main "trunk" of - and subsequent involvement in - the derivation).

A `stage` can be thought of as a single (automata-theoretic) "state" in a derivation, and final convergence depends upon a `derivation` object consisting of a single active convergent `state`.

A given `stage` represents the *start state* when

1. It has a non-empty `lexical array` and
2. It has no non-empty `workspace`s

A given `stage` represents a *convergent* state when

[40] In systems that employ phases - at least as described by (Chomsky 2001) - this will be the *lexical subarray* associated with that phase rather than the full `lexical array` that triggered the derivation.

1. It has an empty `lexical array`
2. It has a single, non-empty, *simplex* `workspace` in which all unvalued features on member `syntactic object`s have been valued

In the case where a probe had failed to find a match, indicating divergence, one of two things would have occurred. An inadequate target might have been found - an intervention effect - in which case the target would have been added to the `workspace`'s working array but never **Merge**d or **Value**d, and so never reintegrated into the active object. Alternately, *no* target may have been found, in which case there are un-**Select**ed items in the `lexical array` (because the derivation will be unable to proceed after no suitable connection is made by the **Probe** - it "crashes"). The requirement that a `workspace` be *simplex* for *convergence* arises from the need to detect a crash in the first case, and the requirement that the `lexical array` be empty for the second. Otherwise, the state *diverges*. Note that it is in keeping with Minimalist principles to regard all intermediate states in a derivation as *divergent*. Minimalist conjecture is that the purpose of C_{HL} is to establish interface-readable assemblages of lexical items, and Narrow Syntax is the engine that does this. By assumption, no work-in-progress is (yet) readable by the interfaces and so is *divergent* by definition. Therefore, `stage`s - including the *start state* stage - are either *divergent* or *convergent* and never anything else.

`stage`s function as "state managers" for derivations in the sense that they accept and execute instructions and report back on the effects, in part by deciding on a `string` from a list of acceptable state labels to associate themselves with. The decision function itself - called a `strategy` in the system - reads the new state off of the stage and creates the next stage from it, at which point the cycle may repeat itelf by calling interface functions on this new `stage` and so on. There is, of course, a default fallback implementation, to be detailed and demonstrated below, but in principle the usefulness of the system comes primarily from allowing user lexicon

and feature specification, and secondarily by allowing user modification of the cycle algorithm by supplying a user-defined `strategy` function.

The Cycle

The Cycle is the process that a derivation goes through at each stage to edge closer to convergence. Recall that Narrow Syntax is tasked with assembling items in a `lexical array` into a single-rooted object that the interfaces can "read." It goes through a series of steps to achieve this goal. At each step, it finds itself in a state that is either *divergent* or *convergent*. As explained, the vast majority of these are *divergent*. Some of them are *irreparably divergent*, in which case the derivation fails - is said to "crash" in the jargon. If a derivation finds itself in a state that is *irreparably divergent*, it is in a failed state and should halt. If it finds itself in a *convergent* state, it is in a success state and should also halt (though it can form part of the input to further derivations). Otherwise, it carries out an implementation of the cycle to determine what action to take next and then to take that action.

The default cycle is simple:

1. Activate
2. Perform
    i. Designate
    ii. Probe
    iii. React
3. Evaluate

Activate decides on a `syntactic object` as the actor in the next cycle. This is user-

scriptable, but in the default case it is simply the leftmost object in the active `workspace`s array followed by, if nothing can be done with the leftmost, the next one and so on down the list until there are none left. This will suffice for almost all implementations. But more complex implementations - for example, those that involve multiple workspaces - may exercise more control over this process.

Perform is a loop that repeats three steps until nothing more can be done. **Designate** decides on a **Probe** on the active item to try. In the general case, this is the first *active* (in the sense of (Chomsky 2001)) probe in the active `syntactic object`'s lexically specified list of features. **Probe** implements the process of looking for a match for the probe's features. In **merge-over-move** honoring systems, this means it will first look for matches in the `lexical array` and only then, when none are found, attempt to probe its **Comp**. In the default algorithm, the **Comp** - i.e. the *first* item in the active `syntactic object`'s constituent list - is the only item tried outside of the `lexical array`. In more sophisticated systems, other things are possible. For example, a probe could search in another workspace, if one is present (implementing what is sometimes called interarboreal or, more colloquially, "sideward" movement), or it could proceed down its own constituent list (effectively **Probe**-ing its specifiers). After **probe** has completed, the system moves on to **React**, which implements any side-effects of probe match. If the probe is unsuccessful, in a standard (Chomsky 2008) system **React** should throw an error, aborting the derivation.[41] Otherwise, the item is copied into the active objects array in the `workspace` to mark that a relationship has been established between the **Probe**-ing item and this target. From there, the system will **value** any items in the `workspace` array according to their array of features - as membership in the array marks that they are in a relationship with the active `syntactic object`. **value** itself will

[41] As in this system *phase* heads are the only locus of uninterpretable features, and failure to find a match in one's own domain means there will never be one.

have side-effects, which in the general case means removing the item from the array.[42] But this will not always be the result. In the case where the item came out of the `lexical array`, chances are it was as the result of a **Select** feature, in which case **value** will instruct the system to **merge** as a side effect. This is also the point where EPP features come into play. EPP, in this system, is effectively an override instruction to **merge** where the system would normally **return**[43].

Perform iterates until all active probes have been tried.

Evaluate fires after the **Perform** loop exits. It is tasked solely with deciding on what state the stage ends in, and what to do about it. If errors were thrown in the **Perform** loop, it "handles" them, generally by aborting the derviation with a printed explanation of why it failed. Otherwise, it checks the state conditions outlined above and decides which applies. It then returns a copy of itself[44] which will be the input to the next cycle.

The apparent simplicity of this process of course conceals a lot of implementational details. For example, on **probe**, it conceals the order in which to attempt each of the active `syntactic`

[42]This does not delete the item. Implementationally, these are pointers, so removing it from the array effectively "returns" it to its position lower in the hierarchy, in the case where it was found in the active `syntactic object`'s **Comp**.

[43]One could also think of this the other way - that EPP is not itself a feature but rather a class of features (or perhaps a diacritic on a feature) that interrupts long-distance **agree** in favor of **merge**. Seen this way, it is the fact of language which is responsible for selection of items from the `lexical array` and into the derivation in the first place. This idea would have to be explored, but it holds the promise of making the EPP somewhat less mysterious. The question of *why* there is an EPP is answered - it just *is* select - and the central question about the EPP would be rephrased as "why does **select** happen in these cases where it is not strictly necessary?" It would effectively set the traditional way of looking at EPP on its head.

[44]Computational note: although pointers are used in many parts of the implementation, stages are "purely functional data structures" in the sense of (Okasaki 1999) - that is, any changes to their state are implemented by copying the stage. This serves two purposes, one future and one present. The present purpose is that it memoizes the derivation, keeping a full record in memory for treatment by the interfaces, and also the user display output systems. The future purpose is that it makes for easy introduction of first-class *continuations* - in the sense of (Strachey and Wadsworth 1972) - into the process. There is no actual evidence that these are useful yet for natural language syntax, the author is simply indulging a hunch that they will prove so.

`object`'s potentially numerous **Probe**s. There are plans in the future to allow users to intervene in this process (by, e.g., allowing conditional rules on probe ordering, or access to workspace selection, etc.). For now, it is assumed that probes are tried in the order in which they are specified on the `lexical item`. Additionally, determining which **probes** match what is a bit complex. This, again, is user-scriptable. In the default case, it is much the same as HPSG in the sense that it obeys the `feature geometry` outlined in the `feature specification file` according to standard unification algorithms. A probe **match**es a goal of opposite polarity if the goal has a category that is identical to its own or else rooted lower in the hierarchy. So, "Feature," the top level, matches any category, and a highly-specific one, like "Author" (see (Preminger 2014)) pretty much only matches other "Author" features. Another wrinkle is that some systems wish to intervene on **match** and prevent **value** in some way. For example, for the purposes of clitic doubling in a number of systems, it's assumed that **Dative** nominals can't value a probe, but they can copy full sets of ϕ features, which show up as a clitic. No attempt is made to try to implement such a complex system here, it is merely noted as justification for treating **value** as a system "hook" where user-defined functions can be inserted.

FEATURE INHERITANCE

Before moving on to the `derivation` class, it is probably worth taking a detour into the **feature inheritance** of (Chomsky 2008). Of all the mechanisms surveyed in previous chapters, this one is perhaps the biggest puzzle.

To start with what is clear - whatever **feature inheritance** is, it must be a side-effect of **merge** and not **value**. We know this is so because the whole point of it is to give **T** countercyclic **probe** features. "Countercyclic" means giving **T** an opportunity to **probe**

after the point in the cycle where it should have been allowed to - which is really just a fancy way of saying that **T** should be allowed to participate in **C**'s **probe** cycle using **C**'s feature. But putting it just that way begs the question of why this can't just stay **C**'s cycle. Presumably, the answer is one or both of that *wh*-objects need to be **Probe**d first (to obviate the intervention effect and make the subject visible to **T**) - and **C** probes go first since it's "**C**'s turn" - or that the subject needs to end up in **T** for various other reasons (articulatory or scopal). Either way, we need more control over the implementation than comes with the default algorithm. To solve the probe-ordering issue, there would need to be some mechanism that relegates the (now-)**T**-associated **probes** to the end of the list. To solve the positioning issue, the probes would have to be marked in some way to indicate that any (re-)**merge** that takes place would have to adjoin to **T** rather than **C**. The problem with this, of course, is that **T**, having already **merge**d with **C**, is not in the array of `workspace`-active `syntactic objects`, breaking the standard way of achieving this goal in this system. Now, there's nothing in principle wrong with having a side-effect of **merge** that leaves the target "active" in the sense that it sits in the array of active `syntactic objects` even after having been **merge**d. But in practice keeping **T** separate breaks the **probe** path, since **C** doesn't yet have an explicit **Comp** (since **merge** wouldn't have completed, as the pair of involved items is still represented separately in the **workspace**). It's like **C** needs to have **T** in its **Comp** at the same time that **T** is in the active array.

The solution is just to do that literally. **Feature inheritance** is implemented as a kind of *incomplete merge*. Since the `syntactic objects` that are in the active array are pointers, there is no problem with adding **T** to **C**'s constituent array (in the first position, as a **Comp**) while also leaving it in the active array. It is both *active* and **merge**d all at once.

The question of how to implement the actual sharing part is not difficult to answer: just do

that literally as well. There is plenty of evidence from the literature to justify including such an operation (as a side-effect of **merge**) in the system. This includes things like pronouncing multiple intermediate copies of simplex *wh*-items in some Germanic languages (Felser 2004) - a fact that (Herring 2009) interprets as pronunciation of intermediate landing sites which have "borrowed" their phonetic features from the moving item. It also includes many analyses of clitic doubling (e.g. (Béjar and Rezac 2003; Preminger 2014)). And, there is also some evidence directly to do with feature inheritance. (Ouali 2008), for example, argues from agreement patterns in Berber - in which subject agreement sometimes appears on **C** - that **feature inheritance** can actually take one of three forms:

1. **Donate** - in which **C** simply gives its ϕ-features to **T**.
2. **Share** - in which the features are active on both **C** and **T** (presumably through a (Pesetsky and Torrego 2007) type of arrangement)
3. **Keep** - in which **C** doesn't transmit anything.

These situations correspond to varying agreement patterns.

If there are plenty of accounts in the literature that depend on the ability of a head to transmit features to another as a side-effect of **merge**, then clearly the system has to make this available as a mechanism to researchers. So, it is provided in the default cycle algorithm for **C** and **T** - to cover the (Chomsky 2008) argument - in the present version of the system, and later iterations of the system will make it modifiable by users.

The problematic bit that remains is "cleanup." After this implementation, **T** remains in the "active" array at the end of the phase, which means the `stage` will incorrectly think it's failed and return in an error state. There doesn't seem to be a good solution to this problem in the present system, so for now a stipulative "cleanup" phase is added to **Evaluate** that

detects this situation. In techical terms, **cleanup** checks to see whether the following situation obtains:

1. There are only two items in the active `workspace`'s active items array
2. One of these items is contained in the other
3. Neither of the items have active probes

If these conditions obtain, remove the contained item from the active array, and retry **Evaluate**.

HEAD MOVEMENT

Head movement is equally problematic to implement, not so much because the necessary primitives are not independently available in the system as that it isn't entirely clear from the literature what head movement is. Through most of the GB period and into early Minimalism, it was taken to be some kind of incorporation: one head raised to the next highest head[45] and in some important (if vague) sense *fused* with it. It wasn't clear whether the moved head left behind any kind of trace, but if it did it certainly didn't *c-command* it, making this kind of movement technically "improper" (see (Müller and Sternefeld 1993) for a detailed discussion of what counts as "improper movement," including examples that go beyond the *c-command* requirement, and (Fukui 1993) for an early Minimalist reworking of GB conditions on proper movement).

(Chomsky 2001) lists a number of characteristics of head movement that make it problematic as a syntactic phenomenon:

[45] Head movement was never allowed to skip heads; it was always strictly local (Travis 1984).

1. It never seems to affect interpretation (though see (Matthew 2015) and (Funakoshi 2013) and (Roberts 2011) for recent arguments that it sometimes does).
2. It is not clear what the trigger is. Relegating it to features would require the feature system to be able to distinguish between heads and phrases, which Bare Phrase Structure arguably tries to avoid
3. It is countercyclic and violates the Extension Condition.
4. It is improper movement as the moved head does not *c-command* its trace (at least, not without complications of the definition of *c-command*).
5. It is not successive-cyclic, but rather always "rolls up" (each subsequent movement always form a successively more complex head; it is not the case that a single head can move to one head position and then continue on to the next without taking the head at the preceding landing site along with it).

These form the motivation for rethinking head movement as an entirely PF process.

Since feature transmission in the sense of (Ouali 2008)'s **Donate** is available in this system, the suggestion is to use that. "Head movement" means that the lower head of the pair **Donate**s all of its features to the higher one. This is essentially "reverse feature inheritance." And since inheritance happens on **merge** using the same mechanisms that underly C-T feature inheritance, there is no need to complicate *c-command* to get the desired result: there is no question that the lower head is "visible" for feature sharing/donation/inheritance at the point the target head **merge**s with it. The operation is simple: *all* the features of the lower head, including `phonetic` and `semantic` features (if any) are "unshifted" onto to the front of their counterparts' values in the higher head (and deleted from the lower head). The lower head ceases to exist as anything other than a position in the hierarchy; all relevant information about it is realized at the analogous position one higher up.

This should accurately capture all of the empirical facts. It captures the pronunciation shift (including - thanks to the fact that these features "unshift" onto the front of the target head's array rather than concatenating on to the end of it - all observations underlying the "mirror principle" of (Baker 1988)), and captures the "rolling up" effect/lack of successive cyclicity. It does not capture any interpretive effects that may need to be accounted for, but this can always be remedied by specifying **Share** rather than **Donate** (so that syntactic features are represented in both associated places in the hierarchy, effectively collapsing them in syntactic terms, but semantic denotations continue to occupy separate nodes), or else distributing the relevant features between **Donate** and **Keep** (i.e. selectively deciding which go where).

SELECT

Nothing much has been said up to this point about **Select** - another mildly problematic category. The question with **Select** in any system is whether to treat it as a system primitive or to leverage features in its implementation. Like with **head movement**, there is no real consensus in the literature about how to answer this, nor is there much in the way of detailed treatments of the topic. Researchers tend to take **Select** as a given and gloss over its mechanics.

As has been hinted at in earlier sections, this system takes the view that **Select** is a part of the feature system - effectively a side-effect of **probe**.

The main wrinkle comes in trying to capture its optionality. While **Select**ion of arguments is (usually) required, adjunction is free and (apparently/often) also unconstrained with regard to number. And yet, it's clear that **Select** plays a role even in adjunction, since it is not typically possible to adjoin *just any* category to a given projection. The head has some say

in what participates in its phrase, even outside of its internal argument. This mirrors one of the puzzles of the EPP, alluded to earlier, whereby it seems to be highly specialized for the category it attracts, suggesting a featural implementation, and yet it does not seem to pair with other features (does not participate in the interpretable/uninterpretable polarity system).

Call it the No Platypus Principle, but unlike natural systems, computational implementations have to decide on an interpretation in situations like this, and this system chooses the feature route. Selectional features are features like any other, meaning they have polarity and come in pairs. In the case of optional selection, the **selector** is *uninterpretable* - since it acts as the **probe** - and the **selectee** *interpretable*.

It is at this point that the distinction between *uninterpretable* and *unvalued* pays dividends. If the interfaces are only concerned with feature *values*, and not with feature *polarity*, as seems reasonable, then we can leverage feature geometry to implement optionality. Optional **Select** is implemented with an *uninterpretable* but *valued* feature, where the value is present but underspecified. For situations like sequences of adjective phrases, where multiple are allowed, but there are nevertheless some constraints on in what order they adjoin, the implementor can simply define a feature hierarchy that takes care of this. The category values of the "closest" adjective types would need to be "highest" in the feature geometry tree, so that at each stage of selection, unification could take place and the value on the **selector** N head could get "more specific" (i.e. successively replaced with lower values in the hierarchy). However, since the N head comes already valued out of the lexicon for its adjectival modifier feature, the interfaces will not complain if it doesn't combine with any.

θ-selection would be handled more traditionally. An argument-seeking **selector** head would have an uninterpretable/unvalued selection feature, enforcing the obligatory nature of argument

selection. The target **selectee** argument would have an interpretable but unvalued θ feature, and an interpretable and valued selection feature - a value it transmits to its **selector** to mark that the **selector** has picked an argument. Getting θ-marked means having its θ feature valued by the corresponding version on the **selector**.

Any implementation that treats θ-marking as feature-driven is probably obligated to say something about the Movement Theory of Control (Hornstein 2000), since the idea that θ-marking is feature-driven is a hallmark of that theory, and the idea that allowing items to accumulate θ-roles overgenerates (without baroque prevention mechanisms) is the main argument against it (Landau 2003). In reality, there's no need to take a position on this here since the debate doesn't hinge on the mechanism of θ-role transmission so much as whether multiple θ-roles are allowed. Any system in which a valued feature can't be re-valued[46], or where accumulation of values is not allowed, will get the "standard" theory of control of (Landau 2003) even if θ roles are implemented with features.

PHASES

Surprisingly, given how much has been written about *phases* in recent years, little need be said about them here. A system such as this one is successful to the extent that seemingly radical departures from the basic system turn out *not* to need *ad hoc* solutions - meaning solutions that have to be implemented directly[47]. Phases in the (Chomsky 2008) and (Gallego 2010) sense, indeed in almost every sense, can be relegated to the lexicon.

[46] At least, not at the same level of the hierarchy. Since this system allowed feature geometry and computes "value" through unification, it is possible to be revalued at a different value IFF the new value is subsumed by the original value in the specified feature geometry. But that does not apply to any known θ systems, in which values are always mutually exclusive.

[47] Like the **cleanup** operation of the previous section.

The two salient points about *phases* for these systems are:

1. Phase heads are the sole locus of uninterpretable features AND
2. Phases are *impenetrable*

The first point is the responsibility of the lexicon writer, so any researcher who buys this analyses need only take responsibility for seeing that the lexicon he feeds the system contains no non-phase heads with uninterpretable features. In practice this will be impossible, since this system treats selectional features like any other. So, at least uninterpretable selectional features will have to exist on non-phase head. But this is not really a problem since it doesn't seem to violate the spirit of the proposal in any way[48].

The second point is a bit more complex, since there are several versions of the **phase impenetrability condition** that would need to be covered. In the basic case, any phrase in this system can be made "impenetrable" by including the "barrier" feature on the relevant head. Wherever the head **merge**d in **is** the barrier in question, the matter is as simple as including the "barrier" feature in the item's lexical entry. In more complex cases, where merger of the phase head causes the (complement of the ... but see (Fox and Pesetsky 2005)) *next lower* phase to be **spell**ed **out**, it is implementable via **feature inheritance** - transmission of the "barrier" feature can simply be included in the **merge** side-effects. So, phases turn out to be a derivative notion, and not really an implementational primitive.

[48]In no *phase* system that the author is aware of is **Select** limited to phase heads - nor is there any real reason to believe it would be all that problematic for it to be.

DERIVATION

`derivation` is the driver class of the system. It is responsible for setting things in motion and evaluating the result. It takes as input a `lexical array`, which can be an array of arrays. An initial `stage` is created for each subarray; if there are no subarrays, only a single stage is created. `derivation` then decides on which of the subarrays to start with and takes responsibility for selecting the initial item. Having done so, it places the active stage in a `history` array (that represents the derivation as a whole), sends a `probe` signal to it, and receives the result. As indicated, the result of sending a signal to a `stage` is always a new `stage` which is a copy of the original `stage` with all changes that occurred during its cycle applied. At a minimum, this means its state indication will have changed (because if no other changes occurred it is either convergent or there was an error). The `derviation` places the new `stage` on the `history` stack, inspects its state and decides what to do from there. It continues this process until there is a convergent state on top of the stack and no more `lexical array`s to account for, in which case it returns a *converge* message, or until it exits with an *error*.

DERIVATION EXAMPLES

A sample derivation of "Parrots like nuts" (from (Citko 2014)) is given in Appendix B along with some pointers for implementing other example sentences. Additionally, the following chapter demonstrates some ways in which the need to carefully implement a theory can expose unintended side-effects of its assumptions. A complete derivation of one critical example sentence from this chapter is available in Appendix A.

CHAPTER 5 - TWO CONTROL-BASED EXAMPLES

OVERVIEW

Having now spelled out the implementational primitives in some detail, what remains is to show their application to a practical example.

The utility in any grammar construction toolkit lies only secondarily in its ability to verify a researcher's claims about the coverage of his theory. More important by far is its ability to highlight areas where a given theory falls short. Often such shortcomings will be known and acknowledged by the researcher, as no theory to date claims to cover all or even most observed syntactic data. But in many cases they are either unanticipated or not (publicly) acknolwledged. Science proceeds by gathering data, reasoning to the best explanation, and then looking for failure - and it is typically in repairing failure that the most interesting and fruitful advances are made. A system such as the one under construction succeeds to the extent it can help researchers - or perhaps their rivals - identify points where theories break down, adjudicating, as it were, disputes about the right way to proceed in the development of a comprehensive theory of human grammar knowledge.

In this context, the Movement Theory of Control (MTC) (Hornstein 1999; Boeckx, Hornstein, and Nunes 2010) presents an attractive test case.

Three qualities in particular suggest it as a fecund patch in which to build and improve a grammar development system for the Minimalist Program: it is controversial, disruptive, and based in attempts to shrink the size of the universal grammar toolkit. Controversy guarantees attention, which in turn tends to generate detailed, quality research as a side-effect. The fact

that the MTC aims at shrinking the grammar inventory - specifically by eliminating PRO and the control module (construal), where possible - makes a grammar development toolkit a particularly nice fit, as the MTC project is, in essence, a "refactoring" of previous analyses. In such situations, it's nice to have a compiler around to verify one's claims, especially when, as is the case with the Movement Theory of Control, such claims are founded in the removal or relaxation of a number of constraints - most prominently the Theta Criterion - that served to narrow the possible outputs of the system. Any time one is removing constraints, overgeneration is a danger, and an automatic generation system can help to spot it early. The most attractive thing about the Movement Theory of Control as an area of focus in developing this system, however, is its disruptive nature. Innovation is born in disruption, and it is the relative freedom Minimalist researchers have to innovate that poses the biggest challenge to developing an LKB analogue for Minimalism. In a formalized system like Head-Driven Phrase Structure Grammar, the author of a software toolkit gets at least a blueprint of how the system should work from the framework architects. Minimalism, by contrast, is a program and not a theory, and it imposes no such discipline. Researchers can use whatever mechanisms they like, provided they are justified by appeals to "minimal design" and "legibility at the interfaces." While in practice, as noted, the Minimalist inventory of grammar mechanisms is not significantly larger than that of other frameworks, in principle there are no hard limits to the devices that researchers can dream up. Areas like the Movement Theory of Control, which disrupt standard assumptions about how the system should work, are worth watching precisely because they are the areas where researchers are most likely to go "off script," forcing the authors of a grammar toolkit to either reanalyze the proposals in terms amenable to the system, or to expand the system to accomodate.

THE MOVEMENT THEORY OF CONTROL

Broadly speaking, the Movement Theory of Control arises from two observations: that PRO is a suspect category under Minimalist principles - ripe for elimination - and that many types of control phenomena pattern with A-movement, suggesting a means to replace PRO.

Problems with PRO

The main problem with PRO is that it was postulated to meet an outdated requirement: "GF-θ," or the requirement that all θ positions be filled at D-Structure. If an item had a θ-role-bearing argument slot, then that slot must be filled *before the derivation begins*. This ruled out, as a consequence, both the idea that an item could bear more than one θ-role and the possibility of moving an item into a θ-position during the course of the derivation[49]. Consequently, if a θ-role appeared to be discharged in an otherwise grammatical sentence, *something* must be there to absorb it at the outset of the derivation, even if that thing were never expressly pronounced. This was PRO, and the fact that it seemed to participate in independently-attested grammatical functions (such as locally binding a reflexive in sentences such as *John ordered Bill*$_{\mathrm{i}}$ *PRO*$_{\mathrm{i}}$ *to shave himself*$_{\mathrm{i}}$) bolstered claims of its existence.

Unmistakable even then, however, was the suspicious fact that PRO has exactly the properties that the grammar needs it to have. It acts like an anaphor when it needs to and like a pronoun otherwise. It is in complementary distribution with full lexical items, only ever appearing as the subject of an infinitive (or a gerund) when one is independently needed

[49] At least, this is true on the assumptions that positions can't be doubly-filled and that movement leaves behind a *trace* that continues to hold a position that was lexically occupied at an earlier stage in the derivation.

but not pronounced, and only in a tightly restricted set of environments (the subject of gerunds and infinitives) - i.e. only in exactly those positions it was created to fill. In fact, this stipulation was quite explicit in the form of "NULL case" (Chomsky and Lasnik 1993), a type of case requirement that was only satisfiable by PRO and only "attested" on heads that took PRO as an argument. Although some conceptual motivation was given for this - PRO was a "pronominal anaphor" and thus could not be governed without violating one or the other of the otherwise mutually exclusive conditions A and B of the Binding Theory - the motivation itself rested on a theory-internal stipulation: that infinitival T obviated government.

Worse still, all of this was theoretically costly. PRO necessitated the addition of a module to the grammar - the Control Module, with a process of Construal - to explain its interpretation. This is because it was never quite correct to say that PRO was a "pronominal anaphor." Rather, as pointed out by (Landau 2004) and (Bouchard 1984), among others, PRO behaved as *either* an anaphor *or* a *pronoun* depending on the situation. It was never actually both at once - and so a separate process was needed to say which it was in any given context.

All of this was tolerable in the GB framework where the GF-θ requirement obtained. It was, one could say, a price duly paid for that assumption, and worth paying so long as that assumption seemed worthwhile. Moreover, in the GB framework, there was a plausible explanation for how it arose. Since lexical insertion happened only after a complete structure had been generated, PRO could be conceptualized as the consequence of *failure* to insert a lexical item at a particular point in the structure. This accorded well with its "purely grammatical" nature: it was syntax without independent lexical content.

With the advent of Minimalism, however, things changed radically, and D-Structure was one of the first constructions eliminated, taking the GF-θ requirement with it. The θ-Criterion

remained important as a way of preventing overgeneration[50], but it had lost its basis in D-Structure and had to be restated as a requirement that θ-roles cannot be discharged by internal merge (i.e. movement) (Chomsky 1993 ; Chomsky 2001). Additionally, the new framework came with a commitment to lexicalism. There should be no purely grammatical formatives; structures should be formed exclusively through the **Merge** of lexical material with other lexical material, or the output of previous such operations. This left no room for positions in the structure that were formed but not filled. With the level of representation that had justified its theoretical cost removed and the avenue for its generation closed off, PRO had to be recast as a peculiar kind of lexical item, and it was natural to begin to question whether it were worth the price. The argument for its existence became entirely empirical, and the stage was set: to the extent phenomena whose explanation rests on PRO can be reanalyzed in PRO-free terms, PRO should be eliminated. This is what the Movement Theory of Control aims to do.

What to do with PRO

As its name implies, the MTC has an idea where to start: with displacement, specifically with A-movement. It has been noticed that in certain types of control (obligatory control), PRO patterns with A-movement, behaving as though it were the trace of a moved item. Some characteristics the two have in common include[51]:

PRO is unpronounced. No explanation for this is needed if PRO is an A-trace, as A-traces are in the general case not pronounced (Hornstein 1999). It is a strange property for a lexical item to have, however (Landau 2015).

[50] $*John_i$ *loved* t_i cannot mean "John loved himself" - and the requirement that each θ-position have a distinct argument prevents the derivation that would allow it.

[51] All examples in this section are from (Boeckx, Hornstein, and Nunes 2010).

Obligatory Control (OC) PRO requires a c-commanding antecedent

1. He$_i$ hoped PRO$_i$ to shave himself$_i$
2. *It$_i$ hoped PRO$_i$ to shave himself$_i$
3. *John$_i$'s campaign hoped PRO$_i$ to shave himself$_i$
4. *John$_i$ thinks that it was hoped PRO$_i$ to shave himself$_i$

Only the first sentence is grammatical. In the second, the intended antecedent doesn't match in ϕ-features, in the third it does not c-command, and in the fourth it crosses over a clause boundary. PRO, it seems, must be locally bound.

OC PRO has an obligatory *de se* reading

1. [The unfortunate]$_i$ expects PRO$_i$ to get a medal

This can only be interpreted to mean that *the unfortunate* expects *of himself* that he will get a medal by contrast with *The unfortunate*$_i$ *expects that he*$_{i(j)}$ *will get a medal*, where *he* can be *the unfortunate* or, optionally, someone *the unfortunate* mistakes for someone else but is actually he himself. The details of whence *de se* readings arise are complex and somewhat unclear, especially as regards why they arise from movement. The point for the present discussion is simply that this is a property that OC shares with A-movement, supporting the intuition that the same process underlies both.

OC PRO has a sloppy reading under ellipsis

1. John$_i$ wants PRO$_i$ to win and Bill$_j$ does too.

does here can only be "wants $\text{PRO}_{\text{j=Bill}}$ to win"; it does not get a reading where Bill wants John to win. If PRO is construed independently by some other module, we'd have to say how that restriction comes to be. If it's simply a(n ellided) trace of the movement of *Bill*, however, no independent explanation is necessary.

(Boeckx, Hornstein, and Nunes 2010) - and others working in the MTC paradigm - are careful not to claim that all empirical obstacles to reanalyzing (OC) PRO as A-movement thereby collapse. Rather, the claim is just that these examples demonstrate sufficient similarity with A-movement to make analyzing PRO in that way the null hypothesis. Since independent stipulations in the form of bans on movement into θ-position are needed to rule it out, the burden of proof is squarely on its opponents.

EMPIRICAL PROBLEMS WITH THE MOVEMENT THEORY OF CONTROL

Arguments against the Movement Theory of Control are indeed primarily empirical. Brief explanations of the most common follow.

Control is not Raising under Passivization

The most prominent argument in (Landau 2003) involves passivization. If the salient difference between raising and control is only the additional θ-role, as would be consistent with the MTC, then processes which consume the outer θ-role, such as passivization, should render control identical to raising. However, this does not seem to be the case:

1. John_{i} tried $\cancel{\text{John}_{\text{i}}}$ to kiss Mary

2. *John$_i$ was hoped ~~John$_i$~~ to kiss Mary
3. John$_i$ was persuaded ~~John$_i$~~ [$_{TP}$ ~~John$_i$~~ to kiss Mary] (cf *Bill persuaded John to kiss Mary.*)

The second sentence is ungrammatical[52], contrary to what the MTC would predict. This has an explanation if PRO is a distinct entity with properties that separate it from lexical DPs - something about those properties (case, probably) can be marshalled to prohibit PRO from raising to a lexical subject position. It's difficult to see how the MTC would block it, though, since "PRO" is simply *John*, a lexical DP of the type that we independently know can be the subject of a passive.

Defective Intervention Should Block Subject Control, Contrary to Fact

Another prominent argument in (Landau 2003) is that the citation of the Minimal Distance Principle as evidence in favor of the Movement Theory of Control is overstated.

1. John$_i$ wanted PRO$_i$ to leave
2. John$_i$ persuaded Mary$_j$ PRO$_j$ to leave
3. John$_i$ promised Mary$_j$ PRO$_i$ to leave

If control is A-movement, and we know from A-movement that there are intervention effects, then the general preference for object control in cases where the matrix verb has two arguments follows naturally. The trouble with that reasoning is that subject control should in principle be impossible, as the object always intervenes, at least on the standard assumpton that "distance"

[52] At least, this is the claim in (Landau 2003). Some speakers do not immediately share this intuition.

is mediated through c-command. The point here is not that there are no workarounds - there are, and proponents of the Movement Theory of Control have proposed in particular to reanalyze the object of *promise* as a PP headed by a null P. The point is that whatever workaround is invoked amounts to recreating the Control Module by proxy, as the semantics of the matrix verb are the appropriate locus of differentiation.

The Embedded Clause is an Independent Case Assignment Domain under Control

The most prominent argument in (Bobaljik and Landau 2009), but well known from earlier sources (e.g. (Sigurdsson 2008; Thrainsson 1986)) is that the Movement Theory of Control fails to account for the fact that the embedded clause appears to be a separate domain for case assignment under control but not for raising. This shows up prominently in quirky case languages, where case assignment is dependent on the verb, with some verbs following normal patterns and some imposing deviant/idiosyncratic ("quirky") case marking. The following pair of Icelandic examples is from (Sigurdsson 2008):

(13) a. Mennirnir/*Mönnunum vonast til [að PRO verða baðum hjalpað]
men.the.N/*D hope for to PRO be both.D helped.DFT

"The men hope to both be helped"

b. Mönnunum/*Mennirnir virðist baðum [t hafa verið hjalpað]
men.the.D/*N seem both.D t have been helped.DFT

"The men seemed to have both been helped"

hjalpað is a quirky case verb and assigns Dative. In the control example (1c), dative marking does appear on the agreeing participle and on *baðum* in the emebedded clause, but in the matrix clause the subject surfaces nominative and indeed cannot be dative. There is case

mismatch between the matrix controller and the controlled PRO. No such mismatch occurs in the raising example, however. Case concord would appear to be clause-bound. These facts are consistent with an implementational split between raising and control in which control involves two chains in separate clauses and raising only one.

Partial Control

Probably the biggest implementation challenge for the Movement Theory of Control comes from partial control. As detailed in (Landau 2000), partial control is a phenomenon where the reference of the controlled PRO is accounted for but not exhasted by the reference of the controller. Examples include:

1. The chair$_{\mathrm{i}}$ preferred PRO$_{\mathrm{i+}}$ to gather at 6.
2. Bill$_{\mathrm{i}}$ regretted PRO$_{\mathrm{i+}}$ meeting without a concrete agenda.
3. Mary$_{\mathrm{i}}$ wondered whether PRO$_{\mathrm{i+}}$ to apply together for the grant.

The challenge for the MTC is obvious: there is no known mechanism which would allow a pronounced copy of an item to refer to itself but its lower copies to additionally refer to outside entities. Possible workarounds would need to either posit additional, unpronounced material to account for the additional reference, or find a way to form an A-chain in which the members were distinct in spite of everything. The former solution is conceptually stipulative, undercutting the main theoretical attraction of the MTC, and the latter recreates the control module by other means.

THE AGREE THEORY OF CONTROL

Whatever its empirical challenges, the Movement Theory of Control is inarguably well grounded in Minimalist principles. In scrutinizing the justification for PRO, it asks minimalist questions, and in suggesting its replacement through independently-attested mechanisms of grammar, it proposes minimalist answers. But it would be too simplistic - and perhaps even a bit romantic - to cast the debate as a classic tumble about where to draw the line between theory and practice[53]. For one thing, MTC proponents argue that they have risen to the challenge of addressing many proposed empirical problems, and that those that remain will be more illuminating for the study of human grammar if investigated with a mind to resolving the issue with displacement. For another, opponents are not without theoretical grounding of their own.

In particular, alternatives to the Movement Theory of Control rely heavily on **Agree** operations to mediate the relevant dependencies. Since in virtually all modern theories displacement only ever occurs as a side-effect of **Agree**, and as instances of **Agree** without accompanying displacement become ever more common in analyses, opponents of the Movement Theory of Control can plausibly claim that it is premature. While it's certainly true that Minimalism guides researchers to eschew stipulative grammar formatives where possible, the argument goes, an equally valid minimalist principle would encourage expression of grammatical relations in terms of mechanisms known to be operative in *all* long-distance dependencies before there is sufficient evidence to assume the more specialized treatment. Control relations will have to be explained in terms of **Agree** no matter what; why add displacement to the mix before such a move is forced by the data?

[53] Or, in the jargon of the field, between explanatory and descriptive adequacy.

Because of the heavy reliance on **Agree**, it is common to refer to alternatives to the Movement Theory of Control as "Agree Theor[ies] of Control." Most prominently, the "Calculus of Control" approach of (Landau 2004; Landau 2006; Landau 2008) is so called. A brief sketch of its operation is given below.

Like movement-based approaches, the Calculus of Control starts with the understanding that PRO is a problematic creature, a byproduct of mostly theory-internal concerns whose distribution and behavior is in need of independent explanation. If a derivation of PRO from independent princples remains out of reach for now, then the next best thing is to account for its distribution by appeal to something less stipulative - or at least stipulated on the basis of better[54] emprirical observations - than "null case." This is done through a "calculus" that involves the interaction of Semantic Tense and Abstract Agreement - features of C and I/T - that conspire, or don't, to create an environemnt where PRO can be merged. Here "Semantic Tense" is meant to be independent of morphological representation. This feature encodes the degree to which meaningful tense is interpreted by a contribution from the bearing head itself ([+T]), or is entirely inherited from a selecting head ([-T]).

Arguing primarily from data on Balkan subjunctives, Landau shows that obligatory control only ever happens in environments where the tense of the embedded clause is in some sense linked to, or dependent on, the tense of the main clause. This could be because the tense of the embedded clause is either non-existent or completely parasitic on the tense of the main clause - called *anaphoric* tense - or because it is determined by it, though represented separately - called *dependent* tense. Where tense is completely independent, there is no control at all, and any null subject is *pro* and has completely independent reference from any aspiring

[54] "Better" in the sense that they go somewhat further beyond simply describing the *immediate* environment in which PRO "appears" by tying control to the degree of tense dependency of the lower clause on the matrix.

matrix controller.

Tense for the purposes of embedded clause selection arises - or doesn't - from a conspiracy of features on embedded C and I/T. It is always interpretable on T and uninterpretable on C. By standard assumption, if the Tense feature on T and C match, the uninterpretable member of the pair is deleted, and the interpretable member lingers. *Anaphoric* tense is represented with a [-T] feature on C, and dependent with [+T]. C need not, however, bear a Tense feature. When it does not, the Tense of the lower clause is free - as is the case in most languages when the lower clause is indicative. Put differently, an abstract Tense feature on C provides a medium through which a matrix clause can *locally* select an appropriately-tensed clausal complement. The checking relation between C and T ensures that C manifests the clause's Tense, and the selecting matrix head can combine, or not, appropriately.

Agreement forms the second component of the Calculus. Landau assumes that the Agree feature is parasitic on Tense, appearing only then when Tense is present and specified for dependent tense. In cases where Tense on C is either anaphoric or not specific, there is no Agree feature.

This breaks down in the following way:

NO CONTROL

Indicative - C: NULL, T:[+T,+Agr]

Balkan Finite Subjunctive - C: [+T,+Agr], T:[+T,+Agr]

OBLIGATORY CONTROL

Exhaustive Control Infinitive - C: [-T], T:[-T,-Agr]

Balkan Control Subjunctive - C: [-T], T:[-T,+Agr]

Partial Control Infinitive - C: [+T, (+Agr)], T: [+T, -Agr]

(Additional specifications for Hebrew finite control clauses are skipped here.)

The main generalization is that control is an "elsewhere" case: it is blocked wherever I/T is [+T,+Agr]. Obligatory control environments are not a natural class - rather, they arise whenever there is some featural deficiency on the T head.

The Control Module itself arises from an interpretable feature [-R] on PRO. An uninterpretable counterpart of this feature arises on T or C whenever (a) *both* [T] and [Agr] are present and (b) one or the other of them is [-]. If both are present and [+], the [R] feature is also present and [+] - i.e. is [+R] and obviates control. R is lacking altogether when either [T] or [Agr] is not present. The right way to think about [R] then is as an "independent reference feature." T (or C) with [+R] selects an expression with independent reference - i.e a lexical DP or a pronoun (or, by implication, *pro*). [-R] matches with an item that needs to get its reference from elsewhere - i.e. selects PRO.

In theoretical terms this is, of course, more or less a recreation of the GB and "null case" systems. We still have a feature that specifically selects PRO and only PRO and happens to only arise in precisely those environments where PRO can appear. Likewise, PRO appears in this environment only because nothing else can. Just like in GB, where PRO was a position without content, where lexical insertion had failed, here PRO represents a kind of environmental deficiency as well. The innovation comes in making this environment parasitic

on independent factors - semantic tense and morphological agreement - that are known to determine control environments by virtue of their absence.

The control module itself works under two assumptions: (1) that features, even checked features, are not "used up" until the end of the phase, so that uninterpretable features can probe more than once before phase end, and (2) that *Agree* establishes relationships between heads, and so is capable of transmitting information - the antecedent's referential index in this case - across long-distance dependencies, a standard assumption.

Exhaustive control (EC) works as follows:

1. T has an uninterpretable [-R] feature by virtue of (a) having specifications for both [T] and [Agr], (b) at least one of which (both in this case) is [-].
2. T cannot check its uninterpretable [-R] feature on C because C, being specified [-T] only, fails to meet the "both" requirement (requirement (a) in the previous point) for having an [R] feature at all - therefore it lacks any such feature.
3. T must check its [-R] feature on PRO, then. So, PRO is present.
4. The [-R] feature on PRO means PRO has to enter into a second relationship to get its reference. It can do this with an item that is [+R] - i.e. with whatever functional head selected the controller. This head (T in the case of subject control) will necessarily bear an uninterpretable [-R] feature, the feature that selects the [+R] referential DP in its specifier. Under the assumption that *probe*s are active until the end of the phase, it will *probe* a second time into its complement. Since C bears no intervening [R] feature of its own (lacking [Agr]), there is nothing to block the *probe* finding the [-R] feature on PRO. Having inherited a [+R] value from the DP in its specifier (that is, having inherited a *reference value* from it), PRO will be able to check its [-R] feature and get a reference. This is how it comes to be coindexed with the controller.

Partial control works similarly, with one critical difference:

1. T has an uninterpretable [-R] feature by virtue of (a) having specifications for both [T] and [Agr], (b) at least one of which (T in this case) is [-].
2. Since T is [-R], as before, it will check PRO.
3. Because C is specified for [+T,+Agr], it also has [+R], so T can enter into a relationship with it. Since it is [+R], PRO will get reference from it, since PRO is already in a relationship with T.
4. As with exhaustive control, the functional head that selects the controller will continue to probe during its phase even though it has already satisfied its [+R] requirement by means of having selected the (ultimate) controller.
5. This time, it will stop at C, which has an intervening feature [+R] of its own, not finding PRO. However, because the (ultimate) controller is in a checking relationship with the probing matrix head, and the probing matrix head is in a checking relationship with C, and C is in a checking relationship with T, and T is in a checking relationship with PRO, the controller comes to control PRO *mediated through C*. It is this mediation that gives rise to partial control effects - because PRO gets its reference from two places, one concrete (the lexical subject controller in the matrix) and one abstract (C in the embedded clause).

IMPLEMENTATION

The purpose of this project is to lay the groundwork - both theoretical and practical - for a grammar construction toolkit for the Minimalist Program. On the theoretical side, it identifies a minimal set of devices capable of covering the most widely-cited analyses, and on the practical side it makes these available to researchers in the form of an implemented and

useable toolkit. The proof, of course, is in the pudding, and the following sections outline how a researcher might go about implementing tricky analyses in the competing systems of the Movement and Agree Theories of Control with the tools available in the system. In each case, the actual implementation reveals hidden assumptions (albeit minor in the case of the Agree Theory) not mentioned in the original reports that illuminate details of the theory.

Movement Theory of Control - Visser's Generalization

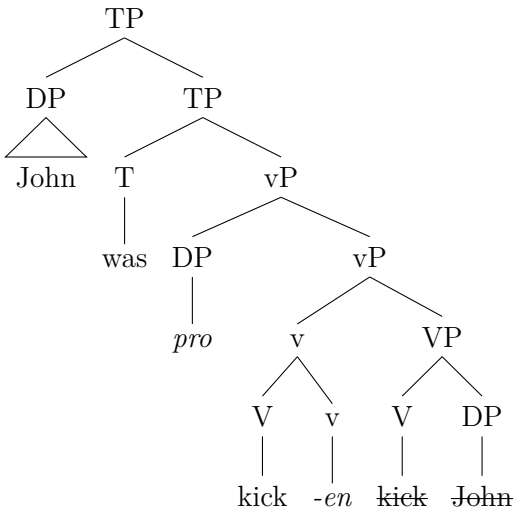
Subject control structures which involve a matrix object are problematic for movement-based theories of control as the theory must account for why the object doesn't intervene. Passivization is a problem for *any* derivational theory for similar reasons - at least to the extent the analysis involves the use of a null subject, as has been standard since shortly after (Baker, Johnson, and Roberts 1989)[55]. Thus, Visser's Generalization (Bresnan 1982), according to which subject control verbs cannot passivize - put differently, control by an implicit subject in passives is impossible - poses a particular difficulty.

The problem is roughly this.

John was kicked is clearly grammatical. On the standard analysis, the canonical object *John* raises to subject position to get case because: (1) the external θ-role of the verb has been absorbed by the passive morpheme which (2) by *Burzio's Generalization* renders it incapable of assigning **ACC** case so (3) to be case-licensed, *John* must move to the next plausible location for case assignment, which is **Spec**-T.

[55] In the original, the passive morpheme itself was taken to be such an argument. Such an analysis would plausibly not block raising to subject (by the object) as the *-en* morpheme is presumably not a D category and is a head in any case. This solution isn't really available to a minimalist researcher in good faith, however, since it would require explanation for why everwhere else in the grammar reference is a property of D and only in this one place of a verbal morpheme.

The structure is meant to look something like this:



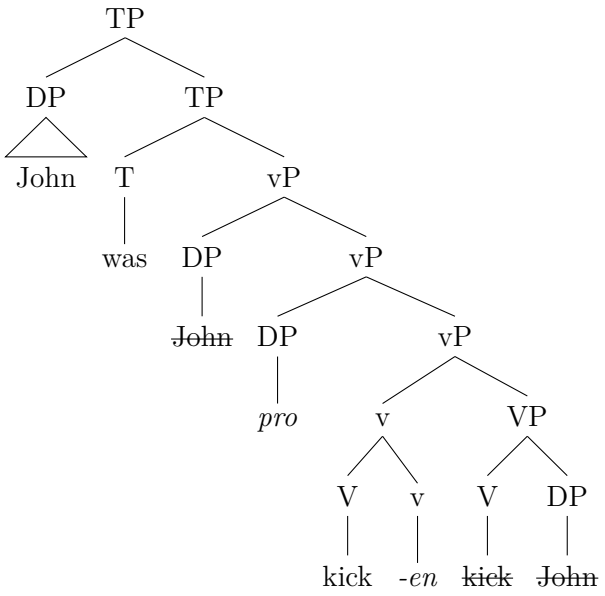
The problematic bit is the *pro* in **Spec**-vP. It is motivated by evidence that the implicit argument in a passive construction can control:

The ship was pro$_{\mathrm{i}}$ *sunk PRO*$_{\mathrm{i}}$ *to collect the insurance* (Bhatt and Pancheva 2006)

Whoever sunk the ship also intended to collect the insurance; this fact is presumably accounted for in the usual way - binding of a lower PRO by a higher argument, leading to coreference.

The trouble is that the *pro*, as a D category, should intervene in any attempts to raise the object to subject position.

The account that (Hornstein and Polinsky 2010) give is not too controversial: in addition to absorbing the external θ-role (and subsequently also **ACC**), passive-v also admits of an additional **Spec**, and *John* makes its way to this position on its way to T. When T is **Merge**d, it is at least equidistant with *pro* and hence visible to the **Probe**.



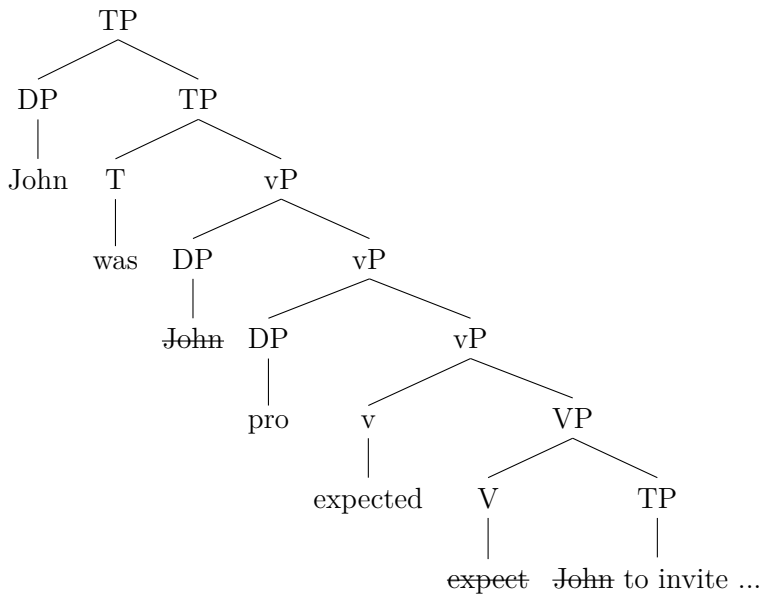
(Hornstein and Polinsky 2010) don't commit themselves to any particular motivation for this movement, but they hint that it might be to value ϕ-features on the passive participle (since this sort of agreement is visible in a number of languages). Whatever the motivation, it solves the problem.

However, it raises another one: if the additional **Spec** is available for passives, how do we prevent the following:

1. *John$_i$ was hoped PRO$_i$ to win.
2. *Mary$_i$ was promised (by John$_j$) PRO$_j$ to win.

The puzzle of the *promise* example requires recognition of the fact that ECM verbs allow passivization under control:

1. John was expected to invite Mary to the prom



(Hornstein and Polinsky analyze ~~*John*~~ *to invite Mary to the prom* as a TP, but nothing hinges on that here.)

Meaning - the additional/outer **Spec**-vP that provided an escape hatch is also available in ECM constructions. It can't, however, be available in object-less subject control:

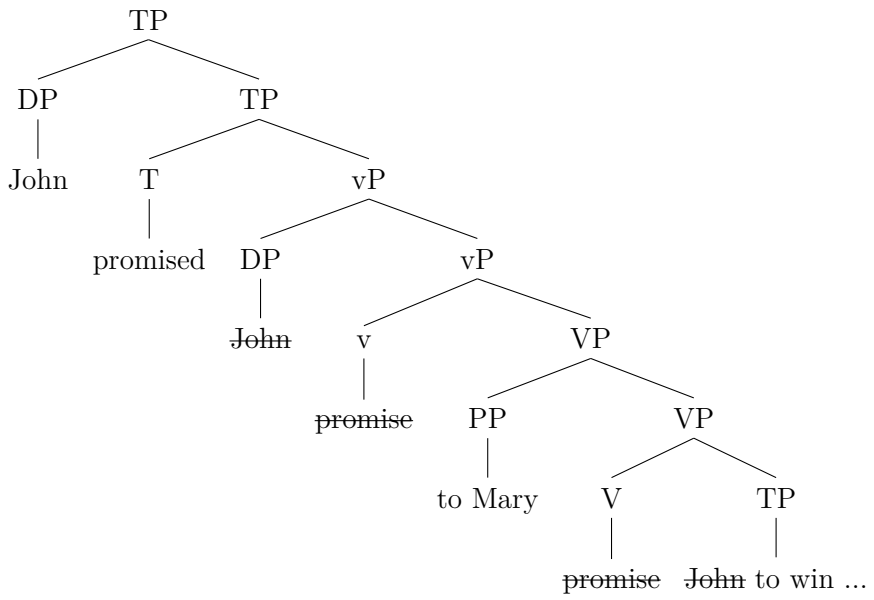
*John$_i$ was pro$_j$ hoped PRO$_i$ to win.

Presumably ECM verbs and ditransitive subject control verbs can be made to form a (natural) class, and the above example ruled out by the fact that *hope* isn't a member. Hornstein and Polinsky never say what generalization accounts for this, but it seems natural to assume that they have in mind that the extra **Spec** is there as a side-effect of the absorption of **ACC** case. In a purely lexicalist system, then, passive-v comes in two forms, one that selects transitive V and one intransitive, and the one that selects transitive happens to have an outer **Spec**.

If that is the generalization, though, lack of an extra **Spec**-vP (obviating intervention by the null subject) cannot be the explanation for the ungrammaticality of **Mary$_i$ was promised (by*

John$_j$*) PRO*$_j$ *to win.*

The solution proposed has a classic "rule-ordering" feel to it. Basically, the idea is that in *promise*-type verbs, the internal argument is not **ACC** anyway, but rather some kind of dative experiencer and thus buried in a PP headed by a null P.



Passivization for such verbs is really some kind of pseudopassive: it's the object of the preposition that passivizes rather than the entire phrase.

Mary was promised a rose garden (by John)[56]

Hornstein and Polinsky take this to indicate that some kind of incorporation has taken place. Presumably once V adjoins to v, v-V P is reanalyzed as v-V-P, transforming the external argument of the V into a regular DP. This is arguably necessary to allow *Mary* any sort of mobility at all - cf:

A rose garden was promised to Mary

[56] *To Mary was promised a rose garden* is not strictly ungrammatical, of course - perhaps a bit stilted - but it is focused in a way that the reference sentence is not.

??A rose garden was promised Mary

Hornstein and Polinsky are not clear on when the incorporation happens in the reference example, but they are certain it must happen before any attempt to raise any embedded *pro* to **Spec**-v, as this is how they propose to rule out

*Mary$_j$ was pro$_i$ promised- ~~to Mary$_j$ promised pro$_i$~~ to win (by John$_i$).

From repeated references to concerns of "violation of cyclicity/respect for the Extension Condition," one can assume they have in mind that incorporation happens *before* **Merge** of T; anything later than that would be altering already-determined truths about the structure "out of turn." While this suffices for the example at hand, one would really like more detail here, in light of the fact that incorporation seems to be optional - cf *A rose garden was promised to Mary* above.[57] For the purposes of the implementation below, it will be taken to happen as a side-effect of head movement of V to *v* - specifically, in the *compact* cycle mentioned in the previous chapter.

With all aspects of the analysis assembled, it's possible to provide an implementation guide in the system.

The lexical array is:

John, promise, to-P, Mary, to$_{inf}$, win, v$_{trans}$, v$_{ditrans}$, T, C

grouped into subarrays as follows:

{win, John, v$_{intrans}$}, {to$_{inf}$}, {to-P, Mary}, {promise, v$_{ditrans}$}, {T-past, C}

[57] Of course, one can always avail oneself of PF interface solutions to these problems. It could be argued that *to* resurfaces in pronunciation, desipte having been syntactically incorporated, only in those cases where its object remains *in situ*.

In "generation mode" all of this could be randomized, but for the purposes of explication the arrangement that works is chosen.

A `stage` is seeded with an empty `workspace` and the first of the `lexical subarrays`: {win, John, $v_{intrans}$}

Activate chooses $v_{intrans}$

$v_{intrans}$ **Select**s *win* - which is to say, it has a **selector** feature that will match a **selectee** feature on *win*. *Win* is moved out of the `lexical array` into the `workspace`, specifically it is unshifted onto *v*'s *constituents stack* - (since **merge** is a side-effect of **select**, and **select** happens whenever selectional features match and value).

compact happens as a side-effect of this **merge**, the uninterpretable **selector** features are eliminated from *v*, and **transfer** occurs as a result - specifically of the **donate** variety: all V's features unshift onto the front of *v*'s feature stack, effectively implementing head adjunction to *v*.

v is still active because it still has a **probe** - a **selector** feature looking for a D-category **selectee**. **merge-over-move** requires it to first look in the `lexical array`, which it does and finds *John*.

Again, **merge** is a side-effect of **select**, *John* is moved to the end of *v*'s "constituents" array, making if effectively a **Spec**.

v's **probes** are not exhausted - there is still a *theta* probe, which is satisfied on *John*. Having exhausted its **probes**, *v*'s **selectee** becomes active.

Now that *John* is active, it can try its *case* **probe**, first on the head T and later, if not successful, on the items in the head's *constituents* array. It finds nothing.

There are no lexical items in the `lexical array`, so the `stage` halts in a *divergent* state.

A new stage is created to deal with the next `lexical array`, which is simply to_{inf}.

The output of the previous `stage` is placed in the `workspace` of the current one. to_{inf} is **activated**, which means **selected** from the array. It has a **selector Probe** which, by **merge-over-move**, must first inspect the `lexical array`. This is empty, so it looks into the stage's items. There is one - the output of the previous stage, so it **probes** into this and finds a match on *v*.

select has **merge** as a side-effect, so the output of the previous stage is moved to *to*'s *constituents* array. **value** and **compact** take place, removing the **selector** feature on *to*.

to is still active, as it has an EPP **probe**. EPP is an aberrent **probe** that **selects** without **valuing**. It finds a match in *John*, which is copied from *v*'s constituent list and added to *to*'s as a **Spec** (i.e. is placed on the end).[58]

Now that *John* is active, it can try its *case* **probe**, first on the head *v* and later, if not successful, on the items in the head's *constituents* array. It finds nothing.

The `stage` remains *divergent*, because although there is only one object in the `workspace`, there are unvalued features. Because the `lexical array` is exhausted, however, the `stage` cannot continue, so a new `stage` is created for the next `lexical subarray`.

[58] In actual implementation, merely the *pointer* to this object is copied.

This is {to-P, Mary}, one of the two crucial points.

As before, previous output is placed into the currently-active `workspace`.

Next, the `stage` **activate**s *to*, which involves **selecting** it from the `lexical array`. *to* then **probes**, which by **merge-over-move** must begin in the `lexical array`. It finds a match for its **selector** feature in `Mary`. As usual, **match** is followed by **value** and **compact**. `Mary` is now active but has only a **case** probe. Depending on individual implementation, this may find a match on P, or it may wait until *to Mary* is (eventually) **merge**d with the rest of the tree. Either way, there's nothing more that happens at this `stage`, so let's assume *Mary* **probe**s *to* and gets its **case** feature valued, just to get it out of the way.

This `stage` is still *divergent* because of (i) unvalued **selectee** features on P and (ii) unvalued **case** features on *John* (which is in the output of the previously-divergent **stage** and now sitting in the **active** array of the current `stage`.

Next up is {promise, $v_{ditrans}$}. This stage illustrates an interesting but ultimately inconsequential quirk of the system in that the argument to the phase head must itself take arguments, and yet the algorithm following up to this point preferentially selects the head with the most active non-selector **probes** - i.e. the phase head (if it exists). This will give the appearance of countercyclic **merge**. However, nothing really hangs on this. (Chomsky 2008) has already suggested that everything at the phase level happens simultaneously; this system takes that to the bank.

v is **activated** and must find a **goal** for its **selector** probe. **merge-over-move** requires that it look for this first in the `lexical array`, and so it finds *promise*. **select** happens, followed by **merge** and **value** and **compact**. Part of **compact** in this case is **head movement**: V

incorporates into *v*.

Recall that this involves transfer of *all* of V's remaining feature array to *v*; in effect, they become the same **head**. Thus, what might have alarmed the reader about the ordering here need not: arguments to V are still selected in the proper order - the only difference from standard accounts in the literature is that V does not have to "cross over" anything in its **Spec** to get to *v* since **head movement** happens first.

V having *unshift*ed its features onto *v*, its **probes** are now active - in "reverse" order, in accordance with the Mirror Principle, actually (Baker 1988).

It will find *John to win* and *to Mary* in the proper order and add them to the constituents array of V-v after **select**, **value** and **compact** on each.

Now, at this stage, **head movement** of P to V-v should really happen, since this would be the final operation that involves only the V and v domains. Since P has no remaining features, this is entirely vacuous. But notice that this, in effect, means that incorporation happens *before John* can raise to subject. *Mary* should therefore block. In system terms, this is so because the next **probe** on V-v will be *v*'s subject probe. Since *John to win* and *to Mary* were **merge**d in that order, *John to win* is in **Comp** and *to Mary* in **Spec**. *to Mary* is in principle higher than *John to win*.

However, remember at what level this is occuring. V-v incorporation has already happened; these are already the same phrase. So, when *v*'s subject **probe** looks in its *constituents* array, it will actually look at *John to win* first. *John* can raise to subject just fine. Thus, this **select**s and then (re)-**merges** followed by **value** and **compact** (of *v*'s **selector** and *John*'s **case** features).

The rest of the derivation will proceed in the obvious way, with T **selecting** v and **probing** to find *John*, which is now the highest element in the P-V-v complex, and then this in turn being selected by C to close off the derivation (or, if prefered, T can **inherit** all relevant features only *after* **merge** with C in a (Chomsky 2008) style system, with *John* "countercyclically" raising to **Spec**-T only after this occurs.

What has been sketched here is the most natural derivation in the present system - and, arguably, the most natural derivation of this sentence. An astute reader will have noticed that it poses some interesting difficulties for the (Hornstein and Polinsky 2010) account of Visser's Generalization, however. In particular, there are two.

First, for all the talk of the importance of having P-incorporation happen "cyclically", it's difficult to defend the idea that it should happen any later than as an immediate consequence of **merge** with v. Even if there are nits to pick with the ordering of operations sketched here, such as insisting that *promise* first **merge**s with *John to win* and then with *to Mary* and only *then* allowing the whole VP complex to merge with v, as is the more standard approach[59], they will not really make it more "natural" for incorporation to have to wait until v has had a chance to raise *John* to subject out of the embedded clause just because that's the outcome the authors want. Delaying incorporation until after all argument positions have been saturated is possible in this system (one need only make incorporation parasitic on a feature specialized for the purpose that comes later in the list of **probes** on v), but it is stipulative.

Of course, the intended analysis in (Hornstein and Polinsky 2010) is that incorporation doesn't happen in the active version at all - it is only forced by the passive, presumably as part of the process that absorbs the internal θ-role of *promise*. That is a reasonably standard

[59]Though it's a bit difficult to defend when one stops to *really* think through head movement.

assumption, and it accords well with acquisition facts (also cited in the chapter) that show that *promise*-type subject control constructions are acquired late, if at all. The insight seems to be that if the system is going to deviate from expected behavior[60] to engineer a **Spec** for T, it can only do so locally, and it is precisely because the attempt is so local that it fails[61]. But given the ordering of operations, it's not really clear that the derivation fails for the reasons the authors assert. Or rather, if it does, then because there is a hidden θ-criterion-like assumption in their analysis which requires that an item not fill more than one argument role for the same head - precisely the sort of thing they are hoping to avoid.

Second, the whole idea of saturating arguments before incorporation begs another question, which is why the **probe** that finds a subject in the *John* buried in the embedded clause that is the Object of *promise* couldn't find one that simply *was* its complement - in, e.g., *John was kicked* from before. Recall that passive morphology adds a **Spec** to allow *John* to be higher than a putative null subject *pro* that has previously been **merged** in. Further recall that the motivation for movement to this **Spec** was never expressly given, but hypothesized to be to check ϕ features on V-v. If this is all true, what reasons would there be that would prevent v, in the event of the hypothetical absence of a *pro* in the `lexical array`, from **probe**ing its **Comp**, finding *John*[62], (re)-**merge**ing *John* in its **Spec**, and then checking the required ϕ-features at the same time?. There do not seem to be any without additional machinery.

Of course, there are good reasons to ban **Comp**-to-**Spec** movement. The point is that Hornstein and Polinsky will need to enforce such a filter. *Something* has to enforce the presence of *pro* on this analysis, and whatever that may be, it's suspiciously close to recreating

[60] By trying to treat the indirect object as the canonical object in passivization, in this case - the canonical object being what is promised, not the person it's promised to.

[61] Put differently, it is when effects extend beyond the individual phrase - in attempting to raise *pro* to subject - that things go awry.

[62] which is not "frozen" in the standard sense as it is not **case**-valued

the θ-criterion by other means.[63]

The upshot is that the explanation for the ungrammaticality of **Mary was promised to win* goes through - incorporation does indeed, in the most natural setup, precede any attempt to raise *pro* to subject, blocking the attempt.

Hidden Assumptions

The careful reader will have noticed that Hornstein and Polinsky got off on a technicality here. The two derivations actually work in the way they described for very different reasons. In the first case, it works only because the default behavior of this system incorporates V *before* merging any arguments. If V were forced to accept its arguments *before* `merge`ing with `v`, as is the standard assumption, then incorporation of P would block raising of *John* to subject just as easily as it blocks *pro*. Everything, therefore, inges on implementation details of a system that doesn't work in the way that the authors probably assume.

More to the point, there is nothing stopping the system from functioning in the way that is generally assumed. More detailed discussion of the mechanics of just such a derivation is given in Appendix A, but the broad point is that if the system is implemented in such a way that a `probe` runs in *reverse* order over the `constituents` array, enforcing hierarchy, the blocking effects return, as the system will find *Mary* - since it is a DP in good standing after P incorporation - before finding *John*. This would require a bit of reinterpretation of system functionality, but it is minor: **head incorporation**, properly understood, is merely a way for a binary branching system to behave like a shallow-tree system (a system in which trees

[63] at least on the phrasal level, because it effectively bans the same item holding more than one θ-role

can branch as much as needed to include all phrasal arguments) without entirely giving up on the binary representation.

Hornstein and Polinsky's timing problem is now real: they need a way for P-incorporation to happen in the passive derivation *only* - or, if it happens in the active derivation, then after a delay. So one possible hidden assumption (assuming it is not simply an oversight) that is discovered by use of a system like this one is that incorporation of the kind described is specific to *passive* "promise." It is, of course, possible to build this assumption in to the system: one need only give passive `v` and active `v` different categories (as seems natural) and then write the `strategy` function (see Chapter 4) so that it forces incorporation only on **select** by the *passive* version. This works, but of course it is not linguistically satisfying; it has the character of a hack.

There are more serious hidden assumptions that are not so easily resolved, however. Foremost among these is the assumption that *pro* can appear freely either in SPEC-to or SPEC-v. The problem is easily revealed by inclusion of *pro* in the lexical array that starts the active derivation. There are reasonable ways to block *pro* from appearing as the external argument to v in the active derivation, of course, but if they are linguistically honest they tie themselves to the presence or absense of the external θ-role, which sounds suspiciously like a step down the road to reinvention of the traditional θ-criterion. But what can't really be done on the assumptions that Hornstein and Polinsky are making is to block *pro* from the SPEC-T position in the lower clause. The trouble is that in the active version, there is absolutely no problem with initial-`merge`ing *John* in SPEC-v. After all, this is the canonical position for its first merge. Here it gets a θ-role, and it is in a position to raise to get case (and become the grammatical subject) in SPEC-T. Without some way of preventing *pro* in the lower clause, *John promised Mary pro to win* can mean *John*$_{\mathrm{i}}$ *promised Mary*$_{\mathrm{j}}$ *pro*$_{\mathrm{k}}$ *to win* - to wit that John

promised Mary that *someone else entirely* would win. This reading is simply not available.

The Calculus of Control avoids these problems by means of its "honest stipulation" - Landau's term of art for his concession that his (former) system reinvents the PRO module by other means. Perhaps it is a stipulation, but by tightly regulating what can appear in PRO positions, the Calculus of Control system correctly gets a crash on any attempt to introduce a disallowed element there. Again, the reader is refered to Appendix A for details of the derivation.

Agree Theory of Control - Visser's Generalization

As seen in the previous section, lack of precision about when, exactly, the incorporation of the P that heads the external argument of *promise* into V-v occurs allows the Movement Theory of Control-based treatment of Visser's Generalization facts in (Hornstein and Polinsky 2010) to appear more principled than it is. On close examination, for the analysis to work, incorporation has to happen at exactly the moment necessary to either allow raising to subject (in *John promised Mary to win*) or to block it (in **Mary was promised to win*), and this seems less than insightful. Can an Agree theory do any better?

Landau's Calculus of Control accepts PRO as an "honest stipulation" (Landau 2004) in exchange for, he argues, a better fit with the empirical facts. More precisely, the "honest stipulation" is that control is an elsewhere case that obtains whenever an embedded T or C head is defective in one or both of **Abstract Tense** and **Agr**. In general, T heads should fully represent Tense ([+T]) and Agreement ([+Agr]). If they are dependent on another head (usually C) in either regard, or simply fail to manifest either (infinitive *to*), they are marked with a diacritic feature ([-R]) indicating that they can only attract an item that is similarly dependent on outside specification - in this case for its *reference*. This place is held by PRO -

admittedly a “pure grammatical formative” of dubious suitability to the Minimalist Program - until more light can be shed on the subject. Since PRO must get reference to be licensed, control happens as a byproduct.

In the case of Visser’s Generalization, as pointed out by (Urk 2013), this has the virtue of tying the phenomenon to **Agree**, which seems empirically correct. That study gives the following revised definition of the phenomenon:

> Obligatory control by an implicit subject is impossible if an overt DP agrees with T

The revision owes to the facts of impersonal passives in languages like German,

(14) *Der Lehrer$_i$ wurde gebeten ihn$_i$ zu kitzeln dürfen.
The teacher was begged him to tickle allow

"The teacher$_i$ was begged to be allowed to tickle him$_i$"

(15) Mir wurde versprochen, mir noch heute den Link für das Update zu schicken
Me.DAT was promised, me.DAT already today the link for the update to send

"I was promised to be sent the link to the update today."

The first sentence shows that the personal passive - where agreement obtains between *Der Lehrer* and T-*wurde* - is ruled out just an in English. But in the second sentence, which is an impersonal passive, a construct that English doesn’t really have in precisely this form, there is no similar problem. Visser’s Generalization effects are correlated with a *nominative* argument agreeing with T; otherwise, they don’t obtain.

This achieves the same result by slightly more intuitive means. Like in the MTC analysis, the core problem with sentences like **Mary was promised to win (by John)* is one of construal:

the *pro* that is the subject of *promise* can't establish a relationship with the PRO that is the subject of *to win*. Given MTC commitments, construal failures must be cast in movement terms, so that means the problem is that *pro* can't move from **Spec**-to to the external argument position of the matrix *v*. This gets the right result, but it doesn't really have any good explanation for why **Agree** between the matrix T would be implicated. Empirically, however, there is reason to believe that Visser's Generalization only applies to constructions in which matrix T has successfully **Agree**d with a nominal.

So, how would that be implemented in the present system?

To rule out **Mary was promised to win*, the derivation proceeds in much the same way as the MTC derivation. We start with the same set of primitives arranged in the same way, just with the addition of *PRO*:

> {win, PRO, $v_{intrans}$}, {to_{inf}}, {C}, {to-P, Mary}, {promise, v_{pass}}, {T-past, C}

The first part of the derivation proceeds as before - *v* **select**s *win* and then PRO. Then to_{inf} **select**s the `syntactic object` formed by this complex. Because to_{inf} is specified [+T, -Agr], it comes with an [-R] feature, which requires it to select only PRO.[64]. So, PRO is **probe**d and moved to **Spec**-*to* (in system terms, it is **select**ed and added to the end of the constituents list of to_{inf}; *PRO win* is in the first position of that list).

Since this is a recreation *in spirit* of the null case approach, this `stage` is *divergent* because PRO is missing features - specifically [R]. But PRO itelf does not come out of the lexicon deficient. Rather, it **inherit**s the deficiency from T - in system terms, T exercises the **share** option on feature valuation, and PRO becomes entangled with T's [-R] feature, guaranteeing

[64]In this sense, as Landau freely admits, the theory more or less recreates the null case approach to PRO.

that it will end up with whatever reference T receives.

From here, it is **merge**d with C, which, since this matrix allows partial control, is [+T, (+Agr)]. It therefore has a [+R] feature which **value**s, again through **share**, the one on *to* and, by implication, the one on PRO. However, at this stage in the derivation the value is abstract - merely a placeholder.

From there, the derivation proceeds as before. The details are slightly different, however, as this is a passive construction. So, *promise* **select**s *to Mary*, and the preposition incorporates, but v_{pass} does not value anything with case features. It also **select**s an external *pro*. This is easy to enforce because this system has no problem with the θ-Criterion, and there is a ban on double assignment of θ features.

Because *Mary* is available to **probe** by T, when that is **merge**d with the existing matrix v structure, *Mary* raises to **Spec**-T (again, in system terms it is pulled out of the first object in T's `constituent list` and put on the end of same). This means that T has **Agree**d with *Mary*. Since matrix T is [+T, +Agr], it is also [+R]. By assumption (in Landau's system), **probe** continues until it is exhausted, so after **probe**ing its **Spec**, the matrix T continues to **probe** into its **Comp**. It will eventually find C and **share** its reference, which has been valued by *Mary*. This fact prevents it from ever being in an reference relationship with *pro* (the implicit subject), and this, rather than outright defective intervention ungrammaticality, explains why subject control is impossible under passivization. Control is still allowed, and indeed, *Mary*$_{\text{i}}$ *was promised PRO*$_{\text{i}}$ *to win* works (at least for many speakers) under that interpretation.

The piece that is missing from Landau's system that close implementation in this system reveals is that there's nothing actually preventing the **probe** from continuing on after it's

entered into a relationship with (embedded) C. If **probe**s can continue exhaustively, why can't this one keep looking in C's **Comp** after having hit C itself?

(Landau 2004) needs the C intermediary to implement Partial Control. It's not entirely clear from the original paper whether PC isn't achieved through relationship with C and PRO simultaneously - with direct reference (to *Mary*) mediated through the direct relationship with the T complex and the implicit additional referents through C. That doesn't *seem* to be what Landau has in mind - reference seems to go exclusively through C as a transmission medium - but if the author has no objection, perhaps this system could actually be implemented that way.

However, it's more likely that what's intended is that C is a *phase* head, and **probe**s can't **probe** over *phase* boundaries. Therefore, C has the **barrier feature** from the previous chapter, and all **probe**s stop at the C head. The relationship is mediated through C because a more direct relationship is simply forbidden.

In light of that, however, it is worth noting that the analysis for Hebrew OC in the same work relied on an *exact* feature match between [+T, +Agr] on C and [+T, +Agr] on T eliminating both pairs, effectively rendering the *phase* transparent. In those cases, EC obtains, and the matrix functional head (T or *v* depending on the specific phenomenon represented) does indeed enter into a relationship with PRO directly. So, *phases* are sometimes transparent for Landau and sometimes not, and the mechanism by which this is enforced can't really be the one advertised (presence of a [+R] feature on C) because we know that **probe**s can go as far as they like in this system. For the purposes of faithful implementation of the system as Landau seems to have intended it, [+R] C heads bear the **barrier** feature in this system.

Hidden Assumptions

The assumptions on this theory are a bit more subtle. One - that the *probe* will continue past C, establishing more links in Partial Control cases than the theory advertises, is real but probably not consequential. Either Landau intended there to be features on the relevant C heads that block such probes, or there are two links - one direct and one mediated - between PRO and its binder rather than one. Two links probably isn't a problem, since the aim of the analysis is for the reference of PRO to expand beyond that inherinted from its antecedent anyway; it's the *presence of the mediated reference* that really matters.

However, the existence of multiple-probe itself is not so innocent. One then needs to be careful what probes are used for. For example, in the system implementation, it's assumed that **select** is a `probe`, meaning that a head - v, say - can keep `probe`ing even *after* it's found an external argument, possibly gathering several. Mechanisms have to be put in place to prevent this in situations where it's not wanted. The typical one would be "case freezing" - not allowing arguments to be considered as goals after their case features are valued. That's fine as far as it goes, but it runs into trouble on standard defective intervention analyses, where it is precisely the presence of an "inactive" (i.e. case-valued) element that prevents the probe from reaching its canonical goal.

Presumably none of this is a problem for Landau's system because it doesn't conceive of **select** in probe-goal terms. As "hidden" assumptions go, this admittedly isn't very revealing, but it is nevertheless the kind of detail that one doesn't notice before making an honest implementation attempt.

CONCLUSION

CONTRIBUTIONS

The primary contribution of this study has been the development of an LKB-style grammar development environment for researchers working in the Minimalist Program. To the best of the author's knowledge, it is the first such attempt within the GB/Minimalist tradition. This is somewhat surprising, given that the speculative nature of Minimalist research provides a number of opportunities for such a tool to be of assistance. In particular, because Minimalist research is directly concerned with the cognitive devices that are involved in sentence production - it goes beyond lexical specification of the component parts for a particular language, as is more the emphasis of competitor theories that tend to be either monostratal (HPSG, CCG, Dependency Grammar) or at least non-derivational (LFG) - the danger that new theories will break prior analyses in unexpected ways is particularly high in this tradition. A tool that can help either verify that a new analysis preserves the insights of prior analyses, or else point out where the points of conflict are, is potentially quite useful.

As it is not, however, the first attempt to make a grammar development environment for research purposes, nor is it the first attempt to capture minimalist insights in a software tool, it is worth taking some space to clarify the points of departure with similar efforts.

The LKB

The system that inspired this work, the *Linguistic Knowledge Builder (LKB)* of (Copestake 2002), is the most obviously similar in purpose. Like this system, the LKB aims to provide

researchers a tool that verifies speculative theoretical results primarily by (a) guaranteeing that results neither over- nor under-generate sentences relative to the researcher's claims and (b) pointing out any points of tension with earlier analyses - specifically areas where assumptions specific to the new analysis conflict with assumptions of previous ones. The only real difference is in the target: where the LKB implements unification grammars - preferably, though not exclusively, in the HPSG paradigm - this system is focused on the derivational approach of the Minimalist Program.

The difference is not trivial. While it is popular in formal circles to speculate that explicit transformation offers no generative power over and above what slash features[65] offer, the implementational details at least are quite different. Researchers used to thinking in transformational terms - indeed, who find transformations to be the correct abstraction for capturing human linguistic knowledge - find themselves in an alien environment working in the LKB. Since the transformational tradition remains prominent and productive, it is good to have a tool that speaks its language.

In addition to their significant implementational differences, Minimalism and Unification Grammars also differ sharply in focus. Unification Grammars are typically more practical-minded in purpose. They make weaker cognitive claims than Minimalism. The concern is less with getting the mechanisms of the mind exactly right than with modeling the output. For this reason, there is usually more concern with consistency and coverage in Unification Grammars. What percentage of observed sentences for a language[66] does a particular implementational lexicon generate? How does changing the feature specification of a particular lexical item's entry affect this coverage? These are the kinds of questions that feature prominently among

[65] A special kind of feature that encodes a long-distance dependency. See (Pollard and Sag 1994) for explication.

[66] Possibly as defined by a reference corpus.

researchers using the LKB. In the Minimalist tradition, by contrast, the focus is on the spadework of uncovering generative mechanisms. In a sense, Minimalist researchers are more focused on exotica, because these are the phenomena that are most likely to help define the boundaries of human linguistic competence. Concerns about coverage are typically secondary, and this ordering of priorities is reflected in the fact that there are, to the author's knowledge, no freely-available text corpora annotated for Minimalist research. One of the purposes of this project, indeed, is to bring coverage concerns to prominence within Minimalism. But even to the extent that it is successful, coverage will always take a back seat to spadework in Minimalism given the nature of the program. For this reason, there is in a real sense no possibility of an "LKB for Minimalism." This system serves a subtly different purpose for a subtly different research program.

Minimalist Grammars

The *Minimalist Grammars* of (Stabler 1997) and subsequent work operate in the same paradigm targeting the same research papers as source material, but again with an important difference in focus. While both systems are, to a degree, interested in *verification* of research results, *Minimalist Grammars* mean this in a purely formal sense. The idea is to identify the kernel of the Minimalist Program and establish it on a firm, formal footing to study the model, provide a template for implementation, and verify results. The emphasis on proveability and formal correctness is indispensible to the continued relevance of Minimalism, but it is also confining in three ways. First, because the formal devices used are somewhat unfamiliar to most minimalist researchers, there is a bit of a barrier to entry. Second, while the Minimalist Program can and undoubtedly will benefit from the more disciplined approach *Minimalist Grammars* take, it is nevertheless a fact that Minimalism as practiced simply doesn't work

this way. Indeed, its founding principles - the repeated insistence that it is a program and not a theory - are arguably designed to *avoid* formal specification. Third, *Minimalist Grammars* tend to be more representational than derivational. Like with the LKB, the focus isn't really on allowing language implemenators to tweak the derviational algorithm; rather, it is up to the researcher to conform to the specifiation. In a *Minimalist Grammar*, the researcher doesn't, for example, have the opportunity to attach a side-effect to a particular feature match. In this sense, *Minimalist Grammars* are perhaps at odds with the way minimalist researchers think about language.

It is expected, however, that this project will eventually converge completely with the research being done by Stabler and colleagues. The approaches diverge due to difference in emphasis, but the end goal is the same.

Minimalist Program Grammar Implementation

As with *Minimalist Grammars*, the point of departure with the *Minimalist Program Grammar Implementation* of (Fong and Ginsburg in preparation) is mainly one of emphasis. Where this project provides a platform for experimenting with different approaches to Minimalism, the MPGI distills the most plausible candidate implementation from current research and implements it for industrial use. Important practical differences include a focus on parsing and a somewhat frozen implementational system. This system, by contrast, is concerned with generation and verification of research results. While it is possible that this system will eventually incorporate the results obtained for minimalist parsing by the *Minimalist Program Grammar Implementation*, the functionality of the two systems is currently aimed at very different purposes, and there is consequently not much overlap.

Specific Contributions

A secondary purpose of this project has been to emphasize the importance of implementation in uncovering hidden assumptions and resolving theoretical disputes. Indeed, several were noticed in the course of program development.

The most important of these has been recognizing the ways in which underspecification of the derivational algorithm can lead to confusing results. Implementing a derivational system in software forces attention to details that can be glossed over in text-based explanations. For example, when **select**ed, how does that process unfold *precisely*? A typical text-basd explanation will simply note the fact that **select** occurred, but an implementor has to actually make it happen. For the present project, this meant, in the case of an item **select**ed from the active `lexical subarray`, deleting it from the `lexical subarray` and copying it into an array of active items, with the assumption that it would subsequently **merge**. This was a step necessitated by the need to eliminate a certain kind of **lookahead** from the system that is probably not obvious to the non-implementational researcher: what if the system **select**s items from the `lexical subarray` in the "wrong" order? There must, after all, be an *unmotivated* initial selection. Ideally, this initial selection is precisely the item that will, in turn, **select** the next appropriate item from the subarray. But no system can know, without arguably illegitimate guidance, which picks are the "right" ones, and no system that uses generation as a tool to verify that the extent of grammar coverage is exactly what is claimed *should* provide such guidance. The system has to be free to **select** items in any order, which means something has to be done with previously-**select**ed items that are either unable to **select** items themselves, or will **select** items that do not necessarily follow the order of selection a typical research paper would report. Many options are available, of course. One could simply return an unsuccessful **select** back to the `lexical array`. One could provide

the system with heuristics to guarantee that the *phase* head (or at least the item with the most immediate selectional feature) is always first **select**ed (this is the approach taken by this system). The point is that once one has determined that **select** actually means (something like) "identify and copy to an array of active items," it highlights the idea that there is no *a priori* reason why **merge** must immediately follow **select**, even if this is the general case. The two operations are computationally distinct[67]. Once this is understood, the EPP seems a shade less arbitrary. After all, any system that starts with `lexical array`s needs an in-principle-unmotivated operation to start the process moving in any case; this recognized, the idea of a diacritic feature that requires an object to undergo **select** and then **merge** on lexical items is nothing more than triggering a function we know the system already has. For this reason more than any, it is useful to have an *explicit* derivational algorithm, and this system has provided that.

Perhaps the next most important is the recognition that Minimalist grammar development has three core components: feature specification, algorithm specification, and triggered side effects. As far as the survey of the literature done here indicates, every tool at a minimalist researcher's disposal falls into one of these three categories. Thus, what seemed impossible in principle - building a system that allows researchers to implement *arbitrary* new minimalist theories - is probably tractable in practice, even if there is no way to guarantee coverage of every future development. What any conceivable minimalist grammar development environment, including future competitors to this project, must provide, then, are (1) a way for users to create lexical items composed of feature sets, (2) a way for users to reorder the steps the system takes, and (3) an API that allows for the addition of user-scripted "hooks" that operate on the system's object in response to system events. It is this last point that arguably separates Minimalism most sharply from more representational systems like HPSG and LFG, that

[67] In fact, it is not unknown for Minimalist theories to leverage this distinction. See (Brannigan 2010) for what is arguably an example.

shows that Minimalism, at its core, retains the derivational character of its ancestors. Certain (combinations of) features trigger certain reactions, and users of any conceivable minimalist grammar development environment must have the power to script those reactions.

Some such reactions that featured prominently in Chapter 5 were **head movement** and **head incorporation**, and there was the related queston of whether these were substantively different. The chapter offered the tentative concluson that they are the same process, that they are *not* substantively different, and that in fact the persistent question of whether **head movement** is a cyclic or post-cyclic (PF-only) phenomenon is actually a bit of a distraction. Again, an implementor must decide exactly *when* **head movement** happens, and also *how* it happens in somewhat more detail than simply showing a head attached to another head. In the present system, it was decided that, without user involvement, **head movement** happens immediately on **merge**. Moreover, **head movement** means that the *entire* feature array of a "moving" head is prepended on the front of the feature array of the receiving head. In effect, they become a single lexical item. This is a process already anticipated by (Chomsky 2008)'s addition of **feature inheritance** to the inventory of operations: **head movement** is nothing more than complete feature transfer (including phonetic features). Since features are *prepended*, there is no change in the order in which arguments are **select**ed relative to treating the merged heads as kernels of separate phrases. This allowed the system to follow the obvious heuristic (alluded to above) of having the *phase head* always be the first-**select**ed item from any `lexical subarray` that contains one without undesired ordering side effects. v is first-**select**ed, and it, in turn, **select**s V, and the v-V complex **select**s V's complement, its external argument, and then v's external argument in the usual order. People who prefer to see head movement as a PF effect can point to the fact that the ultimate sequential ordering must still reflect the underlying architecture - `spec`-v preceeds v-V preceeds `spec`-V preceeds `comp`-V. This is presumably a PF decision. People who prefer to see head movement as

syntactic can point to the feature transfer as an undeniably syntactic implementation. Both parties are correct in their way, and the debate was probably never worth having. But this is the sort of thing that isn't necessarily noticed until an implementation is pursued - whether with software or on paper - to an appropriate level of detail.

A similar contribution that "falls out" of the implementation is the realization that *phases* are not an architectural innovation in need of special enforcement by the system. They are perfectly implementable with chapter 4's *barrier feature*. All that is actually required is that the appropriate phase heads be endowed with one, guaranteeing that all **probe**s halt at the phase boundary. Notice also that a fact about *phases* that seemed synthetic to many researchers is actually quite natural: the idea that the periphery (i.e. the `spec` of the **phase** head) is accessible in a way that the `comp` is not. This is simply a consequence of the fact that the *barrier* feature is represented on the *phase* head, as is natural if it is the head that defines the *phase*. In a normal in-order tree traversal, a **probe** will his any `specs` *before* it hits the head, and it is only on hitting the head that it encounters the *barrier* feature and halts. Phases are not so mysterious after all. Like so much else, they are a factor of lexicon design.

FUTURE DIRECTIONS

Though the core of the system is in place and available for use, the project is far from complete. In broad terms, the work that remains is as follows.

Foremost, it is necessary to do what amounts to a second version of this project dealing with PF processes. The current system simply uses in-order tree traversal string concatenation, augmented by deletion, to yield the surface order. This is useful for little more than demonstrating that the syntactic component is working correctly. The number of hypothesized PF

effects has exploded in recent years, so it will be necessary to do a separate comprehensive survey of what has been proposed and to implement a new API exposing them to users. This is the next step for this project.

Secondarily, semantic hooks will need to be exposed. The distinction between Transfer - which sends information to LF - and SpellOut - which sends output to PF - is not much maintained in recent work, but in the original concept it was typical to treat SpellOut as cyclic ("multiple spellout") and Transfer as a single reading from an assembled tree. This project takes that approach: it is possible even now to add semantic "features" to `lexical items` in the `lexicon` - in the `sem` field reserved for that purpose. A semantic module would need to then traverse the tree, reading these features and assembling them appropriately. Since no particular semantic theory is favored in Minimalism (though (Heim and Kratzer 1998) is a go-to resource for non-specialists), it is best to leave the implementation up to outside module writers. Once the PF implementation is satisfactory, the next step will therefore be to collaborate with semantics module writers to make sure that the system API exposes everything that they need to do their work.

Finally, though this system is focused on sentence generation as a means of verifying research claims, there is no reason why it shouldn't eventually be expanded to include parsing. It is at this point that the system will seek to merge with work done in the *Minimalist Grammars* tradition - specifically on foundations laid in (Harkema 2001) - to run all the devices "in reverse."

REFERENCES

Abels, Klaus. 2003. "Successive Cyclicity, Anti-Locality, and Adposition Stranding." PhD thesis, University of Connecticut.

Adger, David. 2003. *Core Syntax: A Minimalist Approach*. Oxford University Press.

Aissen, Judith, and David Perlmutter. 1976. "Clause Reduction in Spanish." In *BLS 2. Proceedings of the 2nd Annual Meeting of the BLS*, edited by H. Thompson, 1–30. Linguistic Society of America.

Baker, Mark. 1988. *Incorporation: A Theory of Grammatical Function Changing*. University of Chicago Press.

———. 2003. "Syntax." In *The Handbook of Linguistics.*, edited by Mark Aronoff and Janie Rees-Miller. Wiley-Blackwell.

———. 2008. *The Syntax of Agreement and Concord*. Edited by Mark Aronoff and Janie Rees-Miller. Cambridge University Press.

Baker, Mark, Kyle Johnson, and Ian Roberts. 1989. "Passive Arguments Raised." *Linguistic Inquiry* 20 (2). MIT: 219–51.

Béjar, Susana, and Milan Rezac. 2003. "Person Licensing and the Derivation of PCC Effects." In *Romance Linguistics: Theory and Acquisition*, edited by Ana Teresa Pérez-Leroux and Yves Roberge, 49–62. Current Issues in Linguistic Theory. John Benjamins.

Bhatt, Rajesh. 2005. "Long-Distance Agreement in Hindi-Urdu." *Natural Language and Linguistic Theory* 23: 757–807.

Bhatt, Rajesh, and Roumyana Pancheva. 2006. "Implicit Arguments." In *The Blackwell Companion to Syntax*. Blackwell.

Bobaljik, Jonathan. 2002. "Realizing Germanic Inflection: Why Morphology Does Not Drive Syntax." *Journal of Comparative Germanic Linguistics* 6: 129–67.

Bobaljik, Jonathan, and Samuel Brown. 1997. "Interarboreal Operations: Head Movement and the Extension Requirement." *Linguistic Inquiry*, no. 28: 345–56.

Bobaljik, Jonathan, and Idan Landau. 2009. "Icelandic Control Is Not a-Movement: The Case from Case." *Linguistic Inquiry*, no. 40: 269–80.

Bobaljik, Jonathan, and Susi Wurmbrand. 2005. "The Domain of Agreement." *Natural Language and Linguistic Theory* 23: 809–65.

Boeckx, Cedric. 2008a. *Aspects of the Theory of Agreement*. Routledge Leading Linguists. Routledge.

———. 2008b. *Bare Syntax*. Oxford University Press.

———. 2008c. *Understanding Minimalist Syntax: Lessons from Locality in Long-Distance Dependencies*. Wiley.

Boeckx, Cedric, and Kleanthes Grohmann. 2007. "Remark: Putting Phases in Perspective." *Syntax*, no. 2. Blackwell Publishing Ltd.: 204–22.

Boeckx, Cedric, Norbert Hornstein, and Jairo Nunes. 2010. *Control as Movement*. Cambridge University Press.

Boskovic, Jelko, and Jairo Nunes. 2007. "The Copy Theory of Movement: A View from PF." In *The Copy Theory of Movement*, edited by Norbert Corver and Jairo Nunes. Linguistik Aktuell/Lingusitics Today. John Benjamins Publishing Company.

Bouchard, Denis. 1984. *On the Content of Empty Categories*. Studies in Generative Grammar. Foris.

Brannigan, Phil. 2010. *Provocative Syntax*. Linguistic Inquiry Monographs. MIT.

Bray, T. 2014. "The JavaScript Object Notation (JSON) Data Interchange Format." RFC 7159. Internet Engineering Task Force. https://tools.ietf.org/html/rfc7159.

Bresnan, Joan. 1982. "Control and Complementation." *Linguistic Inquiry* 13 (3): 343–434.

———. 2001. *Lexical-Functional Grammar*. Blackwell Publishers Ltd.

Chandra, Pritha. 2007. "[Dis]agree: Movement and Agreement Reconsidered." PhD thesis, University of Maryland.

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

———. 1981. *Lectures on Government and Binding: The Pisa Lectures*. Walter de Gruyter.

———. 1986a. *Barriers*. Linguistic Inquiry Monographs. MIT Press.

———. 1986b. *Knowledge of Language: Its Nature, Origin and Use*. Praeger.

———. 1991. "Some Note on Economy of Derivation and Representation." In *Principles and Parameters in Comparative Grammar*, edited by Robert Freidin, 417–54. The MIT Press.

———. 1993. "A Minimalist Program for Linguistic Theory." In *The View from Building 20: Essays in Linguistics in Honor of Sylvain Bromberger*, edited by Ken Hale and Samuel J. Keyser. The MIT Press.

———. 1995. *The Minimalist Program*. MIT Press.

———. 2000. "Minimalist Inquiries: The Framework." In *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik*, edited by Roger Martin and Juan Uriagereka. MIT Press.

———. 2001. "Derivation by Phase." In *Ken Hale: A Life in Language*, edited by Michael Kenstowicz. MIT Press.

———. 2005. "Three Factors in Language Design." *Linguistic Inquiry* 36 (1).

———. 2008. "On Phases." In *Foundational Issues in Linguistic Theory: Essays in Honor of Jean-Roger Vergnaud*, edited by Robert Freidin, Carlos P. Otero, and Maria Luisa Zubizarreta. The MIT Press.

Chomsky, Noam, and Howard Lasnik. 1993. "The Theory of Principles and Parameters." In *Syntax: An International Handbook of Contemporary Research*, edited by J. Stechow, A. Jacobs, W. Sternefeld, and T. Vennemann. De Gruyter.

Citko, Barbara. 2014. *Phase Theory: An Introduction*. Research Surveys in Linguistics. Cambridge University Press.

Clements, G. N. 1985. "The Geometry of Phonological Features." In *Phonology Yearbook 2*, edited by Collin Ewen and John Anderson, 225–52. Cambridge University Press.

Collins, Chris, and Edward Stabler. 2016. "A Formalization of Minimalist Syntax." *Syntax* 19 (1): 43–78.

Copestake, Ann. 2002. *Implementing Typed Feature Structure Grammars*. CSLI.

Copestake, Ann, and Dan Flickinger. 2000. "An Open-Source Grammar Development Environment and Broad-Coverage English Grammar Using HPSG." In *Proceedings of the Second Conference on Language Resources and Evaluation (LREC-2000)*. Athens.

Culicover, Peter W., and Ray Jackendoff, eds. 2005. *Simpler Syntax*. Oxford University Press.

den Dikken, Marcel. 2014. "On Feature Interpretability and Inheritance." In *Minimalism and Beyond: Radicalizing the Interfaces*, edited by Peter Kosta, Steven Franks, Teodora Radeva-Bork, and Lilia Schürcks. John Benjamins Publishing Company.

Epstein, Samuel David, and T. Daniel Seely. 2006. *Derivations in Minimalism*. Cambridge University Press.

Epstein, Samuel, and Norbert Hornstein. 1999. "Working Minimalism: Introduction." In *Working Minimalism*. The MIT Press.

Felser, Claudia. 2004. "Wh-Copying, Phases, and Successive Cyclicity." *Lingua* 114 (5). MIT Press: 543–74.

Flickinger, Dan, Emily M. Bender, and Stephan Oepen. 2014. "Towards an Encyclopedia of Compositional Semantics: Documenting the Interface of the English Resource Grammar." In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC14)*, edited by Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, 875–81. Reykjavik, Iceland: European Language Resources Association (ELRA).

Fong, Sandiwey, and Jason Ginsburg. in preparation. *The Minimalist Machine*. Research Monograph.

Fox, Danny, and David Pesetsky. 2005. "Cyclic Linearization of Syntactic Structure." *Theoretical Linguistics* 31. MIT Press: 1–46.

Franks, Steven, and James Lavine. 2006. "Case and Word Order in Lithuanian." *Journal of Linguistics* 42: 239–88.

Fukui, Naoki. 1993. "A Note on Improper Movement." *The Linguistic Review* 10 (2): 111–26.

Funakoshi, Kenji. 2013. "Syntactic Head Movement and Its Consequences." PhD thesis, Maryland.

Gallego, Angel J. 2010. *Phase Theory*. John Benjamins Publishing Company.

Haegeman, Liliane. 1994. *Introduction to Government and Binding Theory*. Blackwell Publishers Ltd.

Harkema, Hendrik. 2001. "Parsing Minimalist Languages." PhD thesis, University of

California at Los Angeles.

Harris, Zellig. 1951. *Methods in Structural Linguistics*. University of Chicago Press.

Heim, Irene, and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Wiley-Blackwell.

Herring, Joshua. 2009. "Call by Reference Syntax: Toward a Pointer Implementation of (Re)merge."

Hiraiwa, Ken. 2005. "Dimensions of Symmetry in Syntax: Agreement and Clausal Architecture." PhD thesis, MIT.

Holmberg, Anders. 1986. "Word Order and Syntactic Features in the Scandinavian Languages and English." PhD thesis, University of Stockholm.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2001. *Introduction to Automata Theory, Languages, and Computation*. 2nd ed. Pearson.

Hornstein, Norbert. 1999. "Movement and Control." *Linguistic Inquiry* 30 (1).

———. 2000. *Move! A Minimalist Theory of Construal*. Wiley.

Hornstein, Norbert, and William Idsardi. 2014. "A Program for the Minimalist Program." In *Radicalizing the Interfaces*. John Benjamins Publishing Company.

Hornstein, Norbert, and Maria Polinsky. 2010. "Control as Movement: Across Languages and Constructions." In *Movement Theory of Control*. Linguistik Aktuell. John Benjamins.

Hornstein, Norbert, Jairo Nunes, and Kleanthes K. Grohman. 2005. *Understanding Minimalism*. Cambridge University Press.

Joshi, Aravind, and Yves Schabes. 1997. "Tree Adjoining Grammars." In *Handbook of Formal Languages Vol 3: Beyond Words*, edited by Grzegorz Rozenberg and Arto Salomaa. Vol. 3. Springer.

Kayne, Richard S. 1994. *The Antisymmetry of Syntax*. Linguistic Inquiry Monographs 25. The MIT Press.

Landau, Idan. 2000. *Elements of Control: Structure and Meaning in Infinitival Constructions*. Kluwer.

———. 2003. “Movement Out of Control.” *Lingusitic Inquiry* 34 (3). MIT: 471–98.

———. 2004. “The Scale of Finiteness and the Calculus of Control.” *Natural Language and Linguistic Theory* 22 (4): 811–77.

———. 2006. “Severing the Distribution of PRO from Case.” *Syntax* 9 (2): 153–70.

———. 2008. “Two Routes of Control: Evidence from Case Transmission in Russian.” *Natural Language and Linguistic Theory* 26 (4). Springer: 877–924.

———. 2015. *A Two-Tiered Theory of Control*. Linguistics Inquiry Monographs. MIT.

Lasnik, Howard. 1999. *Minimalist Analysis*. Blackwell.

Lasnik, Howard, and Mamoru Saito. 1991. “On the Subject of Infinitives.” In *Papers from the 27th Regional Meeting of CLS*, 324–43. Chicago, IL: CLS.

Lightfoot, David. 1979. *The Principles of Diachronic Syntax*. Cambridge Studies in Linguistics. Cambridge University Press.

Matthew, Rosmin. 2015. *Head Movement in Syntax*. John Benjamins.

McGinnis, Martha. 2005. “On Markedness Asymmetries in Person and Number.” *Language* 81 (3): 699–718.

Melcuk, Igor, and Nikolai V. Pertsov. 1987. *Surface Syntax of English: A Formal Model Within the Meaning-Text Framework*. John Benjamins Publishing Company.

Merchant, Jason. 2011. “Aleut Case Matters.” In *Pragmatics and Autolexical Grammar: In Honor of Jerry Sadock*, edited by Etsuyo Yuasa Yuasa, Tista Bagchi, and Katharine P. Beals, 382–411. John Benjamins.

Müller, Gereon, and Wolfgang Sternefeld. 1993. “Improper Movement and Unambiguous Binding.” *Linguistic Inquiry* 24 (3): 461–507.

Narita, Hiroki. 2011. “Phasing in Full Interpretation.” PhD thesis, Harvard.

Nevins, Andrew. 2007. “The Representation of Third Person and Its Consequences for Person-Case Effects.” *Natural Language and Linguistic Theory* 25 (2): 273–313.

Nunes, Jairo. 2004. *Linearization of Chains and Sideward Movement*. MIT Press.

Okasaki, Chris. 1999. *Purely Functional Data Structures*. Cambridge University Press.

Ouali, Hamid. 2008. "On c-to-T Phi-Feature Transfer: The Nature of Agreement and Anti-Agree- Ment in Berber." In *Agreement Restrictions*, edited by Roberta D'Alessandro, Susann Fischer, and Gunnar Hrafn Hrafnbjargarson. Walter de Gruyter.

Pesetsky, David, and Esther Torrego. 2007. "The Syntax of Valuation and the Interpretability of Features." In *Phrasal and Clausal Architecture*, edited by Wendy Wilkins Simin Karmini Vida Samiian, 355–426. John Benjamins.

Pollard, Carl, and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar*. CSLI.

Pollock, Jean-Yves. 1989. "Verb Movement, Universal Grammar and the Structure of IP." *Linguistic Inquiry* 20. MIT Press: 365–425.

Postal, Paul. 1974. *On Raising: An Inquiry into One Rule of English Grammar and Its Theoretical Implications*. Current Studies in Linguistics. MIT.

Preminger, Omer. 2014. *Agreement and Its Failures*. Vol. 68. Linguistic Inquiry Monographs. MIT.

Richards, Marc. 2010. "Deriving the Edge: What's in a Phase." *Syntax*, no. 14: 74–95.

Rizzi, Luigi. 1978. "A Restructuring Rule in Italian Syntax." In *Recent Transformational Studies in European Languages*, edited by Samuel Jay Keyser, 113–58. MIT Press.

Roberts, Ian. 2011. "Head Movement and the Minimalist Program." In *The Oxford Handbook of Linguistic Minimalism*, edited by Cedric Boeckx. Oxford University Press.

Sag, Ivan, Tom Wasow, and Emily Bender. 2003. *Syntactic Theory: A Formal Introduction*. University of Chicago Press.

Sedgewick, Robert, and Kevin Wayne. 2011. *Algorithms*. 4th ed. Addison-Wesley Professional.

Sigurdsson, Halldor Armand. 2008. "The Case of PRO." *Natural Language and Linguistic Theory*. Springer.

Stabler, Edward. 1997. "Derivational Minimalism." In *Logical Aspects of Computational Linguistics*, edited by Christian Retore, 68–95. Springer.

———. 2001. "Minimalist Grammars and Recognition." In *Linguistic Form and Its Computation*. CSLI.

———. 2011. “Computational Perspectives on Minimalism.” In *Oxford Handbook of Linguistic Minimalism*, edited by Cedrick Boeckx, 617–42. Oxford University Press.

Strachey, Christopher, and Christopher P. Wadsworth. 1972. *Continuations: A Mathematical Semantics for Handling Full Jumps*. Technical Monograph: Oxford University Computing Laboratory.

Taraldsen, Knut. 1995. “On Agreement and Nominative Objects in Icelandic.” In *Studies in Comparative Germanic Syntax*, edited by Hubert Haider, Susan Olsen, and Sten Vikner. Kluwer.

Thrainsson, Haldur. 1986. “On Auxiliaries, AUX and VPs in Icelandic.” In *Topics in Scandinavian Syntax*, edited by L. Hellan and K. K. Christensen, 235–66. Dordrecht.

Travis, Lisa. 1984. “Parameters and Effects of Word Order Variation.” PhD thesis, MIT.

Urk, Coppe van. 2013. “Visser’s Generalization: The Syntax of Control and the Passive.” *Linguistic Inquiry* 44 (1): 168–78.

Vikner, Sten. 2005. “Object Shift.” In *The Blackwell Companion to Syntax*, edited by Martien Evaert and Henrik van Riemsdijk. Vol. 3. Blackwell Publishing Ltd.

Wurmbrand, Susi. 2001. *Infinitives: Restructuring and Clause Structure*. Vol. 55. Studies in Generative Grammar. Mouton de Gruyter.

———. 2015. “Restructuring Cross-Linguistically.” In *NELS 45. Proceedings of the North Easter Linguistic Society Annual Meeting 45*, edited by Thuy Bui and Deniz Özyıldız, 227–40. University of Massachusetts Amherst: GLSA.

Zeijlstra, Hedde. 2012. “There Is Only One Way to Agree.” *The Lingusitic Review* 29 (3). De Gruyter: 491–539.

APPENDIX A - VISSER'S GENERALIZATION IN DETAIL: A DERIVATIONAL EXAMPLE

This section repeats the system derivation of the Movement Theory of Control [@hornstein:polinsky:10]treatment of Visser's Generalization from Chapter 5 in greater detail for readers with a special interest in the operation of the grammar toolkit. The reader is expected to be familiar with arguments made in chapters 4 and 5 - especially the overview of the MTC analysis of Visser's Generalization, as discussion in that section will be relevant to understanding where this derivation (potentially) goes wrong.

It should be emphasized again that the interpretive component is not a part of this project; it is left for future implementation by qualified semanticists. This commits this system to a clean separation between **SpellOut** and **Transfer**, at least for now - with **SpellOut** being a cyclic process that assembles the input to PF and **Transfer** a post-derivational process that assembles the input to LF. The assumption is that the interpretive process consumes the completed tree as input.

THE MOVEMENT THEORY OF CONTROL

This subsection repeats and augments the text from chapter 5 associated with the MTC derivation of *John*$_i$ *promised Mary* ~~*John*~~$_i$ *to win* to give a clearer illustration of how the system operates. Items used are taken from the lexicon given in Appendix C. Output has been somewhat simplified from the actual system output for reasons of space and formatting requirements.

One point of likely controversy in this derivation and others that (the default version of) this system performs will involve the ordering of **head incorporation**, which frequently comes *before* the `merge` of the incorporating head's arguments. It was argued in chapter 4 that this move is conceptually more sound while yielding the same ultimate output, and the derivation below illustrates why this is so. In effect, `head incorporation` is a way for a system to

allow a construct to accept more than two arguments while maintaining a binary branching representation.

The lexical array is:

John, promise, to-P, Mary, to_{inf}, win, v_{trans}, $\text{v}_{\text{ditrans}}$, T-past, C

grouped into subarrays as follows:

{{win, $\text{v}_{\text{intrans}}$}, {John}, {to-P, Mary}, {to_{inf}}, {promise, $\text{v}_{\text{ditrans}}$}, {T-past, C}}

(1) Create an instance of a `derivation` class and pass this `lexical array` to it, along with the default `strategy`.

(2) The `derivation` determines it is in the **start** state based on the fact that it has a populated `lexical array` and an empty `history` array.

```
{
  lexical_array: [
    [{"v_intrans"}, {"win"}],
    [{"John"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [],
  active_stages: [],
  state: "start"
}
```

(3) Because it is in the "start" state, it creates a `stage` from the first item in its `lexical array` and makes this the active stage. The `stage` likewise determines it is in a

start state based on the fact that it has a populated `lexical array` and an empty `workspace`:

```
DERIVATION
{
  lexical_array: [
    [{"John"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [<stage_1>],
  active_stages: [<stage_1>],
  state: "active",
  convergent: false
}

STAGE_1
{
  lexical_array: [{"v_intrans"}, {"win"}],
  workspace: [],
  state: "designate",
  convergent: false
}

ASSEMBLED TREES
none
```

(4) The `derivation` now passes the active stage to its `strategy` function (the system default). Recall from chapter 4 that a `strategy` is a function that takes a `stage` as input

and returns a new `stage` as output. Based on this `stage`, the `derivation` then decides whether it is `convergent` or `divergent`, and if `divergent` whether pathologically so. If is it `convergent` or pathologically `divergent`, it will halt. Otherwise, it places the new `stage` on the `history` array. In the general case, it will also replace the old active `stage` with this new `stage`. However, this is not always the case. Sometimes it will retain a pointer to a previous `stage`. Examples of this follow, but the basic idea is that it does this when an individual stage has completed but does not represent a completion of the derivation (i.e. the information it holds will have to be reintegrated into the derivation at a later stage). In this particular case, the `strategy` determines that the `stage` should choose an active item. In the default case, this is done with a heuristic based on which of the items in the `lexical array` can **select** which other items and prefers those. In this array, `v_intrans` can select `win` but not the other way around, so `v_intrans` is picked first.

```
DERIVATION
{
  lexical_array: [
    [{"John"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    <stage_2>],
  active_stages: [<stage_2>],
  state: "active",
  convergent: false
}
```

```
STAGE_2
{
  lexical_array: [{"win"}],
  workspace: [{"v_intrans"}],
  state: "probe",
  convergent: false
}

ASSEMBLED TREES
none
```

(5) Since `stage_2` is in an **active** state, it gets passed again to the `strategy` which returns `stage_3`. Where `stage_1` represented the output of the **Activate** part of the cycle outlined in chapter 4, `stage_2` and `stage_3` are in the **Perform** cycle. `stage_3` is specifically in the **Probe** part of the cycle. Because the default `strategy` enforces the **merge-over-move** constraint, the active item `v_intrans` first `probe`s in `stage_3`'s `lexical array`[ˆnochoice]. `v_intrans` has a **selector** feature (`=V[]`) that matches a **selectee** feature on `win` (`cV[win]`), as the reader may verify by consulting the `lexicon`. Therefore, `stage_3` has **select**ed `win` into the `workspace`, marking it as syntactically active.

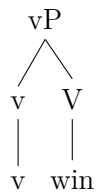
```
DERIVATION
{
  lexical_array: [
    [{"John"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
```

```
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    <stage_2>,
    <stage_3>],
  active_stages: [<stage_3>],
  state: "active",
  convergent: false
}

STAGE_3
{
  lexical_array: [],
  workspace: [{"v_intrans"}, {"win"}],
  state: "react",
  convergent: false
}

ASSEMBLED TREES
none
```

(6) Since the features involved in the selection in `stage_3` were **select** features, and `merge` is a side-effect of `select`, the `strategy` produces a `stage_4` in which the two active items in the `workspace` have `merge`d. The `ASSEMBLED TREES` section shows the `syntactic object` for `v_intrans` with a single member in its `constituents` array, indicating that `win` is now its `complement`. `win` here should be understood as a pointer to another `syntactic object` representing `win`, which of course has no items in its constituents array.

```
DERIVATION
{
  lexical_array: [
    [{"John"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    <stage_2>,
    <stage_3>,
    <stage_4>],
  active_stages: [<stage_4>],
  state: "active",
  convergent: false
}

STAGE_4
{
  lexical_array: [],
  workspace: [{"v_intrans", "win"}],
  state: "react",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
```

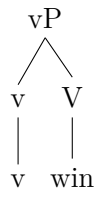
```
    "cv[intrans]",
    "=V[]",
    "=D[]],
  sem: "",
  phon: "",
  constituents: ["win"]
}
```



(7) `probe`s happen in response to `merge`, first from the **select**or and then, in the opposite direction, from the **select**ee. In the interest of space, these steps have been collapsed here (since there are no relevant `probe`s on `win` anyway)[^morestages].

```
DERIVATION
{
  lexical_array: [
    [{"John"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    <stage_2>,
    <stage_3>,
```

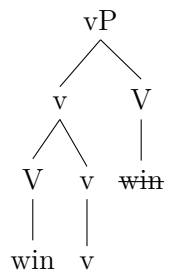
```
    <stage_4>,
    <stage_5>],
  active_stages: [<stage_5>],
  state: "active",
  convergent: false
}

STAGE_5
{
  lexical_array: [],
  workspace: [{"v_intrans", "win"}],
  state: "react",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cv[intrans]",
    "=V[win]",
    "=D[]],
  sem: "",
  phon: "",
  constituents: ["win"]
}
```



(8) The default `strategy` follows `value` with a `compact` step that handles both elimination of *valued*, *uninterpretable* features[ˆcatfeat] and any necessary **head incorporation**. For the implementation being demonstrated, **head incorporation** doesn't follow from any linguistic principles - it is simply coded into the default `strategy` as a side-effect of valuation of a `=V[]` feature. Recall from chapter 4 that **head incorporation** in this system is simply a form of feature-sharing - specifically **donate**, in which all features from a given head are removed from that head's **syn** list and prepended to the featurelist of a receiving head[ˆphoninv]. `stage_6` is in an **complete** state because there are no more actions the `stage` can plausibly take, as all `probe` and `compact` steps have completed, and there are no more items in its `lexical array` to draw from. Note that it is still not **convergent**, however, as there are unvalued features on some of its lexical items.

```
DERIVATION
{
  lexical_array: [
    [{"John"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
```

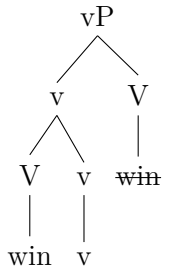
```
    <stage_6>],
  active_stages: [<stage_6>],
  state: "active",
  convergent: false
}

STAGE_6
{
  lexical_array: [],
  workspace: [{"v_intrans", "win"}],
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[]],
  sem: "",
  phon: "",
  constituents: ["win"]
}
```



(9) Because the active stage is **complete** but not **convergent**, the `derivation` leaves it on the list of active stages and begins the cycle anew with the next `lexical array` in its list.

```
DERIVATION
{
  lexical_array: [
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_7>],
  active_stages: [<stage_7>, <stage_6>],
  state: "active",
  convergent: false
}

STAGE_7
{
  lexical_array: [{"John"}],
```

```
  workspace: [],
  state: "designate",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[]],
  sem: "",
  phon: "",
  constituents: ["win"]
}
```

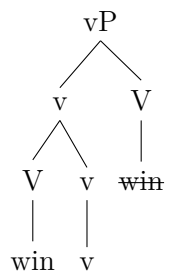


(10) As before, since `stage_7` is in state "designate," the `strategy` will instruct it to `select` from its `lexical array`. There is only one possible target. Again, some minor steps are skipped here in which the `strategy` creates some intermediate `stage`s on the way to determining that `stage_7` is "born completed" - but also is not **convergent**. It is thus added to the `history` array but also retained as an active `stage`.

```
DERIVATION
```

```
{
  lexical_array: [
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_7>],
  active_stages: [<stage_7>, <stage_6>],
  state: "active",
  convergent: false
}

STAGE_7
{
  lexical_array: [],
  workspace: [{"John"}],
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[]],
```

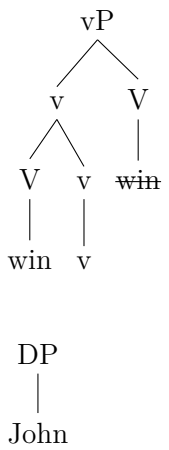
```
  sem: "",
  phon: "",
  constituents: ["win"]
}
```



(11) Because the `derivation` aims at having a single, **convergent** stage, it next compacts the existing active stages.

```
DERIVATION
{
  lexical_array: [
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_8>],
  active_stages: [<stage_8>],
  state: "active",
  convergent: false
}
```

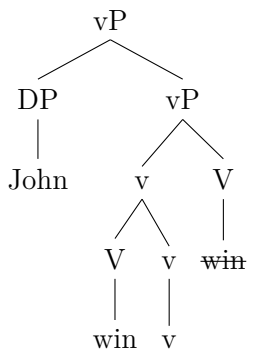
```
STAGE_8
{
  lexical_array: [],
  workspace: [{"v_intrans", "win"}, {"John"}],
  state: "probe",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[]],
  sem: "",
  phon: "",
  constituents: ["win"]
},
{
  tag: "John",
  syn: [
    "cD[john]",
    "uCase[]"],
  sem: "",
  phon: ""
}
```



(12) The `V-v` tree is the active element (for no other reason than it is leftmost in the `workspace`), so it `probe`s. Again, the demonstration skips some steps here in the interest of space and clarity, but as before, `v_intrans` will, by **merge-over-move**, try to `probe` the `lexical array` before `probe`ing in the `workspace`, and will of course find nothing, as the array is empty. It then proceeds to the `workspace`, taking each item in turn. There is only one item - `john` - and the `probe` there is successful. Because it is a **select**or `probe` matching a **select**ee feature, `merge` is triggered. Each head involved then `probe`s the other with any remaining `probe`s. There are none on `v_intrans`. `john` has a `uCase[] probe`, but it fails to find a match[ˆtreedrawing]. `stage_9` is thus **complete** but **divergent**.

```
DERIVATION
{
  lexical_array: [
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
```

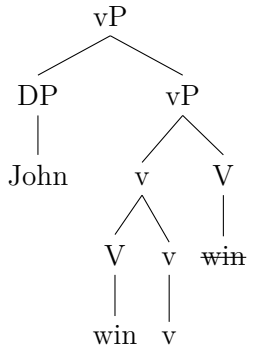
```
    ...,
    <stage_9>],
  active_stages: [<stage_9>],
  state: "active",
  convergent: false
}

STAGE_9
{
  lexical_array: [],
  workspace: [{"John", {"v_intrans", "win"}}],
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[john]],
  sem: "",
  phon: "",
  constituents: ["win", "John"]
}
```



(13) Because `stage_9` is **complete** but **divergent**, it remains on the active array. The `derivation` creates a new `stage` from the next item in the `lexical array`.

```
DERIVATION
{
  lexical_array: [
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_10>],
  active_stages: [<stage_10>, <stage_9>],
  state: "active",
  convergent: false
}

STAGE_10
{
  lexical_array: [{"to-inf"}],
  workspace: [],
```

```
  state: "designate",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[john]],
  sem: "",
  phon: "",
  constituents: ["win", "John"]
}
```

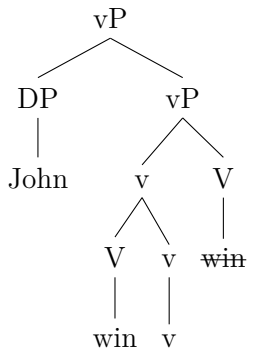


(14) As before, there is no other item in the `lexical array`, so the stage simply completes without further ado (after, of course, failing to `probe`).

```
DERIVATION
{
  lexical_array: [
```

```
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_11>],
  active_stages: [<stage_11>, <stage_9>],
  state: "incomplete",
  convergent: false
}

STAGE_11
{
  lexical_array: [],
  workspace: [{"to-inf"}],
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[john]],
  sem: "",
  phon: "",
  constituents: ["win", "John"]
```

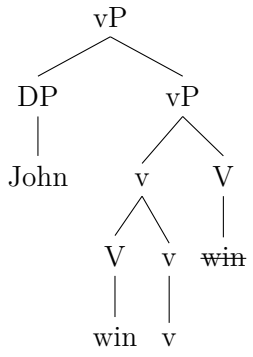
```
}
```



(15) Once again, when the `derivation` is in an "incomplete" state with more than one active stage, it will compact them.

```
DERIVATION
{
  lexical_array: [
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_12>],
  active_stages: [<stage_12>],
  state: "active",
  convergent: false
}

STAGE_12
{
```

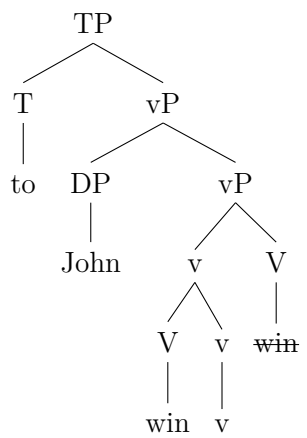
```
  lexical_array: [],
  workspace: [{"to-inf"}, {"John", {"v_intrans", "win"}}],
  state: "probe",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_intrans",
  syn: [
    "cV[win]",
    "cv[intrans]",
    "=D[john]],
  sem: "",
  phon: "",
  constituents: ["win", "John"]
}
```



(16) `to-inf` probes the `lexical array` and finds it empty. Then it `probe`s the assembled `syntactic object` representing `v` ("John win") and finds a match for its **select** (`=v[]`) feature. As always, `merge` results.

```
DERIVATION
{
  lexical_array: [
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_13>],
  active_stages: [<stage_13>],
  state: "active",
  convergent: false
}

STAGE_13
{
  lexical_array: [],
  workspace: [{"to-inf", {"John", {"v_intrans", "win"}}}],
  state: "react",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[]"],
```

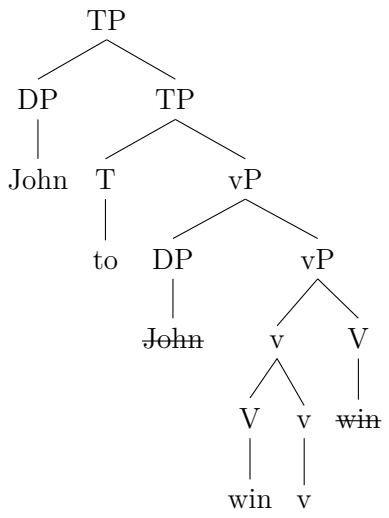
```
  sem: "",
  phon: "",
  constituents: ["v_intrans"]
}
```



(17) After `merge`, the `stage` is in a "react" `state` and so enters a mutual `probe` cycle. This happens to involve another `select` feature, and so begins the first instance of "move." As per standard assumption, this is just `re-merge`. The `probe` starts with `v_intrans`, finds no match, and so (recursively) continues along the `constituents` array of `v_intrans`. `win` is of course featurally empty, and in any case `john` is higher so is probed first. It finds a match on `john`. `John` now appears in the `constituents` arrays of two objects - a multiattachment implementation. `John` again tries its `case` probe but is unable to find a match.

```
DERIVATION
{
  lexical_array: [
    [{"to-P"}, {"Mary"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
```

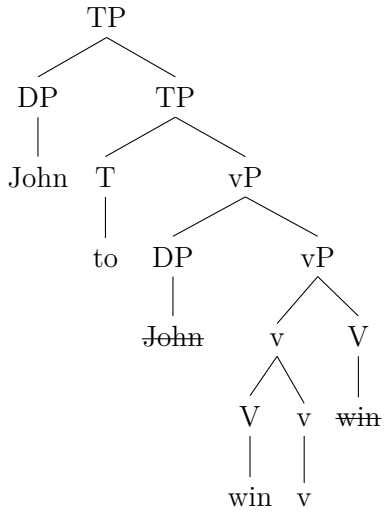
```
  history: [
    <stage_1>,
    ...,
    <stage_14>],
  active_stages: [<stage_14>],
  state: "incomplete",
  convergent: false
}

STAGE_14
{
  lexical_array: [],
  workspace: [{"John", {"to-inf", {"John", {"v_intrans", "win"}}}}],
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
}
```



(18) Finding itself in an "incomplete" state again, the `derivation` pulls out the next item from its `lexical array` and makes a new `astage`

```
DERIVATION
{
  lexical_array: [
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_15>],
  active_stages: [<stage_15>, <stage_14>],
  state: "active",
  convergent: false
}

STAGE_15
{
```

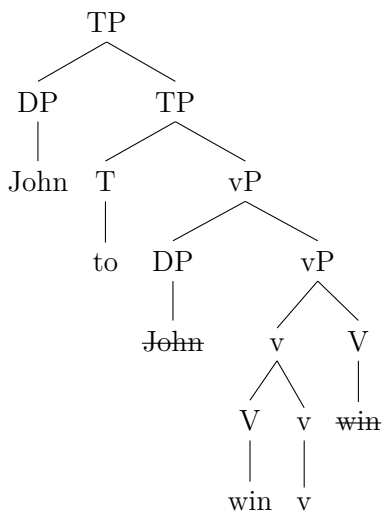
```
  lexical_array: [{"to-P"}, {"Mary"}],
  workspace: [],
  state: "designate",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
}
```



(19) Again using as a selection heuristic that one element can select the other but not the other way around, the `stage` picks `to-P`

```
DERIVATION
{
  lexical_array: [
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_16>],
  active_stages: [<stage_16>, <stage_14>],
  state: "active",
  convergent: false
}

STAGE_16
{
  lexical_array: [{"Mary"}],
  workspace: [{"to-P"}],
  state: "probe",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
```

```
  phon: "",
  constituents: ["v_intrans", "John"]
}
```

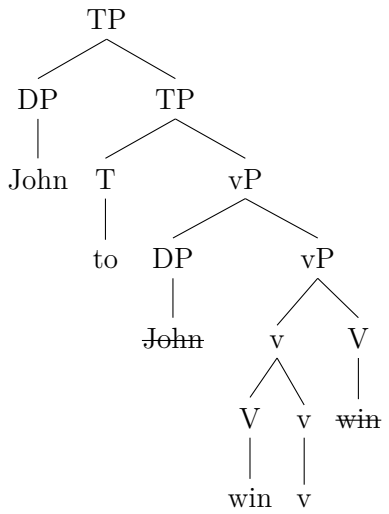


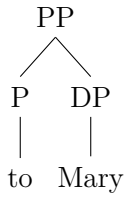
(20) `to-P` finds a match for its **select** feature, so `Mary` is moved to the `workspace`, `merge` results, and the two `merge`d items `probe` each other. This results in a valuation of the case feature on `Mary`.[^skipreact]

```
DERIVATION
{
  lexical_array: [
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_17>],
  active_stages: [<stage_17>, <stage_14>],
  state: "active",
```

```
  convergent: false
}

STAGE_17
{
  lexical_array: [],
  workspace: [{"to-P", "Mary"}],
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
},
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
```

```
  constituents: ["Mary"]
}
```



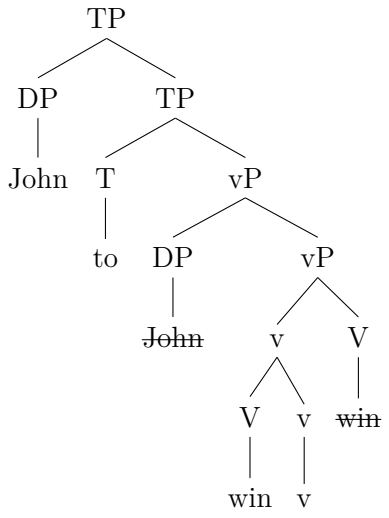


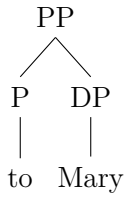
(21) Again, the `derivation` compacts the active stages. However, this time it doesn't help - there are no features on either element that will allow for **select**ion of the other.

```
DERIVATION
{
  lexical_array: [
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
```

```
    <stage_18>],
  active_stages: [<stage_18>],
  state: "active",
  convergent: false
}

STAGE_18
{
  lexical_array: [],
  workspace: [
    {"to-P", "Mary"},
    {"John", {"to-inf", {"John", {"v_intrans", "win"}}}}
  ],
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
},
{
  tag: "to-P",
```

```
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
}
```



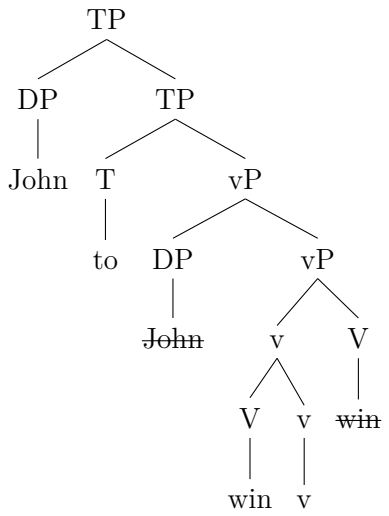


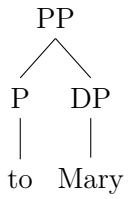
(22) So the `derivation` must create a new `stage` from an item in its `lexical array`.

```
DERIVATION
{
  lexical_array: [
```

```
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_19>],
  active_stages: [<stage_19>, <stage_18>],
  state: "incomplete",
  convergent: false
}

STAGE_19
{
  lexical_array: [{"promise"}, {"v_ditrans"}],
  workspace: []
  state: "designate",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
},
{
```

```
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
}
```



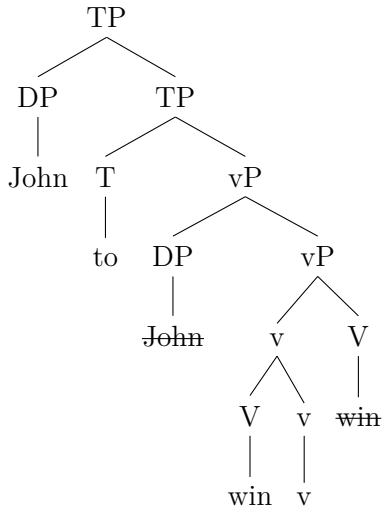


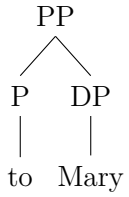
(23) Heuristics select the item that can select the other.

```
DERIVATION
{
```

```
  lexical_array: [
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_20>],
  active_stages: [<stage_20>, <stage_18>],
  state: "incomplete",
  convergent: false
}

STAGE_20
{
  lexical_array: [{"promise"}],
  workspace: [{"v_ditrans"}]
  state: "probe",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
},
```

```
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
}
```



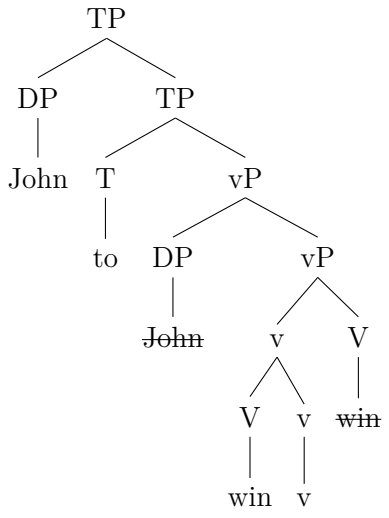


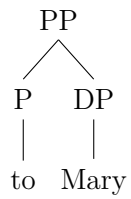
(24) Glossing over the details of the `probe` cycle, `promise` is **selected** and `merge`d

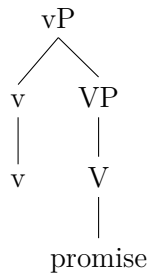
```
DERIVATION
{
```

```
  lexical_array: [
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_21>],
  active_stages: [<stage_21>, <stage_18>],
  state: "incomplete",
  convergent: false
}

STAGE_21
{
  lexical_array: [],
  workspace: [{"v_ditrans", "promise"}]
  state: "react",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
},
```

```
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
},
{
  tag: "v_ditrans",
  syn: [
    "cv[ditrans]",
    "=VDitrans[promise]",
    "=D[]"]
  sem: "",
  phon: "",
  constituents: ["promise"]
}
```





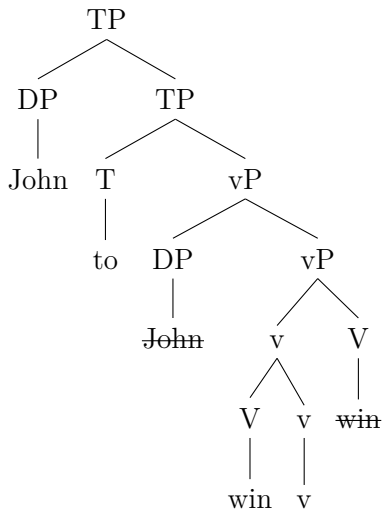


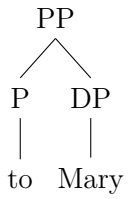
(25) Again, there is **head movement**

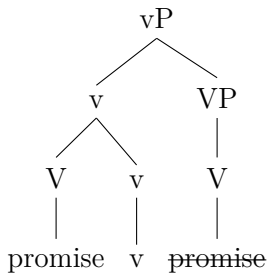
```
DERIVATION
{
  lexical_array: [
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_22>],
  active_stages: [<stage_22>, <stage_18>],
  state: "incomplete",
  convergent: false
}

STAGE_22
{
  lexical_array: [],
```

```
  workspace: [{"v_ditrans", "promise"}]
  state: "complete",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
},
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
},
{
  tag: "v_ditrans",
  syn: [
    "cVDitrans[promise]",
```

```
    "=T[]",
    "=PDat[]",
    "cv[ditrans]",
    "=VDitrans[promise]",
    "=D[]"],
  sem: "",
  phon: "",
  constituents: ["promise"]
}
```





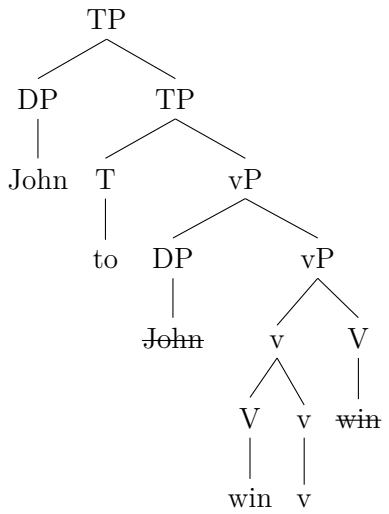


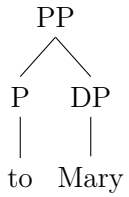
(26) Again, the stages compact

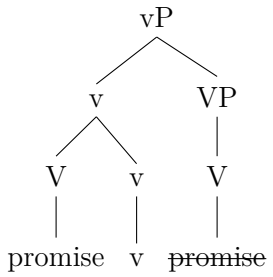
```
DERIVATION
{
  lexical_array: [
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_23>],
  active_stages: [<stage_23>],
  state: "active",
  convergent: false
}

STAGE_23
{
  lexical_array: [],
  workspace: [
    {"v_ditrans", "promise"},
    {"to-P", "Mary"},
    {"John", {"to-inf", {"John", {"v_intrans", "win"}}}}
  ],
  state: "probe",
  convergent: false
}

ASSEMBLED TREES
{
```

```
  tag: "to_inf",
  syn:[
    "cT[to]",
    "=v[intrans]",
    "=D[john]"],
  sem: "",
  phon: "",
  constituents: ["v_intrans", "John"]
},
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
},
{
  tag: "v_ditrans",
  syn: [
    "cVDitrans[promise]",
    "=T[]",
    "=PDat[]",
    "cv[ditrans]",
    "=VDitrans[promise]",
    "=D[]"],
  sem: "",
  phon: "",
```

```
  constituents: ["promise"]
}
```





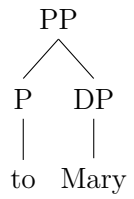


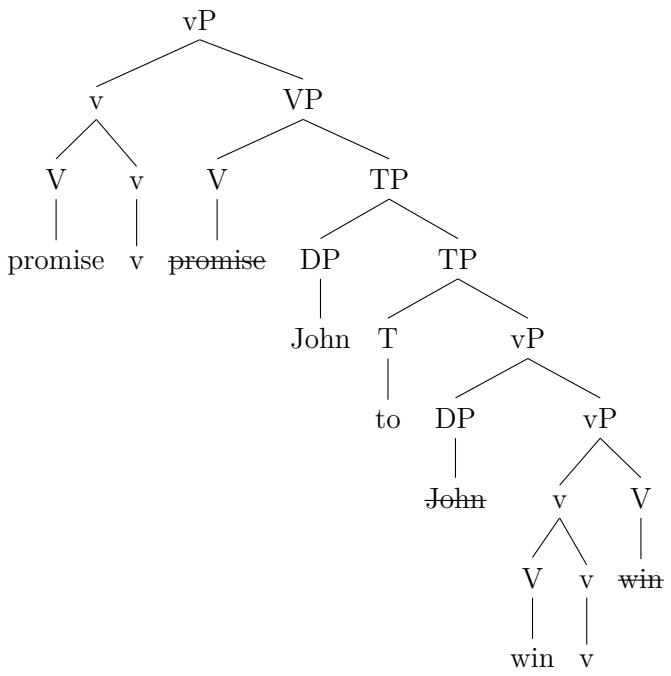
(27) The first active `probe` on `v_ditrans` - which is `=T[]` after **head incorporation** - will find no match on the `to_p`, but will on the TP, resulting, as always, in `merge` and mutual `probe`. This technically results in the TP `merge`ing with v, but as noted in chapter 4, this is ultimately notational. The relative hierarchy of elements is preserved, with `mary` "outranking" the TP and `john` (ultimately) outranking both. The display algorithm is able to produce a normal binary-branching tree from this nothwithstanding

based on the relative ordering of the selectional features (`=T[]`, `=PDat[]`, `=D[]`) and the categorial features (`cVDitrans[promise]`, `cv[ditrans]`)

```
DERIVATION
{
  lexical_array: [
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_24>],
  active_stages: [<stage_24>],
  state: "active",
  convergent: false
}

STAGE_24
{
  lexical_array: [],
  workspace: [
    {
      {"v_ditrans", "promise"},
      {"John", {"to-inf", {"John", {"v_intrans", "win"}}}}
    },
    {"to-P", "Mary"},
  ],
  state: "react",
  convergent: false
}
```

```
ASSEMBLED TREES
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
},
{
  tag: "v_ditrans",
  syn: [
    "cVDitrans[promise]",
    "=T[to]",
    "=PDat[]",
    "cv[ditrans]",
    "=VDitrans[promise]",
    "=D[]"],
  sem: "",
  phon: "",
  constituents: ["promise", "to_inf"]
}
```





(28) The next **select** `probe` finds a match on the remaining item in the `workspace`

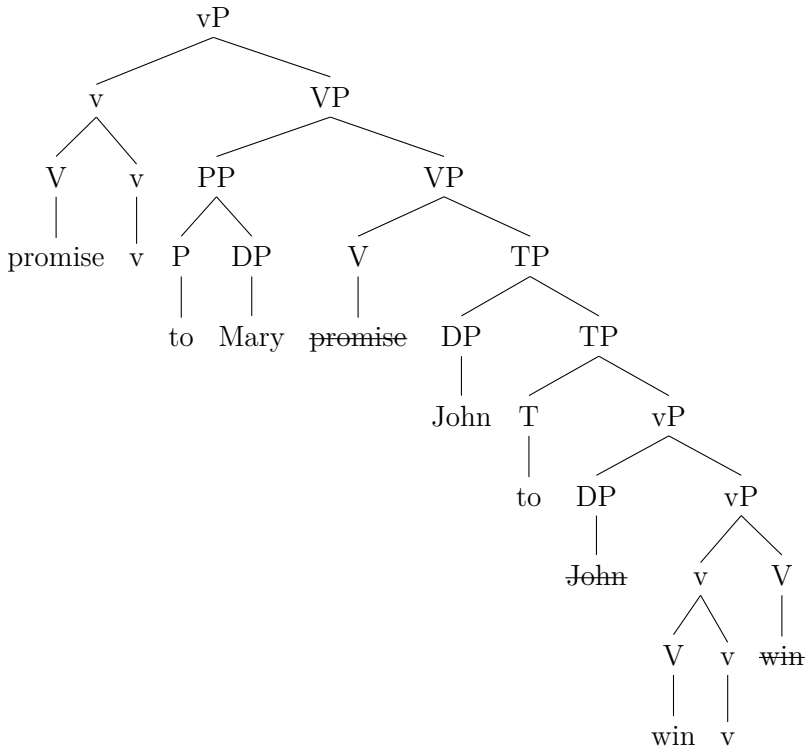
```
DERIVATION
{
  lexical_array: [
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_25>],
  active_stages: [<stage_25>],
  state: "active",
  convergent: false
}

STAGE_25
```

```
{
  lexical_array: [],
  workspace: [
    {
      {{"v_ditrans", "promise"},
      {"John", {"to-inf", {"John", {"v_intrans", "win"}}}}},
      {"to-P", "Mary"}
    }
  ],
  state: "react",
  convergent: false
}
```


ASSEMBLED TREES


```
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[mary]",
    "iCase[dat]"],
  sem: "",
  phon: "",
  constituents: ["Mary"]
},
{
  tag: "v_ditrans",
  syn: [
    "cVDitrans[promise]",
    "=T[to]",
    "=PDat[pdat]",
```

```
    "cv[ditrans]",
    "=VDitrans[promise]",
    "=D[]"],
  sem: "",
  phon: "",
  constituents: ["promise", "to_inf", "to-P"]
}
```

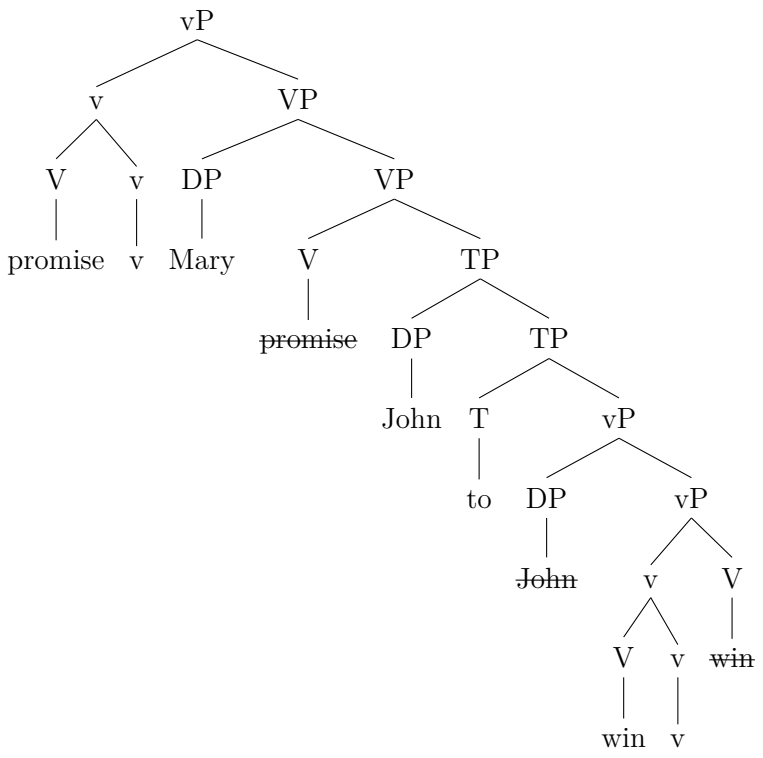


(29) (**The crucial step** - HEAD INCORPORATION) At this point, things get murky, as the original article is a little unclear. If **head incorporation** is going to happen, it pretty much has to happen now (especially given the authors' insistence that their analysis of the ungrammaticality of the passive counterpart is forced by strict adherence to cyclicity). However, if **head incorporation** happens, it would make `Mary` a blocking category for subsequent raising of `John` to subject, which is indeed the next step (because `v_ditrans` still has one unvalued **select**or feature - `=D[]`). At least, this is true if

the same consequences result from **head incorporation** as were assumed to result from it in the analysis of the passive counterpart. Recall that in that analysis, the ungrammatical reading of *Mary was promised (by John) to win* in which the *promiser* (possibly *John*) is also the winner (represented by an unpronounced *pro* in the external argument position of *v* which has raised to this position from the subject position of the lower TP - i.e. *Mary*$_{\mathrm{i}}$ *was* ~~*Mary*~~$_{\mathrm{i}}$ *pro*$_{\mathrm{j}}$ ~~*Mary*~~$_{\mathrm{i}}$ *promised* ~~*pro*~~$_{\mathrm{j}}$ *to win*) was ruled out by appeal to the notion that **head incorporation** of an unpronounced P head caused a category shift of *to Mary* from PP to DP, making it an intervener at the moment when *pro* would otherwise raise to fill the external argument position of *v*. There's no problem with *pro* filling this position, it just can't be *the same pro* that is in the lower clause - i.e. it must be `merge`d in from the `lexical array`. Obviously this logic also serves to block *John* raising in exactly the same way in the current derivation. Discussion continues below, after illustration of the **head incorporation** step.

```
DERIVATION
{
  lexical_array: [
    [{"T-past"}, {"C"}]],
  history: [
    <stage_1>,
    ...,
    <stage_26>],
  active_stages: [<stage_26>],
  state: "active",
  convergent: false
}

STAGE_26
{
```

```
  lexical_array: [],
  workspace: [
    {
      {{"v_ditrans", "promise"},
      {"John", {"to-inf", {"John", {"v_intrans", "win"}}}}},
      "Mary"
    }
  ],
  state: "react",
  convergent: false
}

ASSEMBLED TREES
{
  tag: "v_ditrans",
  syn: [
    "cVDitrans[promise]",
    "=T[to]",
    "=PDat[pdat]",
    "cv[ditrans]",
    "=VDitrans[promise]",
    "=D[]"],
  sem: "",
  phon: "",
  constituents: ["promise", "to_inf", "Mary"]
}
```



Notice this is not the same kind of **head incorporation** used earlier. The mechanics of the operation are left up to the `strategy`. In this implementation, **select** of a `cPDat[pdat]` category causes the *selectee* to restructure, essentially eliminating the PP shell[ˆstrategyimpl]. Since `stage_25` is still in a "react" state, as a result of the `merge`, it will attempt its final `=D[] probe` on its `constituents` list (as there are no items in the `lexical array`). What happens here depends greatly on implementation details.

The obvious thing to happen is for the probe to work backward through the `constituents` array (as this directly corresponds to hierarchy). In this case, the `probe` will find a match on `Mary` and stop, ultimately crashing the derivation (as there are no more D-category items in the `derivation`s remaining inventory of lexical items). This will happen again at the T stage - the T probe will find `Mary` first and stop. T will never find a match for its **select**or feature (an EPP violation) and `John` will never get case. The derivation will assemble a complete tree, but with unvalued features. The interfaces will reject the result.

There are plausible ways to save the derivation, however. One could model a system where

head incorporation does NOT happen unless v is passive. This could be done by specifying two kinds of `to-P` in the lexicon, one selected by passive v and one by the active version, and write the `strategy` in a way that only the former incorporates. A PF rule would then decide whether the P were pronounced.

Superficially, this is quite appealing, as pronounciation of *to* seems structurally dependent anyway[ˆpronounceto]. But it ultimately begs a lot of questions, foremost among them why incorporation should be dependent on the presence or absence of a *passive* structure. Absent such an explanation, it would merely be a hack to get the desired result. Moreover, nothing would actually prevent inclusion of a *pro* in the original `lexical array` alongside `John`. Consider, for example, what happens with this starting `lexical array` instead of the one shown:

```
{
  lexical_array: [
    [{"v_intrans"}, {"win"}],
    [{"pro"}],
    [{"to_inf"}],
    [{"to-P"}, {"Mary"}],
    [{"John"}],
    [{"promise"}, {"v_ditrans"}],
    [{"T-past"}, {"C"}]],
  history: [],
  active_stages: [],
  state: "start"
}
```

Running the system with this array converges without problem. *pro* sits in the subject position of the lower clause, and by **merge-over-move**, `John` is prefered over raising *pro* to

the external argument position of *v* anyway. The intervention issue never arises: it simply doesn't matter whether P incorporates or not. Since *pro* does not need case, there is no problem with it staying in the SPEC-*to*. This derives a sentence where *John promised Mary to win* means that John promised Mary that some third person - neither John nor Mary - would win, an unavailable reading. The only real solution would be to somehow prevent *pro* from appearing in the subject position of the lower clause, but that would obviate the need for this analysis in the first place (as the whole point was to account for why, in a raising analysis, *pro* could not raise to subject in the passive). More to the point, stipulating what kinds of objects can appear in SPEC-to is conceding defeat, since this more or less recreates PRO-type analyses.

FINAL STEPS

It should be obvious how the derivation proceeds from here. Only the "late-to-never" incorporation analysis will work - that is, for the active sentence to go through, incorporation has to be dependent on the presence of the passive morpheme in the way described above. The reader can verify that the feature setup on the remaining items will work. `T` and `C` merge and feature-share in the same way that `v_ditrans` and `promise` did, and `T` then attracts `John` to its SPEC, the side-effect of which is that `John` satisfies its case requirement. The `derivation` is left with an empty `lexical array` and a single, **convergent** (all features are valued), active stage. This is the **convergence** condition for `derivation`s.

APPENDIX B - PROBLEM SOLVING IN THE SYSTEM

This section starts with a straightfoward derivation to show the system functioning properly. It then gives a brief overview of some failure patterns. In the interests of space, no derivations are completely worked out. For a complete derivation, the reader is refered to Appendix A.

PARROTS LIKE NUTS

The lexicon used for this derivation is given in Appendix C.

The start state is as follows:

```
DERIVATION
{
  lexical_array: [
    [{"det"}, {"parrots"}],
    [{"det"}, {"nuts"}],
    [{"v_trans"}, {"like"}],
    [{"C"}, {"T"}]
  ],
  history: [],
  active_stages: [],
  state: "start"
}
```

From there, the system will form each of the DPs in turn. This process starts with a `stage` formed from one of the `lexical array`s.

```
STAGE
```

```
{
  lexical_array: [{"det"}, {"parrots"}],
  workspace: [],
  state: "designate",
  convergent: false
}
```

The `stage` will determine that while `det` can select `parrots` (because of its `=N[]` feature), `parrots` can't select `det`. So, it will start by pulling out `det` as the active item:

```
STAGE
{
  lexical_array: [{"parrots"}],
  workspace: [{"det"}],
  state: "probe",
  convergent: false
}
```

Which will probe and find `parrots` in the lexical array:

```
STAGE
{
  lexical_array: [],
  workspace: [{"det"}, {"parrots"}],
  state: "react",
  convergent: false
}
```

Which will trigger `merge`

```
STAGE
{
  lexical_array: [],
  workspace: [{"det", "parrots"}],
  state: "complete",
  convergent: false
}
```

At this point, there is a tree:

```
{
  tag: "det",
  syn: [
    "cD[det]",
    "=N[parrots]",
    "cCase[]"
  ],
  sem: "",
  phon: "",
  constituents: ["parrots"]
}
```

where a "tree" in this system is represented in a `syntactic object` type data structure.

This stage is complete - because there are no more actions it can take, but not convergent, because there are unvalued features (`uCase[]`). It sits in the `active stages` array until it can be reintegrated into the derivation:

```
DERIVATION
```

```
{
  lexical_array: [
    [{"det"}, {"nuts"}],
    [{"v_trans"}, {"like"}],
    [{"C"}, {"T"}]
  ],
  history: [..., <parrots_stage>],
  active_stages: [<parrots_stage>],
  state: "active"
}
```

The `derivation` has nothing to do at this point but start again with another stage. So, leaving (a pointer to) the previous `stage` in the `active stages` array, it makes a new `stage` and starts operating on that. This repeats the same process as before:

```
DERIVATION
{
  lexical_array: [
    [{"v_trans"}, {"like"}],
    [{"C"}, {"T"}]
  ],
  history: [..., <nuts_stage>],
  active_stages: [<nuts_stage>, <parrots_stage>],
  state: "active"
}
```

Next up is the v stage.

```
DERIVATION
```

```
{
  lexical_array: [
    [{"C"}, {"T"}]
  ],
  history: [..., <nuts_stage>],
  active_stages: [<nuts_stage>, <parrots_stage>],
  state: "active"
}
```

```
STAGE
{
  lexical_array: [{"v_trans"}, {"like"}],
  workspace: [],
  state: "react",
  convergent: false
}
```

Which, as the reader can verify from feature selection, will end up like this:

```
STAGE
{
  lexical_array: [],
  workspace: [{"v_trans", "like"}],
  state: "react",
  convergent: false
}
```

with `v_trans` having `merge`d with `like` - yielding the following tree:

```
{
  "tag": "v_trans",
  "syn": [
    "cv[trans]",
    "=V[like]",
    "=D[]",
    "iCase[acc]"
  ],
  "sem": "",
  "phon": "",
  constituents: ["like"]
}
```

At this point, the `like` head incorporates, effectively turning this tree into a single node. **incorporation** is a side effect of certain `select`or features. This is scriptable by changing the definition of the `strategy` that guides the derivation (see Chapter 4). For the purpose of this derivation, it is assumed to fire whenever a `=V[]` or `=T[]` feature is valued. **head incorporation** is implemented by **donate** (see Chapter 4, section on Feature Inheritance), meaning `v_trans` simply picks up all of the features on `V`. Specifically, these are *prepended* to its own feature list (to ensure that operations remain cyclic - that is, the object, required by V, will be selected before the subject, required by v, etc.)

```
{
  "tag": "v_trans",
  "syn": [
    "cV[like]",
    "=D[]",
    "cv[trans]",
    "=V[like]",
```

```
    "=D[]",
    "iCase[acc]"
  ],
  "sem": "",
  "phon": "",
  constituents: ["like"]
}
```

v_trans now has like's =D[] requirement in addition to its own. This makes the remaining two stages on the active_stages array **select**able, since they feature D heads.

```
DERIVATION
{
  lexical_array: [
    [{"C"}, {"T"}]
  ],
  history: [..., <v_stage>],
  active_stages: [<v_stage>, <nuts_stage>, <parrots_stage>],
  state: "start"
}
```

Following obvious order, the system first merges <nuts_stage> with <v_stage>, which means concatenating the stages lexical_arrays (vacuous, since they're both exhausted)

```
STAGE
{
  lexical_array: [],
  workspace: [{"v_trans", "like"}, {"det", "nuts"}],
```

```
  state: "probe",
  convergent: false
}
```

This can be **select**ed by the head, so `merge` results:

```
STAGE
{
  lexical_array: [],
  workspace: [{{"v_trans", "like"}, {"det", "nuts"}}],
  state: "probe",
  convergent: false
}
```

```
{
  "tag": "v_trans",
  "syn": [
    "cV[like]",
    "=D[det]",
    "cv[trans]",
    "=V[like]",
    "=D[]",
    "iCase[acc]"
  ],
  "sem": "",
  "phon": "",
  constituents: ["like", "det"]
}
```

As a side-effect, `det` is able to counter-`probe` and gets its `case` feature valued as a result.

```
{
  "tag": "det",
  "syn": [
    "cD[det]",
    "=N[nuts]",
    "uCase[acc]"
  ],
  "sem": "",
  "phon": ""
}
```

A critical step at this point is `compact`, which needs to eliminate the `case` feature from `v_trans` (or else it will get transmitted again in the next step, which `merge`s the subject). More details are given in later sections. Again, side-effects are user-definable - they can be written into the scripted `strategy`.

This process repeats with the output of the `parrots` `stage`, and the entire v-complex is now assembled.

```
{
  "tag": "v_trans",
  "syn": [
    "cV[like]",
    "=D[det]",
    "cv[trans]",
    "=V[like]",
    "=D[parrots]"
```

```
  ],
  "sem": "",
  "phon": "",
  constituents: ["like", "det", "det"]
}
```

In the next step, the C-T complex is formed, again involving **feature inheritance**:

```
STAGE
{
  lexical_array: [],
  workspace: [{"C", "T"}],
  state: "react",
  convergent: false
}

DERIVATION
{
  lexical_array: [
  ],
  history: [..., <v_stage>, <c_t_stage>],
  active_stages: [<c_t_stage>, <v_stage>],
  state: "active"
}

{
  "tag": "C",
  "syn": [
```

```
    "cT[pres]",
    "=v[]",
    "=D[]",
    "=uPhi[]",
    "iCase[nom]",
    "cC[decl]",
    "=T[pres]"
  ],
  "sem": "",
  "phon": "",
  "constituents": ["T"]
}
```


The two remaining active `stage`s `merge`, and `C-T` is able to select its arguments in turn - first `v_trans` itself, and second the subject (to satisfy the `=D[]` feature). Here is the state of the tree just before `probe`ing for the subject:


```
{
  "tag": "C",
  "syn": [
    "cT[pres]",
    "=v[trans]",
    "=D[]",
    "=uPhi[]",
    "iCase[nom]",
    "cC[decl]",
    "=T[pres]"
  ],
  "sem": "",
```

```
  "phon": "",
  "constituents": ["T", "v_trans"]
}
```

There are no items in the `constituents` list with the requisite features, of course, so the `probe` will recursively `probe` into each item in the `constituents` array *in reverse order* (i.e. in order of hierarchy). (Notice that this behavior is scriptable, so that systems that need to enforce in-phase equidistance, for example, can `probe` all items simultaneously.) This finds the subject (as the first of the two `det`s in the `constituents` array of `v_trans` - see above). As a side-effect, `T` can value its `uPhi[]` features, and `det` can value its `uCase[]` feature:

```
{
  "tag": "C",
  "syn": [
    "cT[pres]",
    "=v[trans]",
    "=D[parrots]",
    "=uPhi[3pl]",
    "iCase[nom]",
    "cC[decl]",
    "=T[pres]"
  ],
  "sem": "",
  "phon": "",
  "constituents": ["T", "v_trans", "det"]
}
```

The `derivation` converges, because:

1. There are no remaining items in its `lexical array`
2. There is a singular active `stage`
3. This stage is *convergent*

The `stage` is *convergent* because:

1. There are no remaining items in its `lexical array`
2. There is a single remaining item in its `workspace`
3. There are no unvalued, negative-polarity (`=` and `u`) features on its **head** or on the **head**s of any of its constituents

```
DERIVATION
{
  lexical_array: [],
  history: [..., <v_stage>, <c_t_stage>],
  active_stages: [<c_t_stage>],
  state: "convergent"
}
```

SOME FAILURE PATTERNS

To truly understand a system, it is important to know not only how it succeeds, but also how it can fail. More specifically, the purpose of a system like this one is to allow for a series of "what-if"s - so that researchers can explore the (unintended) consequences of minor changes in system architecture.

Two such potential alterations were hinted at above:

1. *What if* `probe` were simultaneous, as it is necessary for it to be for some analyses to work?

2. *What if* `compact` didn't fire after valuing an uninterpretable feature? That is, *what if* the feature were allowed to participate in more than one relationship?

Both of these require alternations in the `strategy` file.

In the first case, the required change is to `probe` all members of the target's `constituents` array at once, returning a list, and then picking the most suitable object from the list. Simultaneous `probe` of this kind mimics analyses where everything in the phase periphery is equidistant[^equid].

THe relevant stage from the derivation above involves `probe` of this tree for the subject

```
{
  "tag": "v_trans",
  "syn": [
    "cV[like]",
    "=D[det]",
    "cv[trans]",
    "=V[like]",
    "=D[parrots]"
  ],
  "sem": "",
  "phon": "",
  constituents: ["like", "det", "det"]
}
```

Rather than simply considering the first `det` (`parrots`), the system will consider both.

Is there anything to distinguish them at this point? There is: case. At this stage in the derivation, each DP looks like this, respectively:

```
{
  "tag": "det",
  "syn": [
    "cD[det]",
    "=N[nuts]",
    "uCase[acc]"
  ],
  "sem": "",
  "phon": "",
  "constituents": ["nuts"]
}

{
  "tag": "det",
  "syn": [
    "cD[det]",
    "=N[parrots]",
    "uCase[]"
  ],
  "sem": "",
  "phon": "",
  "constituents": ["parrots"]
}
```

So this is not necessarily a fatal move for this particular derivation. If the system picks randomly and chooses `nuts`, the derivation will crash in the end, because although it will reach the final stage as before, it will now *NOT* be the case that all negative-polarity features are valued within the single remaning `stage`.

However, adding the second *what if* - "what if `compact` didn't fire after valuing a feature?" -

and running the system removes this escape hatch. This is because the subject will be valued with `acc` case by `v_trans` just like the object is. The relevant step is here:

```
{
  "tag": "v_trans",
  "syn": [
    "cV[like]",
    "=D[det]",
    "cv[trans]",
    "=V[like]",
    "=D[]"
    "iCase[acc]"
  ],
  "sem": "",
  "phon": "",
  constituents: ["like", "det"]
}
```

This happens just after the first of the two pre-assembled `active stages` are `merged` with the `<v_stage>` stage. This much has been successful, of course; the `derivation` looks like this:

```
DERIVATION
{
  lexical_array: [
    [{"C"}, {"T"}]
  ],
  history: [..., <v_stage>],
  active_stages: [<v_stage>, <parrots_stage>],
```

```
  state: "active"
}
```

The `active_stages` will `merge` causing `v_trans` and `det` (parrots) to end up in the same `workspace` which will cause `v_trans` to try to `probe det(parrots)` which succeeds because `det` is a `cD[det]` and `probe` is `=D[]`. This triggers a `merge` which in turn triggers a round of mutual `probe`ing, which means that if `v_trans`' `iCase[acc]` feature has not been eliminated by `compact` that `parrots` will find a match. This is not a problem, of course, without multiple probe[ˆmultiplevalue]. But with multiple `probe` it eliminates the remaining way the system has to disambiguage the "equidistant" items. Of course, all this assumes that the implementation of muitple `probe` allows the system to look at *all* items in the `constituents` array. Coding the `strategy` carefully enough to avoid the `complement` would obviate the problem.

The point here has, of course, not to argue for or against any of these potential failure patterns. Rather, it illustrates the principle that minor tweaks in the algorithm can have derivation-wide consequences. That is to say, it illustrates the point of this project.

APPENDIX C - LEXICONS

This section contains the lexicons used to derive various examples throughout this work. They can be used to duplicate results, or as a starting point for expansion of analyses and further investigation.

VISSER'S GENERALIZATION - MOVEMENT THEORY OF CONTROL

```
{
  lexicon: [
    {
      tag: "C_decl",
      syn: [
        "cC[decl]",
        "=T[]"],
      sem: "",
      phon: ""
    },
    {
      tag: "v_intrans",
      syn: [
        "cv[intrans]",
        "=V[]",
        "=D[]],
      sem: "",
      phon: ""
    },
    {
      tag: "v_ditrans",
```

```
  syn: [
    "cv[ditrans]",
    "=VDitrans[]",
    "=D[]"]
  sem: "",
  phon: ""
},
{
  tag: "win",
  syn: [
    "cV[win]"],
  sem: "",
  phon: ""
},
{
  tag: "to-inf",
  syn: [
    "cT[to]",
    "=v[]",
    "=D[]"],
  sem: "",
  phon: ""
},
{
  tag: "T_past",
  syn: [
  ],
  sem: "",
  phon: ""
},
```

```
{
  tag: "promise",
  syn: [
    "cVDitrans[promise]",
    "=T[]",
    "=PDat[]"],
  sem: "",
  phon: ""
},
{
  tag: "to-P",
  syn: [
    "cPDat[pdat]",
    "=D[]",
    "iCase[dat]"],
  sem: "",
  phon: ""
},
{
  tag: "Mary",
  syn: [
    "cD[mary]",
    "uCase[]"],
  sem: "",
  phon: ""
},
{
  tag: "John",
  syn: [
    "cD[john]",
```

```
        "uCase[]"],
      sem: "",
      phon: ""
    },
    {
      tag: "pro-subj",
      syn: [
        "cD[john]",
        "uCase[]"],
      sem: "",
      phon: ""
    }
  ]
}
```


PARROTS LIKE NUTS


```
{
  "lexicon": [
    {
      "tag": "nuts",
      "syn": [
        "cN[nuts]",
        "iPhi[3pl]"
      ],
      "sem": "",
      "phon": ""
    },
    {
```

```
  "tag": "parrots",
  "syn": [
    "cN[parrots]",
    "iPhi[3pl]"
  ],
  "sem": "",
  "phon": ""
},
{
  "tag": "like",
  "syn": [
    "cV[like]",
    "=D[]"
  ],
  "sem": "",
  "phon": ""
},
{
  "tag": "v_trans",
  "syn": [
    "cv[trans]",
    "=V[]",
    "=D[]",
    "iCase[acc]"
  ],
  "sem": "",
  "phon": ""
},
{
  "tag": "T",
```

```
    "syn": [
      "cT[pres]",
      "=v[]",
      "=D[]",
      "=uPhi[]",
     "iCase[nom]"
    ],
    "sem": "",
    "phon": ""
  },
  {
    "tag": "C",
    "syn": [
      "cC[decl]",
      "=T[]"
    ],
    "sem": "",
    "phon": ""
  },
  {
    "tag": "det",
    "syn": [
      "cD[det]",
      "=N[]",
      "uCase[]"
    ],
    "sem": "",
    "phon": ""
  }
]
```

```
}
```

CURRICULUM VITAE

Education

December 2016	PhD in Linguistics, **Indiana University**, Bloomington
May 1997	BA in Philosophy, English Literature **Appalachian State University**, Boone, NC

Work

Jan 2013-present	Co-Founder and CTO at **Periodic**, Bloomington, IN
Sep 2011-Jan 2013	Lead Developer at **ScheduleThing LLC**, Bloomington, IN
Jan 2010-May 2010	Instructor for L310: Syntax **Indiana University**, Bloomington, IN
Jan 2007-May 2009	Research Assistant **Indiana University**, Bloomington, IN
Jan 2004-May 2005	Research Assistant **Indiana University**, Bloomington, IN

Publications

Markus Dickinson and Joshua Herring (2008). Developing Online ICALL Exercises for Russian. *The 3rd Workshop on Innovative Use of NLP for Building Educational Applications (ACL08-NLP-Education).* Columbus, OH.

Markus Dickinson and Joshua Herring (2008). Russian Morphological Processing for ICALL. *The Fifth Midwest Computational Linguistics Colloquium (MCLC-5).* East Lansing, MI.

Damir Cavar, Joshua Herring, Toshikazu Ikuta, Paul Rodrigues, Giancarlo Schrementi (2006) On Unsupervised Grammar Induction from Untagged Corpora. In: *P. Kaszubski (ed.) PSiCL: Poznań Studies in Contemporary Linguistics.* 41, Adam Mickiewicz University, Poznań, Poland. pp. 57-71. ISBN: 73-7208-165-4

Damir Cavar, Joshua Herring, Toshikazu Ikuta, Paul Rodrigues, Giancarlo Schrementi (2004) On Statistical Bootstrapping. In: William G. Sakas (ed.) *Proceedings of the First Workshop on Psycho-computational Models of Human Language Acquisition, held in cooperation with COLING 2004*, Geneva, pp. 9-16. ACL Anthology

Damir Cavar, Joshua Herring, Toshikazu Ikuta, Paul Rodrigues, Giancarlo Schrementi (2004) Alignment Based Induction of Morphology Grammar and its Role for Bootstrapping. In: Gerhard Jäger, Paola Monachesi, Gerald Penn and Shuly Wintner (eds.) *Proceedings of Formal Grammar 2004*, Nancy, pp. 47-62.