1 Introduction and Data

It has been long recognized that certain pronominal forms do not appear in positions in English where non-pronominal noun phrases are perfectly acceptable. In [Zwi86] this is referred to as the “unaccented pronoun constraint”, and some of that data is repeated here:

(1) We took in the unhappy little mutt right away.
(2) * We took in him right away.
(3) Martha told Noel the plot of Gravity’s Rainbow.
(4) * Martha told Noel it.

This data is somewhat incomplete. When we consider the sentences above, as well as each possibility for the pronominalization of the accusative arguments, the pattern displayed below is exhibited. I have chosen here to render the pronouns in their reduced forms, quasi-phonetically, as affixes\(^\text{2}\) on the verbs or prepositions that select them. The reader should read -it and -im as [it] and [im] respectively, and without stress:

(5) We took in the unhappy little mutt right away.
(6) * We took in-im right away.
(7) We took the unhappy little mutt in right away.
(8) We took-im in right away.
(9) Martha told Noel the plot of Gravity’s Rainbow.
(10) Martha told-im the plot of Gravity’s Rainbow.
(11) * Martha told Noel-it.
(12) ? Martha told-im-it.\(^\text{3}\)
(13) Martha told the plot of Gravity’s Rainbow to Noel.
(14) Martha told-it to Noel.
(15) Martha told the plot of Gravity’s Rainbow to-im.
(16) Martha told-it to-im.

\(^1\)I am grateful for the judgments, examples, and feedback provided by my mentors, colleagues, and friends, in particular Carl Pollard, Bob Levine, Dave Odden, Cynthia Clopper, Julie McGory, Scott Martin, Vedrana Mihalicek, Dahee Kim, Lia V.D. Mansfield, and Kevin Gabbard, as well as the students of the Spring 2010 section of Linguistics 502. The blame for any factual errors, formal errors, or general shortsightedness unquestionably lies with me.

\(^2\)I wish to use ‘affix’ in a neutral way, leaving open the possibility that these words are perhaps better described as clitics, although I wish to follow [Zwi95] in the observation that the criteria for distinguishing one from the other are unclear. This work is part of an effort to make precise exactly what distinctions are necessary.

\(^3\)The grammaticality of this example is plausibly dialectally restricted.
If pronouns are noun phrases, syntactically speaking, why can they not combine in all the same ways as non-pronominal NPs? It is argued here that accepting a grammatical framework which separates the surface representations of syntactic combination (phenogrammar, or concrete syntax) from their structural counterparts (tectogrammar, or abstract syntax) helps to provide a straightforward way to account for the data above.

In the next section, an overview is given of the notions of phenogrammar and tectogrammar, as well as a sketch of the proposed analysis. Section 3 is an introduction to the formal systems we make use of in the paper. In section 4, the salient features of the grammatical framework we will use are laid out. Section 5 contains the analysis of the data above, including lexical entries and derivations for several of the example sentences. In section 6, some remaining issues for future research are outlined.

2 Overview

2.1 Phenogrammar and Tectogrammar

The relationship between syntactic dependency and surface word order has been recognized to be a thorny problem for approximately as long as people have been studying the syntax of natural languages. This distinction is at least implicitly recognized in most modern theories of grammar. Phenogrammar constitutes, minimally, a set of output strings specifying the possible word orders for a given syntactic structure. In mainstream generative grammar theories of syntax, “Move” rules describe operations on trees which reorder constituents in surface form. In Head-driven Phrase Structure Grammar (HPSG), the DOM feature specifies possible constraints on the linear order of complex signs. In the tradition of logical grammar, this idea is represented by Haskell Curry’s distinction between phenogrammar and tectogrammar [Cur61], nicely summarized by Muskens in the following manner: “The latter is language as it appears or manifests itself; the former language as it is built, its underlying structure” [Mus09]. Curry himself draws an analogy between between tectogrammar and morphology, and between phenogrammar and morphophonemics [Cur61]. For the most part, this distinction has mostly been observed in a manner whereby tectogrammar is related to the mechanics of semantic combination, and where phenogrammar is taken to be nothing other than string manipulation. Our view is that this type of phenogrammar is adequate for describing linear word order, but that it can be extended to cover a much broader range of phenomena.

This “Curryesque” view of grammar is a currently active area of research at OSU. A similar view of grammar is shared by researchers in France and Japan (de Groote’s “Abstract Categorial Grammar” [dG01]), in Sweden (Ranta’s “Grammatical Framework” [Ran04]), and in the Netherlands (Muskens’ “Lambda Grammar” [Mus09]). These systems are all characterized by the use of lambda calculi and higher-order logical systems not only for the more familiar representation of truth-conditional semantics, but also for phenogrammatical representations as well. While the use of lambda calculi for representing meaning has been well-established since Montague in the late 1960s, it has been put to other linguistic purposes at least as far back as [Cre73]. Its use as a way to describe phenogrammar is somewhat newer, and originates (as far as I know) with Richard Oehrle in the 1990s [Oeh94].

Such a perspective becomes vital when beginning to consider phenomena like scrambling and agglutination. The question posed is simple: do we wish to abandon the notion of syntactic
constituency simply because the surface realizations of certain strings may permute in predictable ways? We wish to side with those who argue that linear word order lies somewhat closer to the domain of phonology, insofar as that discipline can be roughly characterized as the study of how things sound. I do not wish to imply that “accounting for word order” is entirely the job of phonologists, nor do I wish to imply in any way that it is even a primary area of research. This perspective necessitates an approach to syntax which takes phonology seriously, and treats phonological and prosodic constituency as separate from syntactic combinatorics. We consider the string-based representations of words (notated in this work by a sans-serif font, e.g. a word) to be akin to phonological words, and their combinations to be the part of syntax most closely related to ‘what things sound like’. Perhaps phenogrammatical representations can be thought of as input to a system which calculates surface representation based on such strings, although this is a somewhat less neutral theoretical proposal, and will not be discussed further here.

From this less syntactocentric perspective, the proper role of syntax as tectogrammar is only to specify the combinatoric potential of expressions. It mediates between the potentially audible contribution an expression makes to utterances containing it, i.e. the input to the phonological component of the grammar, and the semantic contribution it makes to the meaning of utterances containing it. Phenogrammar has heretofore been taken to represent word order, but it also encompasses more subtle and varied aspects of prosody including but not limited to intonation and stress. By taking such a perspective, it is possible to give a concise account of the data in question.

2.2 Analytical Strategy

In order to address the question of why reduced pronouns are acceptable in certain positions, and unacceptable in others, we note first that in every grammatical case, the pronouns appear immediately to the right of the words of which they are presumably syntactic arguments. When there is material intervening between the selecting expression and the pronoun, as is potentially the case with ditransitive verbs and verb-particle constructions such as (6) and (11), we find that reduced pronouns are unacceptable. Interestingly, ordinary (unreduced) pronouns appear to be acceptable in the positions in question as long as they are given prosodic prominence, e.g. a contrastive pitch accent. Since they are evidently incapable of bearing a pitch accent on their own, they are deemed to be what [Zwi95] refers to as ‘phonologically dependent’ on other material. Of course, in most standard constructions, it is possible to have both unreduced and unaccented pronouns.

We assume that verbs and prepositions which select accusative noun phrases as arguments generally specify the positions these arguments appear in, relative to the selector itself. In our grammar, these will be represented by ‘linearization functions’, lambda terms specifying word order. As it is our intention to create some room in our grammar formalism to address morphological concerns, we regard the morphological (or at least, affixal) structure of the selector itself and its

---

4 There is recent debate as to whether syntactic constituency or prosodic constituency is the proper domain on which to base semantic representation. See [CS08] and [Yat07] for HPSG-based articulations of each position.
5 Although Ranta’s GF considers morphological inflection to be a part of phenogrammar, which he calls “concrete syntax” [Ran04].
6 Cf. the sentence customarily rendered We took in HIM right away, where “him” stands as opposed to, say, “her”. We note that it is the intervention of the particle in which we believe makes this construction ungrammatical in the unreduced, unaccented case.
7 As is the case with most categorial grammars, as well as HPSG, we do not consider case to be a structural property which must be checked, but instead a feature of words and phrases themselves.
linearization function as separate entities, and our grammar reflects this distinction. This provides room to treat the reduced pronouns as something slightly other than words; instead, we regard them as pronominal affixes, which combine with the word selecting them, and yield a new word, rather than a phrasal expression. There is at least a three-way distinction to be made here: between pronominal affixes, ‘standard’ derivational and inflectional affixes, and phrasal affixes like possessive -’s, which is discussed in section 6: “Future Research”. Additionally, pronominal affixes are different from other ‘standard’ affixes, in that we take them to attach to words, rather than stems, although this claim is under investigation. Once more than one word has been combined, the pronominal affixes are unable to attach\(^8\), and thus the grammaticality pattern observed in the data is accounted for. This particular strategy will allow pronominal affixes to stack, if desired. If such constructions are unacceptable for a group of speakers, an alternate analysis is provided that rules out the examples of questionable grammaticality.

3 Technical Overview

This type of grammar is proof-theoretic, roughly meaning that we conceive of linguistic categories as logical formulas, and expressions themselves as proofs of those formulas. We employ here a proof system known as Gentzen sequent-style Natural Deduction, or Gentzen ND for short. This method of doing things has a few properties which we believe make it an appealing way to discuss linguistic theory in more formally precise terms. First of all, Gentzen ND proofs have tree-based representations which should not look wholly unfamiliar to those more versed in the tradition of mainstream generative grammar. Secondly, the specific method of implementing these systems, Typed Lambda Calculus (TLC), is already well-known to semanticists, and those familiar with Montague Grammar. Thirdly, these systems are already familiar to many logicians, mathematicians, and computer scientists, making it easier to create dialogs between linguistics and those fields.

Gentzen ND proof systems consist, in their most basic formulation (to be complexified later), of statements which exemplify a “provability” relation between contexts and formulas, as well as various rules for the manipulation of these statements. The statements themselves are called sequents or judgments, and the rules are called inference rules. A sequent, for our purposes, takes the following form:

\[ \Gamma \vdash A \]

which is read as “\(\Gamma\) derives \(A\)”, “\(\Gamma\) is a proof of \(A\)”, or perhaps “\(A\) follows from \(\Gamma\)”\(^9\). The provability relation symbol is typically referred to as the turnstile. To its left, we keep track of assumptions, and to its right, we list conclusions. Here, \(\Gamma\) is a metavariable over lists of formulas called a context, so it may consist of a single formula, more than one formula, or possibly no formulas at all. We take \(A\) to be a metavariable over formulas, so \(A\) will always represent a single formula. Conceptually, sequents represent the fact that from some assumptions \(\Gamma\), it is possible to prove \(A\). The actual mechanics of these proofs are performed using various axioms and inference rules, which are read top-down.

Inference rules consist of structural rules, logical rules, and non-logical rules. Structural rules are rules which manipulate contexts, logical rules are concerned with the behavior of whatever logical connectives exist in the system, and non-logical rules are ones which are stipulated, and

\(^8\) unlike phrasal affixes like possessive -’s

\(^9\)
come from no inherent property of the deduction system itself. Another difference between logical and non-logical rules is that while non-logical rules make reference to specific formulas or types, logical rules are schematized, i.e. they hold for any formulas of the system.

We will eventually use two different types of logic here: a variant of positive intuitionistic propositional logic (PIPL), and the implicational fragment of linear logic (ImpLL). The exact nature of the inference rules in an ND system is determined by the logic you wish them to describe. Since the logic out of which we will get the most immediate mileage is PIPL, we now examine its inference rules. While intuitionistic logic is a weaker, less expressive logic than classical propositional logic, these should be familiar to those with some familiarity with classical logic, as PIPL is very much like classical logic without disjunction, negation and the law of the excluded middle.

### 3.1 Positive Intuitionistic Propositional Logic (PIPL)

#### 3.1.1 Hypothesis

It is safe to say that, in most proof systems, there are certain formulas which are taken to be axiomatic, i.e. provable from nothing at all, and these are given in sequent form as well. However, while they are subject to the inference rules of a given system, they are not rules themselves. In order to initiate a proof, we need the logical axiom of hypothesis, which simply states that at any point, if you assume $A$, then you can prove $A$, and it takes the following form:

$$\Gamma \vdash A$$  \hspace{1cm} (Hyp)

Simply put, “$A$ derives $A$”. Of course, since $A$ is a metavariable over formulas, we actually mean that any formula whatsoever is a proof of itself, so we are free to instantiate $A$ in whatever manner we wish.

#### 3.1.2 Structural Rules

The three main structural rules are the rules of weakening (W), contraction (C), and permutation (P), and they take the following form:

$$\frac{}{\Gamma, A \vdash B} \hspace{1cm} (C) \hspace{1cm} \frac{\Gamma \vdash B}{\Gamma, A \vdash B} \hspace{1cm} (W) \hspace{1cm} \frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C} \hspace{1cm} (P)$$

The rule of contraction says explicitly “If it is possible to prove $B$ from some list of formulas $\Gamma$ ending in $A$ and $A$, then it is possible to prove $B$ from the same list of formulas $\Gamma$ ending only in $A$.” Effectively, this says that if you assumed the same thing twice, then you only need to have assumed it once. The rule of weakening says “If $B$ follows from some formulas $\Gamma$, then $B$ also follows from $\Gamma$ and $A$.” Essentially, this says that if assuming $\Gamma$ is sufficient to let you prove $B$, then you can certainly prove $B$ from $\Gamma$ and some other hypotheses as well. The rule of permutation encodes the fact that the order of the formulas in the context is immaterial, and they can thus be freely reordered. For the sake of frugality, multiple uses of this rule are customarily collapsed into one invocation.

---

\(^9\)in particular, the law of the excluded middle does not hold

\(^10\)As will be illustrated later, other logics exist in which one or more of these rules are absent, in particular, ImpLL.

\(^11\)The necessity of these rules results from our choice to treat the context as a list of formulas. If we had chosen a different structure for the context, the need for one or more of these rules could be eliminated.
3.1.3 Logical Rules

In ND systems, logical inference rules exhibit symmetry in that they generally either introduce or eliminate logical connectives. Since we are working in a system with two connectives, implication (\( \rightarrow \)), and conjunction (\( \wedge \)), we will only have introduction and elimination rules for those connectives, although there are two conjunction elimination rules. First, we examine the rules for implication:

\[
\begin{align*}
\Gamma \vdash A \rightarrow B & \quad \Delta \vdash A \quad (\rightarrow E) \\
\Gamma, \Delta \vdash B & \\
\Gamma \vdash A \rightarrow B & \quad (\rightarrow I)
\end{align*}
\]

The rule of implication elimination (\( \rightarrow E \)), sometimes called modus ponens, says that if you can prove, from some list of formulas \( \Gamma \), that \( A \) implies \( B \), and you can prove \( A \) from some other list of formulas, then if you combine those lists of assumptions, you can prove \( B \). Implication introduction (\( \rightarrow I \)) says that if, from some list of formulas \( \Gamma \) ending in (including) \( A \), you can prove \( B \), then without assuming \( A \), you can prove \( A \rightarrow B \). This is sometimes called hypothetical proof.

Now, moving on to conjunction we see that there are two elimination rules. This is because when you eliminate the conjunction connective, you may choose either one of the conjoined formulas as your conclusion. The rules for conjunction are given below:

\[
\begin{align*}
\Gamma \vdash A \wedge B & \quad (\wedge E_1) \\
\Gamma \vdash A & \\
\Gamma \vdash B & \\
\Gamma, \Delta \vdash A \wedge B & \quad (\wedge E_2) \\
\Gamma \vdash A & \quad \Delta \vdash B & \quad (\wedge I)
\end{align*}
\]

The elimination rules effectively say that if you can prove a conjunction of two formulas, then you are capable of proving either formula on its own. As mentioned previously, the elimination rules come in two varieties, \( (\wedge E_1) \) and \( (\wedge E_2) \), which allow you two choose the left and the right conjuncts, respectively. The introduction rule \( (\wedge I) \) simply says the opposite: if you can prove two formulas separately, then you can prove their conjunction by assuming everything that you assumed to prove them each individually.

The following is a sample proof using only the structural rules and the rules for implication, as those are the ones that we will be making the most use of in the future.

\[
\begin{align*}
\text{Hyp} & \quad \text{Hyp} \\
\rightarrow E & \quad \rightarrow E \\
A \rightarrow (A \rightarrow B) & \vdash A \rightarrow (A \rightarrow B) \\
A, A \vdash A \rightarrow B & \\
A \rightarrow (A \rightarrow B), A \vdash A \rightarrow B & \\
A, A \vdash B & \quad C \\
A \rightarrow (A \rightarrow B), A \vdash B & \quad \rightarrow I \\
A \rightarrow (A \rightarrow B) & \vdash A \rightarrow B
\end{align*}
\]

3.2 The Implicative Fragment of Linear Logic (ImpLL)

Linear logic has only one structural rule: the rule of permutation. Since it ignores weakening and contraction, it has been called “resource-sensitive”, in that while you may freely re-order your hypotheses, you may not simply add new ones, or combine like hypotheses. The rule of hypothesis exists unchanged from PIPL, and will not be repeated here, as does the rule of permutation.

\(^{12}\) Since we do not make particular use of the 0-ary connective \( \top \) here, it is left intentionally neglected.

\(^{13}\) An analogy to the money in one’s wallet is frequently drawn. If you have four dollars, you may use them as you wish, but once they’ve been used, they’re gone. You may not hypothesize more dollars in any useful way. Similarly, you may not combine two one-dollar bills into a new one-dollar bill.
The rules for linear implication look exactly like the rules for intuitionistic implication, with the difference being that the connective in question is the linear implication connective (→), sometimes called lollipop, rather than the intuitionistic implication connective (→). Other than that, their formulations are identical:

\[
\begin{align*}
\Gamma & \vdash A \rightarrow B & \Delta & \vdash A (\rightarrow E) \\
\Gamma, \Delta & \vdash B & \Gamma & \vdash A \rightarrow B (\rightarrow I)
\end{align*}
\]

We now give a sample proof in ImpLL, which is of course provable in PIPL, replacing → by to. Note, however, that the theorem proved above in PIPL is not provable in ImpLL, since it makes use of the rule of contraction, which is unavailable in ImpLL.

\[
\begin{array}{c}
\text{Hyp} \\
A \rightarrow (B \rightarrow C) \vdash A \rightarrow (B \rightarrow C) \\
\ \vdash A \\
\end{array}, \ 
\begin{array}{c}
\text{Hyp} \\
A \rightarrow (B \rightarrow C), A \vdash B \rightarrow C \\
B \vdash B \\
\end{array}
\]

By notational convention, → associates to the right, so we typically drop the parentheses where no ambiguity results. The sequent derived in the proof above could thus be written as the following:

\[
A \rightarrow B \rightarrow C \vdash B \rightarrow A \rightarrow C
\]

### 3.3 Linguistic Connections

The study of linguistics in general takes as one of its fundamental observations that there is regularity to language. Languages exhibit patterns, and it is possible to categorize various aspects of language based on the appropriate descriptions of those patterns. Perhaps the most immediate example, though certainly not the only one, is that of syntactic categories. If we regard logical formulas as being names for these categories, then thinking about logic as a way to describe provable relationships between categories and their combinations gives us a way to reason precisely about the syntactic categories themselves.

At this point a question presents itself: if we can represent syntactic categories as formulas in a logic, then what status do actual linguistic expressions have? Insofar as they can be regarded as members of those categories, then they are in some way proofs of those categories. That is to say, if we require a proof that the category of ‘sentence’ is well-founded, we need only look as far as it takes to find one. The actual sentence itself is a proof that the category ‘sentence’ is inhabited, and likewise for noun phrases, etc. It is for this reason that proof theory provides a highly appropriate approach to grammar. If we conceive of words as being proofs of particular syntactic categories that tell you what categories will result when you combine them, then (assuming we’ve set things up right) their combinations will be proofs of the categories that result. There is a parallelism here, between linguistic expressions and their syntactic categories on the one hand, and between proofs and formulas on the other. This notion is reified by thinking of our logics type-theoretically, and

---

14 Consider also the individuals and propositions of Montague semantics, or what we may loosely deem phenogrammatical objects, to be explored in due time.
expanding them somewhat to include actual proof terms (stand-ins for linguistic expressions) of
the formulas we are reasoning about. These terms take the form of statements in the typed lambda
calculus. Each manipulation of the terms is reflected in the logic of formulas underlying the type
system, and it is this relationship between term / type pairs and proof / formula pairs to which
the so-called Curry-Howard correspondence has come to refer.

3.4 Typed Lambda Calculus
We now turn to a concrete example of a proof calculus\footnote{This is partially incomplete. For purposes of simplicity, we omit discussion of the unit type, as well as neglecting \(\alpha\)-conversion and \(\eta\)-reduction rules.} for one of the logics described above, typed lambda calculus (TLC). Our sequents take on a form which encodes a context, a term, and a type:

\[ \Gamma \vdash a : A, \quad \text{that is, (context)} \vdash \text{term} : \text{TYPE} \]

As we move from a simple logic of formulas to one which has terms annotating those formulas,
we must make a few other modifications. First, the contexts (originally lists of formulas) are now
lists of term-type pairs of the form \(x : A\). Practically, this amounts to decorating the formulas in
the context with the proof terms with which they are associated. Secondly, we need to retool some
of our basic rules. The rule of hypothesis now takes the following form:

\[ \frac{x : A \vdash x : A}{x : A \vdash x : A} \quad \text{(Hyp)} \]

Here, we can think of this rule as saying that if we assume that we have a proof, called \(x\), of
some formula \(A\), then it follows that we have a proof, called \(x\), of \(A\).” More simply, if we assume
that \(x\) is of type \(A\), then it follows that \(x\) has type \(A\).

Additionally, we add another rule to introduce non-logical axioms (which we will make extensive
use of later, as these are the forms that our lexical entries will take):

\[ \vdash a : A \quad \text{(Ax)} \]

This simply corresponds to an assertion that some constant \(a\) has type \(A\). Customarily we omit
the line invoking this rule as an instance of \textbf{Ax}, since it is typically clear from context what is
going on. We are now capable of annotating our structural rules with terms as well. The piece of
symbology \(b[x]\) means “\(b\), possibly containing one or more free occurrences of \(x\)”, and \(b[x, y]\) likewise
meaning “\(b\), which may contain one or more free occurrences of \(x\) and \(y\)”. Similarly, \(b[a/y]\) is taken
to mean “\(b\), with \(a\) replacing every free occurrence of \(y\)”, where \(a\) is any term which is substitutable
for \(y\) in \(b\). By “substitutable”, we mean that no variable within \(a\) becomes accidentally bound as
a result of the substitution:

\[
\begin{align*}
\Gamma, x : A, y : A &\vdash b[x, y] : B \\
\Gamma, x : A &\vdash b[x/y]B \\
\Gamma &\vdash b : B \\
\Gamma, x : A &\vdash b : B \\
\Gamma, y : B, x : A, \Delta &\vdash c[x, y] : C
\end{align*}
\]
The rule of contraction is similar, noting that if we have two hypotheses of the same type, we may contract them to one hypothesis of that type, substituting variable names as we go. The rule of weakening, as before, says that we lose nothing by adding a hypothesis to the context. Similarly, the rule of permutation allows for the reordering of hypotheses within the context. At this point we can turn to the term constructors and the inference rules with which they are associated, considering first the rules for implication:

\[
\frac{\Gamma \vdash f : A \to B \quad \Delta \vdash a : A}{\Gamma, \Delta \vdash f(a) : B} \quad (\to E)
\]

\[
\frac{\Gamma, x : A \vdash b[x] : B}{\Gamma \vdash \lambda x.b[x] : A \to B} \quad (\to I)
\]

The implication elimination rule from before is associated with the application term constructor, which is written here as \(f(a)\). It may be helpful to think of this as function application, which it is, in the set-theoretic interpretation of intuitionistic logic. There is a term equivalence for lambda terms\(^{16}\) realized as the rule called \(\beta\)-reduction:

\[
\lambda x.b[x](a) \equiv b[a/x]
\]

For those who are unfamiliar with the lambda calculus, it may be helpful to think of this rule in the following way: the \(\beta\)-reduction rule allows for the substitution of arguments within functional terms, so that wherever a free occurrence of \(x\) is found under the scope of a \(\lambda\), it may be replaced by \(a\). Often we will use the symbol \(\triangleright \) to denote “reduces to after a sequence of \(\beta\)-reductions”. We will see more examples of this in the next section.

The rule for implication introduction allows us to withdraw a hypothesis from the context which appears free in the term \(b\), if it appears at all\(^{17}\), and lambda-bind it.

The rules for conjunction operate exactly the same as before, with the addition of proof terms allowing us to explicitly track when formulas have been ‘put together’ or ‘taken apart’.

\[
\frac{\Gamma \vdash c : A \land B}{\Gamma \vdash \pi(c) : A} \quad (\land E_1)
\]

\[
\frac{\Gamma \vdash c : A \land B}{\Gamma \vdash \pi'(c) : B} \quad (\land E_2)
\]

\[
\frac{\Gamma \vdash a : A \quad \Delta \vdash b : B}{\Gamma, \Delta \vdash (a, b) : A \land B} \quad (\land I)
\]

Examining the introduction rule first, we can see it involves the pairing term constructor \((,\)\). That is, if we have terms (proofs) of two types (formulas), then the pair of those terms will be a proof of the conjunction of those formulas. As before, there are two elimination rules, and each comes associated with a projection term constructor: \(\pi\) for the left (or first) projection, and \(\pi'\) for the right (or second) projection. These term constructors allow us to retrieve the term in the relevant ‘side’ of the pair, corresponding to whichever formula of the conjunction we wish to give a proof of.

Now we can give the same proof as before, although noting that in this case we have explicit proof terms. Notice that if we examine only the types, the proof is identical to the one appearing in the earlier subsection:

\(^{16}\)Much more formal detail is required to define the equivalences of lambda terms, but this is omitted here for the sake of simplicity. The interested reader is referred to [LS86].

\(^{17}\)That is, vacuous binding is allowable, although it will ultimately be prevented by the linearity of the type logic for the tectogrammar.
Hyp \[ f : A \rightarrow (A \rightarrow B) \vdash f : A \rightarrow (A \rightarrow B) \]

\[ \frac{x : A \vdash x : A}{f : A \rightarrow (A \rightarrow B), x : A \vdash f(x) : A \rightarrow B} \]

Hyp \[ y : A \vdash y : A \]

\[ \frac{f : A \rightarrow (A \rightarrow B), x : A, y : A \vdash (f(x))(y) : B}{f : A \rightarrow (A \rightarrow B), x : A \vdash (f(x))(x) : B} \]

\[ \frac{f : A \rightarrow (A \rightarrow B), x : A \vdash (f(x))(x) : B}{f : A \rightarrow (A \rightarrow B) \vdash \lambda x.(f(x))(x) : A \rightarrow B} \]

For those familiar with formal semantics in general, and Montague semantics in particular, this system should be familiar. What may be unfamiliar are the uses to which we wish to put it. While TLC is frequently thought of in linguistics as a system for talking about the meanings of linguistic expressions, that is simply one customary application. In fact, TLC can be used to represent any sort of linguistic data which can be profitably given a proof-theoretic description. Here we wish to make particular use of it to discuss phenogrammar, in particular word order, with some additional refinements to come later.

### 3.5 Phenogrammar

The simplest logical phenogrammar is one wherein terms are simply concatenated to form ever longer strings of words and phrases. The standard assumption of many categorial grammars has been that the (tecto-)type constructors should indicate the linear word order. Since we have a separate phenogrammar, there is no reason to build the linear word order into the type system. Instead, we allow the word order to be lexically specified. If we adopt a system similar to the one outlined in [Oeh94], we define our type inventory as follows:

- **St** is a type.
- If \( A \) and \( B \) are types, then \( A \rightarrow B \) is a type.

The type constructor \( \rightarrow \) is ordinary intuitionistic (as opposed to linear) implication, and its elimination and introduction rules have the term constructor counterparts application and lambda-abstraction, respectively.

\[
\frac{\Gamma \vdash f : S \rightarrow T \quad \Delta \vdash s : S}{\Gamma, \Delta \vdash f(s) : T} \quad \text{App} \quad \frac{\Gamma, x : S \vdash t : T}{\Gamma \vdash \lambda x.t : S \rightarrow T} \quad \text{Abs}
\]

Our terms will typically contain string constants representing words, and sometimes variables which have been abstracted over. We also make use of the string concatenation operation \( \circ \), which has type \( \text{St} \rightarrow \text{St} \rightarrow \text{St} \), axiomatized as the binary associative operation of a free monoid. Additionally, we stipulate the existence of the null string \( \epsilon \) as the two-sided identity for \( \circ \). In such a system, the phenogrammar of a transitive verb, e.g. *kissed*, would be represented by the following lexical entry:

\[ \vdash \lambda y \lambda x.x \circ \text{kissed} \circ y : \text{St} \rightarrow \text{St} \rightarrow \text{St} \]

So the term for phenogrammar of *kissed* is free to combine with another string via application, which the term concatenates to the right of the constant representing the actual word *kissed*. The resulting expression is then free to combine with another string, which it concatenates to the left. The following lexical entries for *Hildegard* and *Elvis* are exactly such strings:
⊢ hildegard : St
⊢ elvis : St

We can apply the function of kissed to Elvis, and then to Hildegard, remembering that ⊢ means ‘reduces to’, and obtain the following result\(^\text{18}\), which asserts that Hildegard kissed Elvis is itself a string:

\[\vdash \lambda y. \lambda x. x \circ \text{kissed} \circ y (\text{elvis})(\text{hildegard})\]
\[\vdash \lambda x. x \circ \text{kissed} \circ \text{elvis} (\text{hildegard})\]
\[\vdash \text{hildegard} \circ \text{kissed} \circ \text{elvis} : \text{St}\]

For the sake of brevity, when a number of lambda-bound variables of the same type occur in sequence, we will write them under the scope of one lambda. Using this convention, our lexical entry for the phenogrammar of kissed would be written as

\[\vdash \lambda y. x \circ \text{kissed} \circ y : \text{St} \rightarrow \text{St} \rightarrow \text{St}\]

### 3.6 Tectogrammar

The basis for our notion of tectogrammar is the **multiplicative kernel** of linear logic (referred to here as MKLL). While we advocate the addition of the dependent product type constructor for various reasons (an enhancement in common with the proposal in \([\text{dGM07}]\)), it will not be discussed here, and nothing in our analysis currently hinges on its use. Linear logic is well-suited for application to linguistic theory, as it is resource sensitive. Unlike classical logics, where formulas have a somewhat more abstract status, linear logic formulas are often spoken of in terms of ‘storage’, ‘production’, and ‘consumption’, and are suitable to model the idea of linguistic expressions as finite resources. We do not, for example, wish to consider two occurrences of the same noun phrase to potentially be the same as one occurrence of that same noun phrase.

The primary aspect of MKLL that is of tectogrammatical importance is the linear implication type constructor, \(\to\). While this connective has much in common with the intuitionistic implication connective \(\rightarrow\), it is not subject to the structural rules of weakening or contraction, causing it to behave in a slightly different manner. We think of a formula \(A \to B\) as something which potentially consumes an \(A\) to produce a \(B\). A traditional categorial analysis of verb phrases / intransitive verbs would be the type \(\text{NP} \to \text{S}\), i.e. a function which, when provided with a noun phrase, produces a sentence. Note that there is no inherent concept of ‘word order’, only an order of combination; for us, word order is taken care of in the phenogrammar. In order to combine expressions in the tectogrammar, we make use of the rule of \(\to\) elimination, which is laid out in the beginning of this section. Additionally, this connective is the one which is introduced in the tectogrammar when we make use of function abstraction. It is worth noting that unlike classical implication, linear implication does not allow for vacuous abstraction. More precisely, this is due to the absence of the structural rule of weakening in linear logic.

As far as our type theory goes, to keep things simple, we assume very few basic types: \(\text{NP}_{\text{nom}}\), \(\text{NP}_{\text{acc}}\), \(\text{S}\), and (for the time being) \(\text{PP}_{\text{to}}\), the type of to-marked prepositional phrases. This is a preliminary setup for the sake of simplicity; it will undoubtedly be necessary to add more primitives to the grammar later. No theoretical claim is made that these are the only types we will eventually need.

\(^{18}\)For an alternative introduction to using the lambda calculus in a phenogrammatical context, see [Oeh94].
The reader will note that a distinction is made in the type inventory between nominative and accusative NPs. This division is well-motivated by the pronouns, whose distribution differs based on their case. The choice to treat these as separate types is one made for the sake of expository simplicity. Nevertheless, most English NPs are syncretic between nominative and accusative case, and this has the consequence of introducing an undesirable amount of lexical redundancy, with respect to every other non-pronominal lexical NP (e.g. proper names), as well as making the details of the construction of complex NPs more difficult. To give a more precise analysis which captures the generalizations observable requires some implementation of syncretism in the grammar. The addition of dependent product types in a manner similar to the one outlined in [dGM07] is one of several potential solutions, but the amount of technical detail required to do so here somewhat outweighs their usefulness. It must be noted that the dependent product appears to be a useful tool for describing languages which are morphologically richer than English, and Ranta’s GF [Ran04] has had success in these exact areas.

In summary:

- NP\textsubscript{nom}, NP\textsubscript{acc}, S, PP\textsubscript{to} are types.
- If A and B are types, then A \rightarrow B is a type.

4 Further Details of the Grammar Formalism

The grammar framework which will be outlined here can be described as Curryesque, logical, and relational. It has its origin in Pollard’s Higher Order Grammar (HOG, [Pol04]), but has commonalities with, and is inspired by, both constraint-based grammars such as Head-Driven Phrase Structure Grammar, other logical grammars, in particular Morrill’s Type Logical Grammar (TLG, [Mor94]) and de Groote’s Abstract Categorial Grammar (ACG, [dG01]).

We describe the grammar as “Curryesque”, because it maintains a distinction between phenogrammar and tectogrammar, as well as semantics. In this way, the grammar generates triples consisting of a phenogrammatical representation, a tectogrammatical representation, and a semantic representation. Each member of the triple is a term-type pair. Intuitively, terms can be thought of as an aspect of a particular linguistic expression, and types can be thought of as the category to which that piece of the expression belongs. While the logic underlying the type systems for phenogrammar and semantics is intuitionistic, the type logic of the tectogrammar is linear logic. In the current work, since the tecto-terms record primarily the order in which a derivation proceeded, they will unilaterally be suppressed. Their use becomes increasingly necessary when we begin to consider languages which have a more complicated morphological system than English. The interface between the three derivational systems is realized in inference rules which govern not only the inference steps within each logic, but the relations between the three logics. Lexical entries will take the following form:

\[
\vdash \begin{array}{c}
\text{pheno-term} : \text{Pheno-Type} \\
\text{tecto-term} : \text{Tecto-Type} \\
\text{semantic term} : \text{Semantic Type}
\end{array}
\]

When the phenogrammatical and semantic types may be easily inferred, this is abbreviated (in [MP10] and elsewhere) in the following manner:
However, since we will be making more profligate use of pheno-types than is typically assumed, we will write the unabbreviated versions, in the interest of placing explicitness over succinctness.

This framework differs notably from ACG in that the mapping between one logic and another is relational, rather than functional. We wish to allow, on one hand, multiple phenogrammatical realizations for the same tectogrammatical structure, in order to reify the notions of flexibility in word order. We might wish to consider allowing multiple semantic representations for the same tectogrammatical structure, in order to allow for ‘information structural’ differences in meaning imposed by prosody, for example, although this is a somewhat more controversial move. Since semantics are not the focus of this work, the semantic terms and types will be systematically omitted here, and the nature of the syntax-semantics interface will not be explored further.

Like many other kinds of categorial grammar, and HPSG, the framework outlined here is highly lexicalized, and lexical entries are modeled as axioms of the overall logical system, i.e. as triples. The primary method of combination is through application, the rule for which is rendered as follows:

\[
\Gamma \vdash \begin{bmatrix}
  f' : S \rightarrow T \\
  f : A \rightarrow B \\
  f'' : M \rightarrow N
\end{bmatrix}
\quad \Delta \vdash \begin{bmatrix}
  a' : S \\
  a : A \\
  a'' : M
\end{bmatrix}
\]

\[\text{App} \quad \Gamma, \Delta \vdash \begin{bmatrix}
  f'(a') : T \\
  f(a) : B \\
  f''(a'') : N
\end{bmatrix}\]

While this rule appears daunting, it can be thought of as three ‘versions’ of the same rule, all performed in parallel. That is, it is just three concurrent instances of implication elimination (intuitionistic or linear), which are given here in separated form:

\[
\Gamma \vdash f' : S \rightarrow T \quad \Delta \vdash a' : S
\]

\[\text{App (Pheno)} \quad \Gamma, \Delta \vdash f'(a') : T\]

\[
\Gamma \vdash f : A \rightarrow B \quad \Delta \vdash a : A
\]

\[\text{App (Tecto)} \quad \Gamma, \Delta \vdash f(a) : B\]

\[
\Gamma \vdash f'' : M \rightarrow N \quad \Delta \vdash a'' : M
\]

\[\text{App (Semantics)} \quad \Gamma, \Delta \vdash f''(a'') : N\]

\[\text{19} \quad \text{and possibly differences in intonation, although since this particular question has bearing on the relationship between tectogrammar and semantics, we do not wish to address it here.}\]

\[\text{20} \quad \text{cf. [Ste00] which argues against this.}\]

\[\text{21} \quad \text{The naming and notation of this rule imply that it is function application. This is not the case, since the type system for the tectogrammar is based on linear, rather than intuitionistic logic. Nevertheless, one will not be led terribly astray here by thinking of it as ‘like’ function application.}\]

\[\text{22} \quad \text{although it can be simplified in a manner corresponding to the abbreviated form for lexical entries given above.}\]
This amounts to thinking of certain expressions (e.g. intransitive verbs) as ‘looking for’ other expressions (e.g. nominative noun phrases), which they can then combine with in principled ways to produce expressions of other categories (e.g. sentences). The tripartite nature of the rule reflects that we take this process to occur concurrently in the phenogrammar, the tectogrammar, and the meaning, rather than taking one component to be primary and deriving the others from it.

The formal aspects of the phenogrammatical and tectogrammatical systems will be more specifically examined in the next section. Our logics also makes use of pairing and projection, which will be discussed in the next section. For the sake of brevity, we wish to provide the following as a table of the variables used here, the types they may be presumed to have, where relevant, and a brief description of each.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y, z</td>
<td>St</td>
<td>strings corresponding to argument positions in linearization functions</td>
</tr>
<tr>
<td>h</td>
<td>Wd</td>
<td>corresponds to the position of the ‘word itself’ in a linearization function</td>
</tr>
<tr>
<td>s, t</td>
<td>St</td>
<td>strings used in the discussion of phenogrammar</td>
</tr>
<tr>
<td>k</td>
<td>Wd</td>
<td>a word used as a ‘placeholder’ in affixes</td>
</tr>
<tr>
<td>p</td>
<td>Phn</td>
<td>a phenogrammatical pair</td>
</tr>
<tr>
<td>b</td>
<td>varies</td>
<td>a tecto-term</td>
</tr>
<tr>
<td>A, B</td>
<td>N/A</td>
<td>tecto-types</td>
</tr>
<tr>
<td>S, T</td>
<td>N/A</td>
<td>pheno-types</td>
</tr>
<tr>
<td>M, N</td>
<td>N/A</td>
<td>semantic types</td>
</tr>
<tr>
<td>Γ, Δ</td>
<td>N/A</td>
<td>metavariables over contexts</td>
</tr>
<tr>
<td>f, g</td>
<td>varies</td>
<td>mnemonic for ‘functor’, generally</td>
</tr>
<tr>
<td>a</td>
<td>varies</td>
<td>mnemonic for ‘argument’, generally</td>
</tr>
</tbody>
</table>

### 4.1 Phenogrammatical Refinements

The phenogrammatical framework I outline here takes one step closer to making morphological / phonological distinctions more meaningful. I wish to make three major distinctions, which I take to be uncontroversial: there are words, there are larger linguistic expressions which comprise more than one word, and there are linguistic expressions which are smaller than words, e.g. bound morphemes. The particular ones we will be dealing with here are affixes. To make this notion a little more precise, we need to have a slightly richer type system. If we take words to be basic, and make \( Wd \) a basic type, then we can think of strings as simply groups or lists of words. It is theoretically possible to construct strings over any type, but for our purposes here, we will only be concerned with strings of words, which are given the type \( Wd^* \). Throughout the rest of this work, as we are unconcerned with strings of other types, we will simply write \( St \) which is intended to be mnemonic for “string”.

Because we will be dealing with fairly long functional types, for the sake of legibility, they will be abbreviated as follows:

- \( St_0 \) abbreviates \( St \) (i.e. not functional, an ordinary string)
- \( St_{n+1} \) abbreviates \( St \rightarrow St_n \)

e.g. \( St_2 \) abbreviates the type \( St \rightarrow St \rightarrow St \).

It is worth noting at this point that there is a distinction to be made between a word, and a length-one string containing that word. We identify the canonical injection and name it \( toSt \), an operation which maps every word to the length-one string containing it:
⊢ toSt : Wd → St0

This fundamental difference between words and strings will be profitably exploited later. For the sake of notational clarity, a word will appear in sans-serif, and the length-one string containing that word in small capital letters, e.g. a word and a WORD, respectively.

The affixes are slightly more complicated. The constants which represent the affixes themselves (e.g. -im) will be taken to be of type Af, which allows us to define a new mode of concatenation, called suffixation:

⊢ ◦ suf : (Wd ∧ Af) → Wd

This operation takes an affix, and a word, and returns a new word. The actual phenogrammar of the affixes themselves will be somewhat different, since they will be rendered as functions which take words as their arguments, and which return words in kind.

In addition to the implication type constructor, we will make use of the conjunction type constructor ∧, the inference rules for which come equipped with three associated term constructors. The pairing term constructor (,) is associated with the rule of ∧-Introduction, and allows for the creation of ordered pairs of phenogrammatical terms. Likewise, conjunction also has term constructors associated with its elimination rules: the left (first) and right (second) projections, π and π′ respectively. The reader will note that these rules are just term-annotated versions of the rules for conjunction given earlier.

\[
\begin{align*}
&\Gamma \vdash s : S \\
&\Delta \vdash t : T \\
&\therefore \Gamma, \Delta \vdash (s, t) : S \land T \\
\end{align*}
\]

\[
\begin{align*}
&\Gamma \vdash s : S \land T \\
&\therefore \Gamma \vdash \pi(s) : S \\
\end{align*}
\]

\[
\begin{align*}
&\Gamma \vdash s : S \land T \\
&\therefore \Gamma \vdash \pi′(s) : T \\
\end{align*}
\]

This now gives us a phenogrammatical logic which is positive and intuitionistic. Function application corresponds to implication elimination, and function abstraction to implication introduction. Pairing corresponds to conjunction introduction, and the two projection functions to conjunction elimination. It should be noted that the introduction rule is not explicitly used here (since the paired formulas we are concerned with are lexically specified).

In summary:

- Wd, St, Af are types.
- If A and B are types, then A → B is a type.
- If A and B are types, then A ∧ B is a type.

This technology is specifically used in the way we will represent "functional" lexical entries. While the system outlined in the previous section wrote its phenogrammatical constants in the body of lambda terms, the refinement here treats the words themselves and their linear combinatoric potential as separate at the outset. This restores a domain to the grammar within which broadly-defined morphological operations can take place, making words themselves susceptible to morphological alteration. We choose to represent functional ‘heads’ like verbs, case-marking prepositions, and the like as pairs, whose whose first member is the linear argument structure that a

\[\text{The choice to render words and length-one strings in different fonts introduces an unfortunate piece of notational ambiguity: the suffixation operation } \circ \text{suf may appear to be polymorphic. Our intent is that it function as defined above, i.e. over words and affixes, rather than over strings.}\]
word is associated with, and whose second member is the word itself. For example, the phenogrammatical part of the lexical entry above for *kissed* would be modified to the following (although the earlier term is retrievable, as we will see):

\[ \vdash (\lambda_h \lambda_x.x \circ \text{toSt}(h) \circ y, \text{kissed}) : \text{Ph}_2 \]

The lexical entries for the phenogrammatical part of *Hildegard* and *Elvis*, being strings already (i.e. non-functional) remain the same\(^{24}\). This phenogrammatical representation for *kissed* makes use of an apparently new type, \(\text{Ph}_2\). Much like the numerically subscripted string type above, this is simply an abbreviation.

* \(\text{Ph}_n\) abbreviates \((\text{Wd} \to \text{St}_n) \land \text{Wd}\)

So if we decompile the pheno-type of the lexical entry for *kissed*, it is easy to see that \(\text{Ph}_2\) abbreviates \((\text{Wd} \to \text{St}_2) \land \text{Wd}\), which abbreviates \((\text{Wd} \to \text{St} \to \text{St} \to \text{St}) \land \text{Wd}\), the correct typing judgment for the paired term.

The variable \(h\) corresponds to the position that the word itself will ultimately hold in the expression. Notice that the original, Oehrle-style term is immediately recoverable if we simply apply the lambda term (the left projection) to the word (the right projection). We will refer to this entire process as “encapsulation”, because it is in this way that the actual word in question is closed off from further morphological alteration. This process is represented by the non-logical rule called Encap:

\[
\Gamma \vdash \left[ \begin{array}{l}
\ p : \text{Ph}_n \\
\ b : \text{A}
\end{array} \right] \quad \text{Encap} \\
\Gamma \vdash \left[ \begin{array}{l}
\ \text{eval} (p) : \text{St}_n \\
\ b : \text{A}
\end{array} \right]
\]

where the function \(\text{eval}\) is defined in the following manner:

\[
\text{eval} = \text{def} \lambda p.\pi(p)(\pi'(p))
\]

So the (phenogrammatical part of the) derivation of *Hildegard kissed Elvis* now looks like the following:

\[
\vdash (\lambda_h \lambda_y.x \circ \text{toSt}(h) \circ y, \text{kissed}) : \text{Ph}_2 \quad \text{Encap} \\
\vdash (\lambda_x.x \circ \text{kissed} \circ y : \text{St}_2) \quad \vdash \left[ \begin{array}{l}
\ \text{ELVIS} : \text{St}
\end{array} \right] \quad \text{App} \\
\vdash (\lambda_x.x \circ \text{kissed} \circ \text{ELVIS} : \text{St}_1) \quad \vdash \left[ \begin{array}{l}
\ \text{HILDEGARD} : \text{St}
\end{array} \right] \quad \text{App}
\]

\(^{24}\)It would certainly be possible to treat them analogously to the functional entries, with an identity in the tecto component, but in absence of a compelling reason to do so, we choose to keep things as simple as possible.
4.2 Semantics

The semantics of English pronouns are unquestionably important, and much ink has been spilled on exactly how to represent them. For the time being, I wish to remain as agnostic about them as possible. Since our grammar formalism is relational (i.e. the relations between components are not functions), the reader will be free to assume whatever logical view of semantics is desirable; it is my intention to allow interface to a wide variety of semantic theories. This is unquestionably the largest piece of latitude I wish to ask, but as my central point here is primarily one about the relationship between phenogrammar and tectogrammar, I hope that the omission can be temporarily overlooked, as it is one that I hope to remedy in the future.

4.3 Pheno-Tecto Derivation

A simultaneous phenogrammatical and tectogrammatical derivation of Hildegard kissed Elvis is provided here. As the axioms and inferences become increasingly large, a presentation style is used whereby each statement is numbered, and abbreviated in an obvious fashion. A numbered line-by-line derivation follows, so the reader can see both the ‘shape’ of the proof, as well as the specifics simultaneously, and can reconstruct the entire proof as necessary.

1. kisses
2. kisses 3. Elvis
4. kisses Elvis
5. Hildegard
6. Hildegard kissed Elvis

1. \[\left(\lambda_h \lambda y . x \circ \text{toSt}(h) \circ y, \text{kissed}\right) : \text{Ph}_2 \]
2. \[\left(\lambda y . x \circ \text{KISSED} \circ y: \text{St}_2\right) \]

The first two lines of the proof show the verb kissed undergoing encapsulation, thus making it insusceptible to affixation. Since the pronominal affixes may only attach to terms of type Wd, and there are no longer any terms of that type, any future combination must take place between words, not within a word. The Encap rule takes the word kissed and passes it over to the linearization function, which subsequently reduces to the Oehrle-style term, awaiting string arguments whose tecto-types are those of an accusative and a nominative noun phrase.

3. \[\text{ELVIS} : \text{St} \]
4. \[\left(\lambda x . x \circ \text{KISSED} \circ \text{ELVIS} : \text{St}_1\right) \]

We apply the function of the encapsulated entry for kissed to its argument. The string Elvis is provided as an argument to the linearization function, which accurately positions it to the immediate right of the verb. The tectogrammatical accusative NP argument is eliminated, leaving us with an expression which is looking to combine with a nominative NP string-type argument.

5. \[\text{HILDEGARD} : \text{St} \]
In these lines, we show that *Hildegard* is typed as a nominative NP, and a string.

6. \[ \text{HILDEGARD o KISSED o ELVIS : St} \] (FA)

By applying the *kissed Elvis* function to *Hildegard*, we are left with the complete, single string sentence *Hildegard kissed Elvis*.

5 Formalization of Analysis

This section comprises enough lexical entries to derive the example sentences (9), (10), and (12), and to disallow the ungrammatical (11), as well as discussion of these entries. Complete line-by-line derivations follow in the appendix.

5.1 Lexical Entries

5.1.1 Verbs

\[ (\lambda h \lambda y z z . x o \tau o \text{St}(h) o y o z , \text{told}) : \text{Ph}_3 \text{ (i.e. \((Wd \rightarrow \text{St}_3) \land Wd\))} \]

\[ : \text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{nom}} \rightarrow \text{S} \]

This lexical entry captures several important things about the verb *told* simultaneously, but they should be decompiled somewhat. First, the phenogrammatical type, \( \text{Ph}_3 \) abbreviates the type \((Wd \rightarrow \text{St}_3) \land Wd\). Notice that *told* is itself a word, but it also has a lambda term indicating how it will ultimately combine with other material, in terms of linear order. It is important here that *told* has status as a word, rather than a string of words (even as a length-one string of words), since these two differ in their combinatoric potential in the system described here. Specifically, affixes will generally be allowed to combine with words, but not with strings of words\(^{25}\). This is a step in the right direction as far as accounting for the idiosyncratic behavior noted above.

Now, the first half of the phenogrammatical type, \( Wd \rightarrow \text{St}_3 \), abbreviates the ‘Curried’ type \( Wd \rightarrow \text{St} \rightarrow \text{St} \rightarrow \text{St} \rightarrow \text{St} \). This is perhaps not what we would immediately expect, given the presumed argument structure of *told*. Its two objects are represented in the term above by the variables \( y \) and \( z \), and its subject by \( x \). The variable \( h \) is a placeholder for the eventual position of the word itself, which we will obtain when we are done messing about with affixation.

Tectogramatically speaking, the typing judgment tells us that the verb *told* requires two accusative NP arguments, and a nominative NP argument, and will then construct a finite sentence. The reader may notice the mismatch between the pheno-type \( \text{Ph}_3 \) and the tecto-type \( \text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{nom}} \rightarrow \text{S} \); the pheno-type contains one more argument (of type \( Wd \)). This is by design, and reflects the fact that the verb *told* is unable to combine with other word-level material until any affixation in which it is involved is complete. This is an explicit reification of the idea of different levels of attachment boundaries in the syntax – phonology interface. It will be possible at any time to cause *told* to combine with other non-affixal material by ‘encapsulating’ it, allowing the verb to take its string-linear position in the associated pheno-term. This process will reduce the arguments of the pheno-term by one, bringing it directly in line with what we would expect to see, given the tecto-term. In the meantime, it is precisely this mismatch that blocks the derivation of sentences with the reduced pronouns in unacceptable positions. This blocking is

\(^{25}\)noting again that the aforementioned phrasal suffix possessive ‘*-s*’ behaves differently.
explored after the technical details of the pronominal affixes, at the end of the next subsection.

\[
\begin{align*}
\vdash (\lambda_h \lambda_{yx}.x \circ \text{toSt}(h) \circ y \circ z, \text{told}) & : \text{Ph}_3 \text{ (i.e. } (\text{Wd} \to \text{St}_3) \land \text{Wd}) \\
\vdash (\text{NP}_{\text{acc}} \to \text{PP}_\text{to} \to \text{NP}_{\text{nom}} \to \text{S})
\end{align*}
\]

This is the lexical entry for the version of \text{told} that selects a \text{to}-marked prepositional phrase as one of its arguments, rather than a noun phrase. This change is noted in the tectogrammatical type, but the lexical entry is otherwise identical in the phenogrammar.

### 5.1.2 Pronominal Affixes

\[
\begin{align*}
\vdash \lambda_p. (\lambda_k. \pi(p)(k)(\epsilon), \pi'(p) \circ \text{suf -im}) & : \text{Ph}_n \to \text{Ph}_{n-1} \\
\vdash (\text{NP}_{\text{acc}} \to A) \to A
\end{align*}
\]

\[
\begin{align*}
\vdash \lambda_p. (\lambda_k. \pi(p)(k)(\epsilon), \pi'(p) \circ \text{suf -it}) & : \text{Ph}_n \to \text{Ph}_{n-1} \\
\vdash (\text{NP}_{\text{acc}} \to A) \to A
\end{align*}
\]

It is assumed that ordinary pronouns would be taken as string-type NP arguments in a manner similar to the proper names, albeit with their case lexically specified in some cases. However, the reduced pronominal affixes function somewhat differently. It is easiest to examine their phenogrammar and tectogrammar separately. Tectogrammatically, the affixes are type-raised; they are functions which will take as their arguments the expressions which would ordinarily select regular accusative NPs. They are given the polymorphic type \((\text{NP}_{\text{acc}} \to A) \to A\), meaning that if they find something looking for an NP argument, they can combine with it, and return whatever the original return type of that expression is. For example, take the type of the verb \text{kissed}, \text{NP}_{\text{acc}} \to \text{NP}_{\text{nom}} \to \text{S}. This is of the right type to combine with an affix, if we take \text{A} to be \text{NP}_{\text{nom}} \to \text{S}. So the instantiated tecto-type of the pronoun would be \((\text{NP}_{\text{acc}} \to \text{NP}_{\text{nom}} \to \text{S}) \to (\text{NP}_{\text{nom}} \to \text{S})\). In this way, they can ‘reduce the valence’ of a transitive verb in the same way that ‘saturating’ it by providing an ordinary NP argument can. The only difference is in which expression is the function and which is the argument.

The phenogrammatical side of things is a bit more complicated. Examining the type, \text{Ph}_n \to \text{Ph}_{n-1}, it should be obvious that an analogous process is going on. The resulting function will be looking for one less string in the phenogrammar, as it should when the string-linear position occupied by an ordinary NP is instead taken up by the pronominal affix. Importantly, though, the pheno-term is still a pair consisting of a word and its linearization function. Since encapsulation has yet to occur, the word is still susceptible to further affixation, allowing pronominal affixes to ‘stack’ if desired. More importantly, in order to combine with words rather than affixes, an expression whose pheno-type is \text{Ph}_n must undergo encapsulation, closing it off from affixation, and effectively blocking the derivation of sentences such as (11), \text{Martha told Noel-it}, which will be addressed in due course.

The pheno-term for the affixes performs multiple tasks simultaneously. It takes as its argument an unencapsulated pheno-term \(p\), i.e., a pair of type \((\text{Wd} \to \text{St}_n) \land \text{Wd}\). It takes the right projection of that argument \((\pi'(p))\), the word itself, and attaches the pronominal affix to that word via suffixation. The left projection of the argument will be the linearization function. To that, the pronoun first provides a ‘placeholder’ variable \(k\), above, then binds that variable. The purpose of this is to maintain the position of the word itself within the linear order. Next, the affix provides the empty string as an argument, ‘filling in’ the linear position where the accusative NP argument
would ordinarily go\(^{26}\). Finally it collects the newly affixed word, and the reduced linearization function, and creates a new paired pheno-term.

This process of affixation is illustrated here with the verb phrase *kissed-im*:

\[
\Gamma \vdash \left[ \lambda \lambda x. \, \pi \cdot (\lambda p. \, \pi(p)(\epsilon), \pi'(p) \circ \text{suf}\circ \text{im}) : \text{Ph}_n \rightarrow \text{Ph}_{n-1} \right] \quad \Gamma \vdash \left[ \lambda \lambda y z. \, x \circ \text{toSt}(h) \circ \, y, \, \text{kissed} \circ \text{suf}\circ \text{im} : \text{NP}_{acc} \rightarrow \text{NP}_{acc} \rightarrow \text{S} \right]
\]

\[
\vdash \left[ \lambda \lambda x. \, x \circ \text{toSt}(k) \circ \, \epsilon, \, \text{kissed} \circ \text{suf}\circ \text{im} : \text{Ph}_1 \right] \quad \text{FA}
\]

Now, we encapsulate the new verb phrase:

\[
\Gamma \vdash \left[ \lambda \lambda x. \, x \circ \text{toSt}(k) \circ \, \epsilon, \, \text{kissed} \circ \text{suf}\circ \text{im} : \text{Ph}_1 \right] \quad \Gamma \vdash \left[ \lambda \lambda x. \, x \circ \text{toSt}(k) \circ \, \epsilon, \, \text{kissed} \circ \text{suf}\circ \text{im} : \text{NP}_{nom} \rightarrow \text{S} \right]
\]

\[
\Gamma \vdash \left[ \lambda \lambda x. \, x \circ \text{KISSED}\circ \text{IM} \circ \, \epsilon : \text{St}_1 \right] \quad \text{Encap}
\]

\[
\Gamma \vdash \left[ \lambda \lambda x. \, x \circ \text{KISSED}\circ \text{IM} \circ \, \epsilon : \text{NP}_{nom} \rightarrow \text{S} \right]
\]

We are now left with the verb phrase *kissed-im*, which is ready to combine with its subject argument. Since the empty string is the two-sided identity for strings, it will ultimately vanish from the pheno-term.

The interplay between the phenogrammar and tectogrammar of the pronominal affixes is well illustrated by the ungrammatical example (11), *Martha told Noel-it*. In an attempted derivation of this sentence, -it is tectogrammatically incompatible with *Noel* and *Martha*, leaving the possibilities that it combines with *told* or *told Noel*, either of which, tectogrammatically speaking, would be perfectly acceptable, since -it attaches to expressions which are themselves looking for accusative NPs. If -it combines with *told*, then the lexical entry for -it ensures that the resulting expression will be the grammatical *told-it*. If, however, we attempt to combine -it with *told Noel*, we find that since -it operates on word / linearization function pairs (type Ph\(_n\), i.e. type \((\text{Wd} \rightarrow \text{St}\_n) \land \text{Wd}\)) it is prevented from combining with *told Noel* (type St\(_2\)) on the grounds that their types are incompatible.

### 5.1.3 Other Material

\[
\Gamma \vdash \left[ \text{MARTHA} : \text{St} \right] \quad \Gamma \vdash \left[ \text{MARTHA} : \text{NP}_{acc} \right]
\]

\[
\Gamma \vdash \left[ \text{NOEL} : \text{St} \right] \quad \Gamma \vdash \left[ \text{NOEL} : \text{NP}_{acc} \right]
\]

\[
\Gamma \vdash \left[ \text{THE PLOT OF GRAVITY’S RAINBOW} : \text{St} \right] \quad \Gamma \vdash \left[ \text{THE PLOT OF GRAVITY’S RAINBOW} : \text{NP}_{acc} \right]
\]

As the internal structure of the noun phrase *the plot of gravity’s rainbow* is tangential to the point here, I hope the reader will grant me the latitude to treat it simply and analogously with the NPs *Martha* and *Noel*. Examining the NPs, we can see that in the phenogrammar, they are simply strings. Tectogrammatically speaking, they come in both nominative and accusative varieties. So we are free to choose whichever version we desire for these NPs.

\(^{26}\)This particular strategy owes much to [Mus03].
The ‘case-marking’ preposition to looks very similar to the verbs, phenogrammatically speaking. It has the Ph₁ ‘pair’ type, licensing it as a word to which reduced pronouns may affix themselves. It linearizes its argument string to its immediate right. Tectogrammatically, it selects an accusative NP argument, and returns a term of type PP\textsubscript{to}, the type we use for to-marked prepositional phrases. We note that these types of PPs bear a non-implicative / non-functor type, making them distinct from adjunct PPs, which, as is common to most categorial frameworks, we take to select their (VP or NP) arguments, i.e., to be implicative types. Marked PPs are selected by verbs in a manner similar to NP argument selection, and cannot be said to modify anything, whereas adjunct PPs are modifiers proper, and as such, will have modifier types such as (\text{NP\textsubscript{nom} → S}) → (\text{NP\textsubscript{nom} → S}), for example.

6 Future Research

6.1 Semantics

The most obvious gap in the work is the lack of any semantics associated with the grammar. Surface word order clearly has semantic and pragmatic effects. In English, topicalization represents an alternative word order for declarative sentences which is associated with presentational or contrastive focus, as well as with a characteristic tune (“B accent”). Pollard (p.c.) notes that a similar effect is exhibited with “in-situ” topicalization, where the expression bearing the B accent remains in place, e.g., \textit{I gave the BOOK to John}. It is easy to see that different utterances which appear to be syntactically identical can bear different intonational contours, with corresponding differences in meaning. The same string of words can be used as a declarative sentence, as the answer to a question, as a question in and of itself, as a member of a list of facts, as the selection of a choice, etc. Each of these typically correlates with some characteristic intonational contour. Unless we wish to claim that these differences are in some way syntactic, these facts motivate an analysis which, minimally, contains some notion of prosodic constituency suitable to describe intonational contours. Such an analysis must be amenable to a semantics which models not only truth-conditional semantics, but aspects of “projective meaning” such as presupposition and conventional implicature, as well as information-structural meaning and context change potential. Insofar as most semantic theories I know about are also lexical in nature, I trust that the general phenogrammatical / tectogrammatical strategy outlined here can be extended to such. As to the specific nature of the interface between syntax and semantics, more thinking must clearly be done.

6.2 Coordination

One notable source of complication is the questionable acceptability of sentences featuring coordinated verbs. Consider the following:

(17) When John and Mary saw the twins emerge safely from the cave, they joyfully hugged-em and kissed-em.

(18) ? When John and Mary saw the twins emerge safely from the cave, they joyfully hugged and kissed-em.
My analysis predicts that (18) should be unacceptable, and while it strikes me as being strange, I grant that it may be possible for some speakers. The potential for this type of structure differentiates English pronominal affixes from, say, French clitics, where a similar structure is impossible:

\[(19) \quad \text{* Pierre les voit et écoute ([MS97]'s (10))} \]

I note that the acceptability of such sentences increases when the coordinated verbs are ones which are likely to appear together, either for reasons of idiomaticity, or because they bear some kind of semantic / pragmatic association. For example, (18) seems preferable to the following:

\[(20) \quad \text{* When John and Mary saw the twins emerge safely from the cave, they joyfully hugged and ate-em.} \]

The oddness of (20) is possibly due to the unlikeliness of the scenario, although I note that the corresponding example is perfectly grammatical, albeit unpleasant:

\[(21) \quad \text{When John and Mary saw the twins emerge safely from the cave, they joyfully hugged-em and ate-em.} \]

My preliminary, and extremely speculative suspicion is that in such sentences, the actual occurrence of ‘and’ may itself be affixal (and is realized as [an]), and in light of the data, the current analysis mandates that reduced ‘and’ is involved in word formation processes similar to those of the reduced pronouns. More phonological evidence can surely be brought as to the plausibility of this idea, in particular investigations into phenomena which occur at, say word boundaries, but not morpheme boundaries, or vice versa. Part of the difficulty is an unclarity as to which and how many levels of representation are necessary to discuss the syntax-phonology interface. It is also necessary to note the difficulty of obtaining reasonable data, due to the unavoidable register clash associated with using reduced pronouns and unreduced ‘and’. These data are under investigation (in particular the status of ‘and’), and it would be premature to offer anything more than the suggestion of an analysis at this point.

6.3 Extensibility

In addition to the coordination data given above, it remains to address how this analysis provides accounts of verb-particle constructions like those exhibited in (5) through (8). Such an extension, though fairly straightforward, is complicated by the variability in word order displayed by the particle itself, with respect to full NP objects. Similarly, the behavior of the pronominal affixes with respect to the passive requires further investigation. As it stands, there is nothing that rules out sentences like:

\[(22) \quad \text{? John was given’em.} \]

The behavior of the pronominal affixes with respect to unbounded dependency constructions likewise remains unclear. Consider the following:

\[(23) \quad \text{John gave Mary the goats.} \]
\[(24) \quad \text{Who did John give the goats?} \]
\[(25) \quad \text{* Who did John give-em? (under the reading where -em is standing in for the goats)} \]
We note that by assuming a theory of extraction in the manner of [MP10], it is almost certainly possible to integrate the analysis given here to account for the ungrammaticality of (25), though this is left for the future.

6.4 A “Real” Phenogrammar

The other primary outstanding issue is one that has already been alluded to, namely the impoverished nature of the actual structure of the phenogrammatical logic. Insofar as types are taken to represent relevant categories for the classification of linguistic material, the types of the phenogrammar should be those which allow for the most complete and accurate description of data possible. Unfortunately, there is little agreement as to exactly what those categories are. To that end, I have endeavored in this work to choose those categories which I think are the least contentious; there are linguistic expressions that are something like words, there are expressions that are smaller than words, and there are expressions that are larger than words.

One of the ultimate goals of the investigation that begins here is a descriptively adequate calculus for notating the manifold aspects of phenogrammar. Of particular interest to me are those parts of ‘phonology’ that potentially have some bearing on morphosyntactic structure, in particular, prosodic information. Ideally, we would be able to render information about intonation, stress, tone, and the like in a well-founded way using a more full-featured logical grammar than the zygotic version presented here. It is my intention to make this calculus somewhat mutable, and useful to those who work in different areas of prosodic representation (ToBI, Ladd-esque, etc).

Additionally, I take pronominal affixes and the English phrasal affix possessive ‘-s’ to be instances of two different phenomena. Pronominal affixes, as outlined here, are unable to attach phenogrammatically to anything larger than a word, and mimic the function of accusative noun phrases in the tectogrammar. Phrasal affixes, however, are a slightly more complicated case, in that they potentially need to be able to attach to an entire noun phrase, tectogrammatically speaking, but to the last phenogrammatical word in the phrase. This is envisioned as a recursive process terminating in phenogrammatical suffixation in the same general manner as the pronominal affixes. The technical details remain beyond the scope of this work, but will be addressed in the future.

7 Conclusion

While the idea of separating the phenogrammatical from the tectogrammatical component of a grammar is not new, it has yet to be deployed with a view towards looking at more complicated structures. Syntacticians have customarily looked at phenogrammar as being primarily concerned with word order. In this paper we have shown how a categorial grammar with a richer phenogrammatical logic can be profitably employed in accounting for data which heretofore has lain primarily in the domain of phonology. Among other things, the phenogrammar is envisioned as the environment which mediates between ‘larger’ syntactic structures and ‘smaller’ phonological ones, making it an appropriate tool for beginning to examine the interactions between syntax, morphology, phonology, and prosody. While there are serious questions about the fundamental architecture of a logical phenogrammar, the hope is to provide a framework within which the interfaces between these semi-disparate subfields can be explored, allowing for a high degree of future collaboration.

---

I wish to use this word as atheoretically as possible, in suspension of the unfortunate historical tendency for syntacticians and phonologists each to regard “word order” as within the purview of the other.
References


A Appendix

A.1 Sample Derivations

A.1.1 Ordinary Combination

1. ⊢ \[ (\lambda h \lambda_{yxz}.x \circ toSt(h) \circ y \circ z, told) : \text{Ph}_3 \text{(i.e. } (Wd \to St_3) \land Wd) \] (Lexical)

2. ⊢ \[ \lambda_{yxz}.x \circ TOLD \circ y \circ z : \text{St}_3 \] (Encap)

3. ⊢ \[ \text{NOEL} : \text{St}_1 \] (Lexical)

4. ⊢ \[ \lambda_{xzx}.x \circ TOLD \circ NOEL \circ z : \text{St}_2 \] (App)

5. ⊢ \[ \text{THE PLOT OF GRAVITY'S RAINBOW} : \text{St}_1 \] (Lexical)

6. ⊢ \[ \lambda_{xzx}.x \circ TOLD \circ NOEL \circ THE PLOT OF GRAVITY'S RAINBOW : \text{St}_1 \] (Lexical)

7. ⊢ \[ \text{MARTHA} : \text{St}_1 \] (Lexical)

8. ⊢ \[ \text{MARTHA} \circ TOLD \circ NOEL \circ THE PLOT OF GRAVITY'S RAINBOW : \text{S} \] (App)

A.1.2 Affixation

1. ⊢ \[ (\lambda h \lambda_{yxz}.x \circ toSt(h) \circ y \circ z, told) : \text{Ph}_3 \text{(i.e. } (Wd \to St_3) \land Wd) \] (Lexical)

2. ⊢ \[ \lambda_{p.(\lambda k.\pi(p)(k) \circ \epsilon \circ \pi' \circ p) \circ suf \circ -im} : \text{Ph}_n \to \text{Ph}_{n-1} \] (Lexical)

3. ⊢ \[ (\lambda h \lambda_{yxz}.x \circ toSt(k) \circ \epsilon \circ z, told \circ suf \circ -im) : \text{Ph}_2 \] (App)
4. \[\lambda_{x}.x \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ \text{o} \circ z : \text{St}_{2} \]  
\[\text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{nom}} \rightarrow \text{S} \] (Encap)

5. \[\lambda_{x}.x \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ \text{o} \circ z : \text{St} \]  
\[\text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{nom}} \rightarrow \text{S} \] (Lexical)

6. \[\lambda_{x}.x \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ \text{o} \circ z : \text{St} \]  
\[\text{NP}_{\text{nom}} \rightarrow \text{S} \] (App)

7. \[\text{MARThA} : \text{St} \]  
\[\text{NP}_{\text{nom}} \rightarrow \text{S} \] (Lexical)

8. \[\text{MARThA} \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ \text{o} \circ z : \text{St} \]  
\[\text{NP}_{\text{nom}} \rightarrow \text{S} \] (App)

8'. \[\text{MARThA} \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ \text{o} \circ z : \text{St} \] (reduction)

A.1.3 Affix–stacking

\[
1. \text{told} \quad 2. \text{-im} \quad \text{FA} \quad 3. \text{told-im} \quad 4. \text{-it} \quad \text{FA} \quad 5. \text{told-im-it} \quad \text{Encap} \quad 6. \text{told-im-it} \quad 7. \text{Martha} \quad \text{FA} \quad 8. \text{Martha told-it}
\]

1. \[\lambda_{x}.\lambda_{y}.x \circ \text{toSt}(h) \circ y \circ z \circ \text{told} : \text{Ph}_{3} \] (i.e. \((Wd \rightarrow \text{St}_{3}) \land Wd\))
\[\text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{nom}} \rightarrow \text{S} \] (Lexical)

2. \[\lambda_{p}.(\lambda_{k}.\pi(p)(k)(\epsilon), \pi'(p) \circ _{\text{suf}-\text{im}}) : \text{Ph}_{n} \rightarrow \text{Ph}_{n-1} \] (Lexical)

3. \[\lambda_{p}.(\lambda_{k}.\pi(p)(k)(\epsilon), \pi'(p) \circ _{\text{suf}-\text{im}}) : \text{NP}_{\text{acc}} \rightarrow \text{A} \rightarrow \text{S} \] (App)

4. \[\lambda_{p}.(\lambda_{k}.\pi(p)(k)(\epsilon), \pi'(p) \circ _{\text{suf}-\text{it}}) : \text{Ph}_{n} \rightarrow \text{Ph}_{n-1} \] (Lexical)

5. \[\lambda_{p}.(\lambda_{k}.\pi(p)(k)(\epsilon), \pi'(p) \circ _{\text{suf}-\text{it}}) : \text{NP}_{\text{acc}} \rightarrow \text{NP}_{\text{nom}} \rightarrow \text{S} \] (App)

6. \[\lambda_{x}.x \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ _{\text{suf}-\text{IM}} \circ _{\text{o} \circ \epsilon} : \text{St}_{1} \] (Encap)

7. \[\text{MARThA} : \text{St} \]  
\[\text{NP}_{\text{nom}} \rightarrow \text{S} \] (Lexical)

8. \[\text{MARThA} \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ _{\text{suf}-\text{IM}} \circ _{\text{o} \circ \epsilon} : \text{St} \] (App)

8'. \[\text{MARThA} \circ \text{TOLD} \circ _{\text{suf}-\text{IM}} \circ _{\text{suf}-\text{IM}} \circ _{\text{o} \circ \epsilon} : \text{St} \] (reduction)

It must be noted that some speakers find the above to be unacceptable. If we wish to modify the grammar so as to prevent more than one pronominal affix from attaching to a verb, this can be done by creating an alternate lexical entry for the pronominal affix. The strategy here is to have the affixes themselves perform the work of encapsulation. In effect, the pronouns attach to the word, then immediately send the word into its linear position, disallowing further affixation. Next, as before, they ‘fill in’ the relevant argument position with the empty string. A sample lexical entry for the “non-stacking” version of -im follows:

\[
\lambda_{p}.(\pi'(p) \circ _{\text{suf}-\text{im}}(\epsilon) : \text{Ph}_{n} \rightarrow \text{St}_{n-1} \]  
\[\text{NP}_{\text{acc}} \rightarrow \text{A} \rightarrow \text{S} \]