The Phenogrammar of Coordination

Chris Worth
Department of Linguistics
The Ohio State University
worth@ling.osu.edu

Abstract

Linear Categorial Grammar (LinCG) is a sign-based, Curryesque, relational, logical categorial grammar (CG) whose central architecture is based on linear logic. Curryesque grammars separate the abstract combinatorics (tectogrammar) of linguistic expressions from their concrete, audible representations (phenogrammar). Most of these grammars encode linear order in string-based lambda terms, in which there is no obvious way to distinguish right from left. Without some notion of directionality, grammars are unable to differentiate, say, subject and object for purposes of building functorial coordinate structures. We introduce the notion of a phenominator as a way to encode the term structure of a functor separately from its "string support". This technology is then employed to analyze a range of coordination phenomena typically left unaddressed by Linear Logic-based Curryesque frameworks.

1 Overview

Flexibility to the notion of constituency in conjunction with introduction (and composition) rules has allowed categorial grammars to successfully address an entire host of coordination phenomena in a transparent and compositional manner. While "Curryesque" CGs as a rule do not suffer from some of the other difficulties that plague Lambek CGs, many are notably deficient in one area: coordination. Lest we throw the baby out with the bathwater, this is an issue that needs to be addressed. We take the following to be an exemplary subset of the relevant data, and adopt a fragment methodology to show how it may be analyzed.

(1) Tyrion and Joffrey drank.

(2) Joffrey whined and sniveled.

(3) Tyrion slapped and Tywin chastised Joffrey.

The first example is a straightforward instance of noun phrase coordination. The second and third are both instances of what has become known in the categorial grammar literature as "functor coordination", that is, the coordination of linguistic material that is in some way incomplete. The third is particularly noteworthy as being an example of a "right node raising" construction, whereby the argument *Joffrey* serves as the object to both of the higher NP-Verb complexes. We will show that all three examples can be given an uncomplicated account in the Curryesque framework of **Linear Categorial Grammar** (LinCG), and that (2) and (3) have more in common than not.

Section 1 provides an overview of the data and and central issues surrounding an analysis of coordination in Curryesque grammars. Section 2 introduces the reader to the framework of LinCG, and presents the technical innovations at the heart of this paper. Section 3 gives lexical entries and derivations for the examples in section 1, and section 4 discusses our results and suggests some directions for research in the near future, with references following.

1.1 Curryesque grammars and Linear Categorial Grammar

We take as our starting point the **Curryesque** (after Curry (1961)) tradition of categorial grammars, making particular reference to those originating with Oehrle (1994) and continuing with Abstract Categorial Grammar (ACG) of de Groote (2001), Muskens (2010)'s Lambda Grammar (λG), Kubota and Levine's Hybrid Type-Logical Categorial Grammar (Kubota and Levine, 2012) and to a lesser extent the Grammatical Framework of Ranta (2004), and others. These dialects

Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS), pages 28–36,
Gothenburg, Sweden, April 26-30 2014.

of categorial grammar make a distinction between **Tectogrammar**, or "abstract syntax", and **Phenogrammar**, or "concrete syntax". Tectogrammar is primarily concerned with the structural properties of grammar, among them cooccurrence, case, agreement, tense, and so forth. Phenogrammar is concerned with computing a pre-phonological representation of what will eventually be produced by the speaker, and encompasses word order, morphology, prosody, and the like.

Linear Categorial Grammar (LinCG) is a sign-based, Curryesque, relational, logical categorial grammar whose central architecture is based on linear logic. Abbreviatory overlap has been a regrettably persistent problem, and LinCG is the same in essence as the framework varyingly called Linear Grammar (LG) and Pheno-Tecto-Differentiated Categorial Grammar (PTDCG), and developed in Smith (2010), Mihalicek (2012), Martin (2013), Pollard and Smith (2012), and Pollard (2013). In LinCG, the syntax-phonology and syntax-semantics interfaces amount to noting that the logics for the phenogrammar, the tectogrammar, and the semantics operate in parallel. This stands in contrast to 'syntactocentric' theories of grammar, where syntax is taken to be the fundamental domain within which expressions combine, and then phonology and semantics are 'read off' of the syntactic representation. LinCG is conceptually different in that it has relational, rather than functional, interfaces between the three components of the grammar. Since we do not interpret syntactic types into phenogrammatical or semantic types, this allows us a great deal of freedom within each logic, although in practice we maintain a fairly tight connection between all three components. Grammar rules take the form of derivational rules which generate triples called **signs**, and they bind together the three logics so that they operate concurrently. While the invocation of a grammar rule might simply be, say, point-wise application, the ramifications for the three systems can in principle be different; one can imagine expressions which exhibit type asymmetry in various ways.

By way of example, one might think of 'focus' as an operation which has reflexes in all three aspects of the grammar: it applies pitch accents to the target string(s) in the phenogrammar (the difference between accented and unaccented words being reflected in the phenotype), it creates 'lowering' operators in the tectogrammar (that is, expressions which scope within a continuation), and it 'focuses' a particular meaningful unit in the semantics. A focused expression might share its tectotype ((NP $\multimap$ S) $\multimap$ S) with, say, a quantified noun phrase, but the two could have different phenotypes, reflecting the accentuation or lack thereof by placing the resulting expression in the domain of prosodic boundary phenomena or not. Nevertheless, the system is constrained by the fact that the tectogrammar is based on linear logic, so if we take some care when writing grammar rules, we should still find resource sensitivity to be at the heart of the framework.

1.2 Why coordination is difficult for Curryesque grammars

Most Curryesque CGs encode linear order in lambda terms, and there is no obvious way to distinguish 'right' from 'left' by examining the types (be they linear or intuitionistic).[1] This is not a problem when we are coordinating strings directly, as de Groote and Maarek (2007) show, but an analysis of the more difficult case of functor coordination remains elusive.[2] Without some notion of directionality, grammars are unable to distinguish between, say, subject and object. This would seem to predict, for example, that $\lambda s.\ s \cdot \text{SLAPPED} \cdot \text{JOFFREY}$ and $\lambda s.\ \text{TYRION} \cdot \text{SLAPPED} \cdot s$ would have the same syntactic category (NP $\multimap$ S in the tectogrammar, and St $\rightarrow$ St in the phenogrammar), and would thus be compatible under coordination, but this is generally not the case. What we need is a way to examine the structure of a lambda term independently of the specific string constants that comprise it. To put it another way, in order to coordinate functors, we need to be able to distinguish between what Oehrle (1995) calls their **string support**, that is, the string constants which make up the body of a particular functional term, and the linearization structure such functors impose on their arguments.

2 Linear Categorial Grammar (LinCG)

Curryesque grammars separate the notion of linear order from the abstract combinatorics of linguis-

[1] A noteworthy exception is Ranta's Grammatical Framework (GF), explored in, e.g. Ranta (2004) and Ranta (2009). GF also makes distinctions between tectogrammar and phenogrammar, though it has a somewhat different conception of each.

[2] A problem explicitly recognized by Kubota (2010) in section 3.2.1.

tic expressions, and as such base their tectogrammars around logics other than bilinear logic; the Grammatical Framework is based on Martin-Löf type theory, and LinCG and its cousins ACG and λG use linear logic. Linear logic is generally described as being "resource-sensitive", owing to the lack of the structural rules of weakening and contraction. Resource sensitivity is an attractive notion, theoretically, since it allows us to describe processes of resource production, consumption, and combination in a manner which is agnostic about precisely how resources are combined. Certain problems which have been historically tricky for Lambek categorial grammars (medial extraction, quantifier scope, etc.) are easily handled by LinCG.

Since a full introduction to the framework is regrettably impossible given current constraints, we refer the interested reader to the references in section 1.1, which contain a more in-depth discussion of the potential richness of the architecture of LinCG. We do not wish to say anything new about the semantics or the tectogrammar of coordination in the current discussion, so we will expend our time fleshing out the phenogrammatical component of the framework, and it is to this topic that we now turn.

2.1 LinCG Phenogrammar

LinCG grammar rules take the form of tripartite inference rules, indicating what operations take place pointwise within each component of the signs in question. There are two main grammar rules, called **application** (App) for combining signs, and **abstraction** (Abs) for creating the potential for combination through hypothetical reasoning. Aside from the lexical entries given as axioms of the theory, it is also possible to obtain typed variables using the rule of **axiom** (Ax), and we make use of this rule in the analysis of right node raising found in section 3.4. While the tectogrammar of LinCG is based on a fragment of linear logic, the phenogrammatical and semantic components are based on higher order logic. Since we are concerned only with the phenogrammatical component here, we have chosen to simplify the exposition by presenting only the phenogrammatical part of the rules of application and abstraction:

$$\frac{}{f : A \vdash f : A}\ \text{Ax}$$

$$\frac{\Gamma \vdash f : A \to B \qquad \Delta \vdash a : A}{\Gamma, \Delta \vdash (f\ a) : B}\ \text{App}$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x : A.\ b : A \to B}\ \text{Abs}$$

We additionally stipulate the following familiar axioms governing the conversion and reduction of lambda terms:[3]

$\vdash \lambda x : A.\ b = \lambda y : A.\ [y/x]b$ (α-conversion)
$\vdash (\lambda x.\ b\ a) = [a/x]b$ (β-reduction)

As is common to any number of Curryesque frameworks, we encode the phenogrammatical parts of LinCG signs with typed lambda terms consisting of strings, and functions over strings.[4] We axiomatize our theory of strings in the familiar way:

$$\vdash \epsilon : \text{St}$$
$$\vdash \cdot : \text{St} \to \text{St} \to \text{St}$$
$$\vdash \forall stu : \text{St}.\ s \cdot (t \cdot u) = (s \cdot t) \cdot u$$
$$\vdash \forall s : \text{St}.\ \epsilon \cdot s = s = s \cdot \epsilon$$

The first axiom asserts that the empty string ϵ is a string. The second axiom asserts that concatenation, written $\cdot$, is a (curried) binary function on strings. The third axiom represents the fact that concatenation is associative, and the fourth, that the empty string is a two-sided identity for concatenation. Because of the associativity of concatenation, we will drop parentheses as a matter of convention.

The phenogrammar of a typical LinCG sign will resemble the following (with one complication to be added shortly):

$$\vdash \lambda s.\ s \cdot \text{SNIVELED} : \text{St} \to \text{St}$$

Since we treat **St** as the only base type, we will generally omit typing judgments in lambda terms when no confusion will result. Furthermore, we use SMALL CAPS to indicate that a particular constant is a string. So, the preceding lexical entry provides us with a function from some string s, to strings, which concatenates the string SNIVELED to the right of s.

2.1.1 Phenominators

The center of our analysis of coordination is the notion of a **phenominator** (short for phenocombinator), a particular variety of typed lambda term. Intuitively, phenominators serve the same purpose for LinCG that bilinear (slash) types do for Lambek categorial grammars. Specifically,

[3] The exact status of the rule of η-conversion with respect to this framework is currently unclear, and since we do not make use of it, we omit its discussion here.

[4] although other structures have been proposed, e.g. the node sets found in Muskens (2001).

they encode the linearization structure of a functor, that is, where arguments may eventually occur with respect to its string support. To put it another way, a phenominator describes the structure a functor "projects", in terms of linear order.

From a technical standpoint, we would like to define a phenominator as a closed monoidal linear lambda term, i.e. a term containing no constants other than concatenation and the empty string. The idea is that phenominators are the terms of the higher order theory of monoids, and they in some ways describe the abstract "shape" of possible string functions. For those accustomed to thinking of "syntax" as being word order, then phenominators can be thought of as a kind of syntactic combinator. In practice, we will make use only of what we call the unary phenominators, the types of which we will refer to using the sort Φ (with φ used by custom as a metavariable over unary phenominators, i.e. terms whose type is in Φ). These are not unary in the strict sense, but they will have as their centerpiece one particular string variable, which will be bound with the highest scope. We will generally abbreviate phenominators by the construction with which they are most commonly associated: **VP** for verb phrases and intransitive verbs, **TV** for transitive verbs, **DTV** for ditransitive verbs, **QNP** for quantified noun phrases, and **RNR** for right node raising constructions. Here are examples of some of the most common phenominators we will make use of and the abbreviations we customarily use for them:

Phenominator	**Abbreviation**
$\lambda s.s$	(omitted)
$\lambda vs.s \cdot v$	VP
$\lambda vst.t \cdot v \cdot s$	TV
$\lambda vstu.u \cdot v \cdot s \cdot t$	DTV
$\lambda vP.(P\ v)$	QNP
$\lambda vs.v \cdot s$	RNR

As indicated previously, the first argument of a phenominator always corresponds to what we refer to (after Oehrle (1995)) as the string support of a particular term. With the first argument dispensed with, we have chosen the argument order of the phenominators out of general concern for what we perceive to be fairly uncontroversial categorial analyses of English grammatical phenomena. That is, transitive verbs take their object arguments first, and then their subject arguments, ditransitives take their first and second object arguments, followed by their subject argument, etc. As long as the arguments in question are immediately adjacent to the string support at each successive application, it is possible to permute them to some extent without losing the general thrust of the analysis. For example, the choice to have transitive verbs take their object arguments first is insignificant.[5] Since strings are implicitly under the image of the identity phenominator $\lambda s.s$, we will consistently omit this subscript.

We will be able to define a function we call `say`, so that it will have the following property:

$$\vdash \forall s : \mathrm{St}.\forall \varphi : \Phi.\mathtt{say}\ (\varphi\ s) = s$$

That is, `say` is a left inverse for unary phenominators.

The function `say` is defined recursively via certain objects we call **vacuities**. The idea of a vacuity is that it be in some way an "empty argument" to which a functional term may apply. If we are dealing with functions taking string arguments, it seems obvious that the vacuity on strings should be the empty string ϵ. If we are dealing with second-order functions taking $\mathrm{St} \rightarrow \mathrm{St}$ arguments, for example, quantified noun phrases like *everyone*, then the vacuity on $\mathrm{St} \rightarrow \mathrm{St}$ should be the identity function on strings, $\lambda s.s$. Higher vacuities than these become more complicated, and defining all of the higher-order vacuities is not entirely straightforward, as certain types are not guaranteed to have a unique vacuity. Fortunately, we can do it for any higher-function taking as an argument another function under the image of a phenominator – then the vacuity on such a function is just the phenominator applied to the empty string.[6] The central idea is easily understood when one asks what, say, a vacuous transitive verb sounds like. The answer seems to be: by itself, nothing, but it imposes a certain order on its arguments. One practical application of this clause is in analyzing so-called "argument cluster coordination", where this definition will ensure that the argument cluster gets linearized in the correct manner. This analysis is regrettably just outside the scope of the current inquiry, though the notion of the phenomina-

[5] Since we believe it is possible to embed Lambek categorial grammars in LinCG, this fact reflects that the calculus we are dealing with is similar to the **associative** Lambek Calculus.

[6] A reviewer suggests that this concept may be related to the "context passing representation" of Hughes (1995), and the association of a *nil* term with its continuation with respect to contexts is assuredly evocative of the association of the vacuity on a phenominator-indexed type with the continuation of ϵ with respect to a phenominator.

tor can be profitably employed to provide exactly such an analysis by adopting and reinterpreting a categorial account along the lines of the one given in Dowty (1988).

We formally define vacuities as follows:

$$\texttt{vac}_{\text{St}\to\text{St}} =_{def} \lambda s.s$$
$$\texttt{vac}_{\tau_\varphi} =_{def} (\varphi\ \epsilon)$$

The reader should note that as a special case of the second clause, we have

$$\texttt{vac}_{\text{St}} = \texttt{vac}_{\text{St}_{\lambda s.s}} = (\lambda s.s\ \epsilon) = \epsilon$$

This in turn enables us to define say:

$$\texttt{say}_{\text{St}} =_{def} \lambda s.s$$
$$\texttt{say}_{\tau_1\to\tau_2} =_{def} \lambda k : \tau_1 \to \tau_2.\ \texttt{say}_{\tau_2}(k\ \texttt{vac}_{\tau_1})$$
$$\texttt{say}_{(\tau_1\to\tau_2)_\varphi} =_{def} \texttt{say}_{\tau_1\to\tau_2}$$

For an expedient example, we can apply say to our putative lexical entry from earlier, and verify that it will reduce to the string SNIVELED as desired:

$$\begin{aligned}
&\texttt{say}_{\text{St}\to\text{St}}\ \lambda s.\ s\cdot\text{SNIVELED}\\
&=\lambda k : \text{St}\to\text{St}.\\
&(\texttt{say}_{\text{St}}(k\ \texttt{vac}_{\text{St}}))\ \lambda s.\ s\cdot\text{SNIVELED}\\
&=\texttt{say}_{\text{St}}\ (\lambda s.\ s\cdot\text{SNIVELED}\ \texttt{vac}_{\text{St}})\\
&=\texttt{say}_{\text{St}}\ (\lambda s.\ s\cdot\text{SNIVELED}\ \epsilon)\\
&=\texttt{say}_{\text{St}}\ \epsilon\cdot\text{SNIVELED}\\
&=\texttt{say}_{\text{St}}\ \text{SNIVELED}\\
&=\lambda s.s\ \text{SNIVELED}\\
&=\text{SNIVELED}
\end{aligned}$$

2.1.2 Subtyping by unary phenominators

In order to augment our type theory with the relevant subtypes, we turn to Lambek and Scott (1986), who hold that one way to do subtyping is by defining predicates that amount to the characteristic function of the particular subtype in question, and then ensuring that these predicates meet certain axioms embedding the subtype into the supertype. We will be able to write such predicates using phenominators. A **unary** phenominator is one which has under its image a function whose string support is a single contiguous string. With this idea in place, we are able to assign subtypes to functional types in the following way.

For τ a (functional) type, we write τ_φ (with φ a phenominator) as shorthand for $\tau_{\varphi'}$, where:

$$\varphi' = \lambda f : \tau.\ \exists s : \text{St}.f = (\varphi\ s)$$

Then φ' constitutes a subtyping predicate in the manner of Lambek and Scott (1986). For example, let $\tau = \text{St}\to\text{St}$ and $\varphi = \lambda vs.s\cdot v$. Let us consider the following (putative) lexical entry (pheno only):

$$\vdash \lambda s'.\ s'\cdot\text{SNIVELED} : (\text{St}\to\text{St})_{\text{VP}}$$

Then our typing is justified along the following lines:

$$\begin{aligned}
\tau_\varphi\ &::=\ (\text{St}\to\text{St})_{\text{VP}}\\
&::=\ (\text{St}\to\text{St})_{\lambda vs.s\cdot v}\\
&::=\ (\text{St}\to\text{St})_{\lambda f:\text{St}\to\text{St}.\ \exists t:\text{St}.f=(\lambda vs.s\cdot v\ t)}
\end{aligned}$$

So applying the subtyping predicate to the term in question, we have

$$\begin{aligned}
&(\lambda f : \text{St}\to\text{St}.\ \exists t : \text{St}.\\
&f = (\lambda vs.s\cdot v\ t)\ \lambda s'.\ s'\cdot\text{SNIVELED})\\
&=\exists t : \text{St}.\lambda s'.\ s'\cdot\text{SNIVELED} = (\lambda vs.s\cdot v\ t)\\
&=\exists t : \text{St}.\lambda s'.\ s'\cdot\text{SNIVELED} = \lambda s.\ s\cdot t\\
&=\exists t : \text{St}.\lambda s.\ s\cdot\text{SNIVELED} = \lambda s.\ s\cdot t
\end{aligned}$$

which is true with $t = \text{SNIVELED}$, and the term is shown to be well-typed.

3 Analysis

The basic strategy underlying our analysis of coordination is that in order to coordinate two linguistic signs, we need to track two things: their linearization structure, and their string support. If we have access to the linearization structure of each conjunct, then we can check to see that it is the same, and the signs are compatible for coordination. Furthermore, we will be able to maintain this structure independent of the actual string support of the individual signs.

Phenominators simultaneously allow us to check the linearization structure of coordination candidates and to reconstruct the relevant linearization functions after coordination has taken place. The function say addresses the second point. For a given sign, we can apply say to it in order to retrieve its string support. Then, we will be able to directly coordinate the resulting strings by concatenating them with a conjunction in between. Finally, we can apply the phenominator to the resulting string and retrieve the new linearization function, containing the entire coordinate structure as its string support.

3.1 Lexical entries

In LinCG, lexical entries constitute the (nonlogical) axioms of the proof theory. First we consider the simplest elements of our fragment, the phenos for the proper names *Joffrey*, *Tyrion*, and *Tywin*:

(4) a. $\vdash$ JOFFREY : St
b. $\vdash$ TYRION : St
c. $\vdash$ TYWIN : St

Next, we consider the intransitive verbs *drank*, *sniveled* and *whined*. :

(5) a. $\vdash \lambda s.\ s \cdot \text{DRANK} : (\text{St} \to \text{St})_{\text{VP}}$
b. $\vdash \lambda s.\ s \cdot \text{SNIVELED} : (\text{St} \to \text{St})_{\text{VP}}$
c. $\vdash \lambda s.\ s \cdot \text{WHINED} : (\text{St} \to \text{St})_{\text{VP}}$

Each of these is a function from strings to strings, seeking to linearize its 'subject' string argument to the left of the verb. They are under the image of the "verb phrase" phenominator $\lambda vs.s \cdot v$.

The transitive verbs *chastised* and *slapped* seek to linearize their first string argument to the right, resulting in a function under the image of the VP phenominator, and their second argument to the left, resulting in a string.

(6) a. $\vdash \lambda st.\ t \cdot \text{CHASTISED} \cdot s$
$: (\text{St} \to \text{St} \to \text{St})_{\text{TV}}$
b. $\vdash \lambda st.\ t \cdot \text{SLAPPED} \cdot s$
$: (\text{St} \to \text{St} \to \text{St})_{\text{TV}}$

Technically, this type could be written $(\text{St} \to (\text{St} \to \text{St})_{\text{VP}})_{\text{TV}}$, but for the purposes of coordination, the present is sufficient. Each of these entries is under the image of the "transitive verb" phenominator $\lambda vst.t \cdot v \cdot s$.

Finally, we come to the lexical entry schema for *and*:

(7) $\vdash \lambda c_1 : \tau_\varphi.\ \lambda c_2 : \tau_\varphi.$
$\varphi\ ((\texttt{say}_{\tau_\varphi}\ c_2) \cdot \text{AND} \cdot (\texttt{say}_{\tau_\varphi}\ c_1))$
$: \tau_\varphi \to \tau_\varphi \to \tau_\varphi$

We note first that it takes two arguments of identical types τ, and furthermore that these must be under the image of the same phenominator φ. It then returns an expression of the same subtype.[7] This mechanism bears more detailed examination. First, each conjunct is subjected to the function `say`, which, given its type, will return the string support of the conjunct. Then, the resulting strings are concatenated to either side of the string AND. Finally, the phenominator of each argument is applied to the resulting string, creating a function identical to the linearization functions of each of the conjuncts, except with the coordinated string in the relevant position.

3.2 String coordination

String coordination is direct and straightforward. Since string-typed terms are under the image of the identity phenominator, and since $\texttt{say}_{\text{St}}$ is also defined to be the identity on strings, the lexical entry we obtain for *and* simply concatenates each argument string to either side of the string AND. We give the full term reduction here, although this version of *and* can be shown to be equal to the following:

$\vdash \lambda c_1 c_2 : \text{St}.\ c_2 \cdot \text{AND} \cdot c_1 : \text{St} \to \text{St} \to \text{St}$

Since our terms at times become rather large, we will adopt a convention where proof trees are given with numerical indexes instead of sequents, with the corresponding sequents following below (at times on multiple lines). We will from time to time elide multiple steps of reduction, noting in passing the relevant definitions to consider when reconstructing the proof.

$$\dfrac{\dfrac{\dfrac{1 \quad 2}{3} \quad 4}{6 \quad 5}}{7}$$

1. $\vdash \lambda c_1 : \text{St}.\ \lambda c_2 : \text{St}.$
$\lambda s.s\ ((\texttt{say}_{\text{St}}\ c_2) \cdot \text{AND} \cdot (\texttt{say}_{\text{St}}\ c_1))$
$: \text{St} \to \text{St} \to \text{St}$
2. $\vdash \text{JOFFREY} : \text{St}$
3. $\vdash \lambda c_2 : \text{St}.$
$\lambda s.s\ ((\texttt{say}_{\text{St}}\ c_2) \cdot \text{AND} \cdot (\texttt{say}_{\text{St}}\ \text{JOFFREY}))$
$= \lambda c_2 : \text{St}.$
$\lambda s.s\ ((\texttt{say}_{\text{St}}\ c_2) \cdot \text{AND} \cdot (\lambda s.s\ \text{JOFFREY}))$
$= \lambda c_2 : \text{St}.\ \lambda s.s\ ((\texttt{say}_{\text{St}}\ c_2) \cdot \text{AND} \cdot \text{JOFFREY})$
$: \text{St} \to \text{St}$
4. $\vdash \text{TYRION} : \text{St}$
5. $\vdash \lambda s.s\ ((\texttt{say}_{\text{St}}\ \text{TYRION}) \cdot \text{AND} \cdot \text{JOFFREY}) :$
St
$= \lambda s.s\ ((\lambda s.s\ \text{TYRION}) \cdot \text{AND} \cdot \text{JOFFREY}) : \text{St}$
$= \lambda s.s\ (\text{TYRION} \cdot \text{AND} \cdot \text{JOFFREY}) : \text{St}$
$= \text{TYRION} \cdot \text{AND} \cdot \text{JOFFREY} : \text{St}$
6. $\vdash \lambda s.\ s \cdot \text{DRANK} : (\text{St} \to \text{St})_{\text{VP}}$
7. $\vdash \text{TYRION} \cdot \text{AND} \cdot \text{JOFFREY} \cdot \text{DRANK} : \text{St}$

3.3 Functor coordination

Here, in order to understand the term appearing in each conjunct, it is helpful to notice that the following equality holds (with f a function from strings to strings, under the image of the VP phenominator):

$f : (\text{St} \to \text{St})_{\text{VP}} \vdash \texttt{say}_{(\text{St} \to \text{St})_{\text{VP}}}\ f$
$= \texttt{say}_{\text{St} \to \text{St}}\ f$
$= \texttt{say}_{\text{St}}\ (f\ \texttt{vac}_{\text{St}})$
$= \texttt{say}_{\text{St}}\ (f\ \epsilon)$
$= \lambda s.s\ (f\ \epsilon)$
$= (f\ \epsilon) : \text{St}$

[7] Since φ occurs within both the body of the term and the subtyping predicate, we note that this effectively takes us into the realm of dependent types. Making the type theory of the phenogrammar precise is an ongoing area of research, and we are aware that constraining the type system is of paramount importance for computational tractability.

This says that to coordinate VPs, we will first need to reduce them to their string support by feeding their linearization functions the empty string. For the sake of brevity, this term reduction will be elided from steps 5 and 8 in the derivations below. Steps 2 and 6 constitute the hypothesizing and subsequent withdrawal of an 'object' string argument t', as do steps 10 and 14 (s'). Formatting restrictions prohibit rule-labeling on the proof trees, so we note that these are each instances of the rules of axiom (Ax) and abstraction (Abs), respectively.

$$\dfrac{\dfrac{\dfrac{1 \quad 2}{3} \quad 4}{5} \quad 6}{7}$$

1. $\vdash \lambda c_1 : (\text{St} \rightarrow \text{St})_{\text{VP}}.\ \lambda c_2 : (\text{St} \rightarrow \text{St})_{\text{VP}}.$
   $\lambda vs.s \cdot v$
   $((\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{VP}}}\ c_2) \cdot \text{AND} \cdot (\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{VP}}}\ c_1))$
   $: (\text{St} \rightarrow \text{St})_{\text{VP}} \rightarrow (\text{St} \rightarrow \text{St})_{\text{VP}} \rightarrow (\text{St} \rightarrow \text{St})_{\text{VP}}$
2. $\vdash \lambda s.\ s \cdot \text{SNIVELED} : (\text{St} \rightarrow \text{St})_{\text{VP}}$
3. $\vdash \lambda c_2 : (\text{St} \rightarrow \text{St})_{\text{VP}}.\ \lambda vs.s \cdot v$
   $((\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{VP}}}\ c_2) \cdot \text{AND}$
   $\cdot\ (\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{VP}}}\ \lambda s.\ s \cdot \text{SNIVELED}))$
   $\vdots$
   $= \lambda c_2 : (\text{St} \rightarrow \text{St})_{\text{VP}}.\ \lambda vs.s \cdot v$
   $((\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{VP}}}\ c_2) \cdot \text{AND} \cdot \text{SNIVELED})$
   $: (\text{St} \rightarrow \text{St})_{\text{VP}} \rightarrow (\text{St} \rightarrow \text{St})_{\text{VP}}$
4. $\vdash \lambda s.\ s \cdot \text{WHINED} : (\text{St} \rightarrow \text{St})_{\text{VP}}$
5. $\vdash \lambda vs.s \cdot v\ ((\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{VP}}}\ \lambda s.\ s \cdot \text{WHINED})$
   $\cdot\ \text{AND} \cdot \text{SNIVELED})$
   $\vdots$
   $= (\lambda vs.s \cdot v\ \text{WHINED} \cdot \text{AND} \cdot \text{SNIVELED})$
   $= \lambda s.\ s \cdot \text{WHINED} \cdot \text{AND} \cdot \text{SNIVELED}$
   $: (\text{St} \rightarrow \text{St})_{\text{VP}}$
6. $\vdash \text{JOFFREY} : \text{St}$
7. $\vdash \text{JOFFREY} \cdot \text{WHINED} \cdot \text{AND} \cdot \text{SNIVELED} : \text{St}$

3.4 Right node raising

In the end, 'right node raising' constructions prove only to be a special case of functor coordination. The key here is the licensing of the 'rightward-looking' functors, which are under the image of the phenominator $\lambda vs.v \cdot s$. As was the case with the 'leftward-looking' functor coordination example in section 3.3, this analysis is essentially the same as the well-known Lambek categorial grammar analysis originating in Steedman (1985) and continuing in Dowty (1988) and Morrill (1994). The difference is that we encode directionality in the phenominator, rather than in the type. Since our system does not include function composition as a rule, but as a theorem, we will need to make use of hypothetical reasoning in order to permute the order of the string arguments in order to construct expressions with the correct structure.[8]

As was the case with the functor coordination example in section 3.3, applying say to the conjuncts passes them the empty string, reducing them to their string support, as shown here:

$$\begin{aligned}
&f : (\text{St} \rightarrow \text{St})_{\text{RNR}} \vdash \texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{RNR}}}\ f \\
&= \texttt{say}_{\text{St}\rightarrow\text{St}}\ f \\
&= \texttt{say}_{\text{St}}\ (f\ \texttt{vac}_{\text{St}}) \\
&= \texttt{say}_{\text{St}}\ (f\ \epsilon) \\
&= \lambda s.s\ (f\ \epsilon) \\
&= (f\ \epsilon) : \text{St}
\end{aligned}$$

As before, this reduction is elided in the proof given below, occurring in steps 8 and 15.

$$\dfrac{\dfrac{\dfrac{\dfrac{1 \quad \overline{2}}{3} \quad 4}{5} \quad \dfrac{\dfrac{9 \quad \overline{10}}{11} \quad 12}{13}}{6} \quad 7}{\dfrac{8 \qquad\qquad 14}{15} \qquad 16}{17}$$

1. $\vdash \lambda st.\ t \cdot \text{CHASTISED} \cdot s : (\text{St} \rightarrow \text{St} \rightarrow \text{St})_{\text{TV}}$
2. $t' : \text{St} \vdash t' : \text{St}$
3. $t' : \text{St} \vdash \lambda t.\ t \cdot \text{CHASTISED} \cdot t' : (\text{St} \rightarrow \text{St})_{\text{VP}}$
4. $\vdash \text{TYWIN} : \text{St}$
5. $t' : \text{St} \vdash \text{TYWIN} \cdot \text{CHASTISED} \cdot t' : \text{St}$
6. $\vdash \lambda t'.\ \text{TYWIN} \cdot \text{CHASTISED} \cdot t' : (\text{St} \rightarrow \text{St})_{\text{RNR}}$
7. $\vdash \lambda c_1 : (\text{St} \rightarrow \text{St})_{\text{RNR}}.\ \lambda c_2 : (\text{St} \rightarrow \text{St})_{\text{RNR}}.$
   $\lambda vs.v \cdot s\ ((\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{RNR}}}\ c_2)$
   $\cdot\ \text{AND} \cdot (\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{RNR}}}\ c_1))$
   $: (\text{St} \rightarrow \text{St})_{\text{RNR}} \rightarrow (\text{St} \rightarrow \text{St})_{\text{RNR}}$
   $\rightarrow (\text{St} \rightarrow \text{St})_{\text{RNR}}$
8. $\vdash \lambda c_2 : (\text{St} \rightarrow \text{St})_{\text{RNR}}.\ \lambda vs.v \cdot s$
   $((\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{RNR}}}\ c_2) \cdot \text{AND}$
   $\cdot (\texttt{say}_{(\text{St}\rightarrow\text{St})_{\text{RNR}}}\ \lambda t'.\ \text{TYWIN} \cdot \text{CHASTISED} \cdot t'))$
   $\vdots$
   $= \lambda c_2 : (\text{St} \rightarrow \text{St})_{\text{RNR}}.\ \lambda vs.v \cdot s$

[8] Regrettably, space constraints prohibit a discussion verifying the typing for the 'right node raised' terms. The reader can verify that the terms are in fact well-typed given the subtyping schema in section 2.1.2. It is possible to write inference rules that speak directly to the introduction and elimination of the relevant functional subtypes, but these are omitted here for the sake of brevity.

$((\texttt{say}_{(\text{St}\to\text{St})_{\text{RNR}}}\ c_2)$
$\cdot$ AND $\cdot$ TYWIN $\cdot$ CHASTISED$)$
$: (\text{St} \to \text{St})_{\text{RNR}} \to (\text{St} \to \text{St})_{\text{RNR}}$

9. $\vdash \lambda st.\ t \cdot$ SLAPPED $\cdot\ s : (\text{St} \to \text{St} \to \text{St})_{\text{TV}}$
10. $s' : \text{St} \vdash s' : \text{St}$
11. $s' : \text{St} \vdash \lambda t.\ t \cdot$ SLAPPED $\cdot\ s' : (\text{St} \to \text{St})_{\text{VP}}$
12. $\vdash$ TYRION $: \text{St}$
13. $s' : \text{St} \vdash$ TYRION $\cdot$ SLAPPED $\cdot\ s' : \text{St}$
14. $\vdash \lambda s'.$ TYRION $\cdot$ SLAPPED $\cdot\ s' : (\text{St} \to \text{St})_{\text{RNR}}$
15. $\vdash \lambda vs.v \cdot s\ ((\texttt{say}_{(\text{St}\to\text{St})_{\text{RNR}}}$
$\lambda s'.$ TYRION $\cdot$ SLAPPED $\cdot\ s')$
$\cdot$ AND $\cdot$ TYWIN $\cdot$ CHASTISED$)$
$\vdots$
$= (\lambda vs.v \cdot s$ TYRION $\cdot$ SLAPPED
$\cdot$ AND $\cdot$ TYWIN $\cdot$ CHASTISED$)$
$= \lambda s.$ TYRION $\cdot$ SLAPPED $\cdot$ AND
$\cdot$ TYWIN $\cdot$ CHASTISED $\cdot\ s : (\text{St} \to \text{St})_{\text{RNR}}$
16. $\vdash$ JOFFREY $: \text{St}$
17. $\vdash$ TYRION $\cdot$ SLAPPED $\cdot$ AND
$\cdot$ TYWIN $\cdot$ CHASTISED $\cdot$ JOFFREY $: \text{St}$

4 Discussion

We provide a brief introduction to the framework of Linear Categorial Grammar (LinCG). One of the primary strengths of categorial grammar in general has been its ability to address coordination phenomena. Coordination presents a uniquely particular problem for grammars which distinguish between structural combination (tectogrammar) and the actual linear order of the strings generated by such grammars (part of phenogrammar). Due to the inability to distinguish 'directionality' in string functors within a standard typed lambda calculus, a general analysis of coordination seems difficult.

We have elaborated LinCG's concept of phenogrammar by introducing phenominators, closed monoidal linear lambda terms. We have shown how the recursive function `say` provides a left inverse for unaryphenominators, and we have defined a more general notion of an 'empty category' known as a vacuity, which `say` is defined in terms of. It is then possible to describe subtypes of functional types suitable to make the relevant distinctions. These technologies enable us to give analyses of various coordination phenomena in LinCG, extending the empirical coverage of the framework.

4.1 Future work

It is possible to give an analysis of argument cluster coordination using phenominators, instantiating the lexical entry for *and* with τ as the type $(\text{St} \to \text{St} \to \text{St} \to \text{St})_{\text{DTV}} \to (\text{St} \to \text{St})_{\text{VP}}$ and φ as $\lambda vPs.\ s \cdot (P\epsilon\epsilon\epsilon) \cdot v$, and using hypothetical reasoning. Regrettably, the necessity of brevity prohibits a detailed account here.

Given that phenominators provide access to the structure of functional terms which concatenate strings to the right and left of their string support, it is our belief that any Lambek categorial grammar analysis can be recast in LinCG by an algorithmic translation of directional slash types into phenominator-indexed functional phenotypes, and we are currently in the process of evaluating a potential translation algorithm from directional slash types to phenominators. This should in turn provide us with most of the details necessary to describe a system which emulates the HTLCG of Kubota and Levine (2012), which provides analyses of various gapping phenomena, greatly increasing the overall empirical coverage.

There are a number of coordination phenomena that require modifications to the tectogrammatical component. We would like to be able to analyze unlike category coordinations like *rich and an excellent cook* in the manner of Bayer (1996), as well as Morrill (1996), which would require the addition of some variety of sum types in the tectogrammar. Further muddying the waters is so-called "iterated" or "list" coordination, which requires the ability to generate coordinate structures containing a number of conjuncts with no coordinating conjunction, as in *Thurston, Kim, and Steve*.

It is our intent to extend the use of phenominators to analyze intonation as well, and we expect that they can be fruitfully employed to give accounts of focus, association with focus, contrastive topicalization, "in-situ" topicalization, alternative questions, and any number of other phenomena which are at least partially realized prosodically.

Acknowledgements

I am grateful to Carl Pollard, Bob Levine, Yusuke Kubota, Manjuan Duan, Gerald Penn, the TTNLS 2014 committee, and two anonymous reviewers for their comments. Any errors or misunderstandings rest solely on the shoulders of the author.

References

Samuel Bayer. 1996. The coordination of unlike categories. *Language*, pages 579–616.

Haskell B. Curry. 1961. Some Logical Aspects of Grammatical Structure. In Roman. O. Jakobson, editor, *Structure of Language and its Mathematical Aspects*, pages 56–68. American Mathematical Society.

Philippe de Groote and Sarah Maarek. 2007. Type-theoretic Extensions of Abstract Categorial Grammars. In Reinhard Muskens, editor, *Proceedings of Workshop on New Directions in Type-Theoretic Grammars*.

Philippe de Groote. 2001. Towards Abstract Categorial Grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155.

David Dowty. 1988. Type raising, functional composition, and non-constituent conjunction. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*, volume 32 of *Studies in Linguistics and Philosophy*, pages 153–197. Springer Netherlands.

John Hughes. 1995. The design of a pretty-printing library. In *Advanced Functional Programming*, pages 53–96. Springer.

Yusuke Kubota and Robert Levine. 2012. Gapping as like-category coordination. In D. Béchet and A. Dikovsky, editors, *Logical Aspects of Computational Linguistics (LACL) 2012*.

Yusuke Kubota. 2010. *(In)flexibility of Constituency in Japanese in Multi-Modal Categorial Grammar with Structured Phonology*. Ph.D. thesis, The Ohio State University.

J. Lambek and P.J. Scott. 1986. *Introduction to higher order categorical logic*. Cambridge University Press.

Scott Martin. 2013. *The Dynamics of Sense and Implicature*. Ph.D. thesis, The Ohio State University.

Vedrana Mihalicek. 2012. *Serbo-Croatian Word Order: A Logical Approach*. Ph.D. thesis, The Ohio State University.

Glyn V. Morrill. 1994. *Type Logical Grammar*. Kluwer Academic Publishers.

Glyn Morrill. 1996. Grammar and logic*. *Theoria*, 62(3):260–293.

Reinhard Muskens. 2001. Lambda grammars and the syntax-semantics interface. In *Proceedings of the Thirteenth Amsterdam Colloquium*, pages 150–155. Universiteit van Amsterdam.

Reinhard Muskens. 2010. New Directions in Type-Theoretic Grammars. *Journal of Logic, Language and Information*, 19(2):129–136. DOI 10.1007/s10849-009-9114-9.

Richard Oehrle. 1994. Term-Labeled Categorial Type Systems. *Linguistics and Philosophy*, 17:633–678.

Dick Oehrle. 1995. Some 3-dimensional systems of labelled deduction. *Logic Journal of the IGPL*, 3(2-3):429–448.

Carl Pollard and E. Allyn Smith. 2012. A unified analysis of *the same*, phrasal comparatives, and superlatives. In Anca Chereches, editor, *Proceedings of SALT 22*, pages 307–325. eLanguage.

Carl Pollard. 2013. Agnostic Hyperintensional Semantics. *Synthese*. to appear.

Aarne Ranta. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. *Journal of Functional Programming*, 14:145–189.

Aarne Ranta. 2009. The GF resource grammar library. *Linguistic Issues in Language Technology*, 2(1).

E. Allyn Smith. 2010. *Correlational Comparison in English*. Ph.D. thesis, The Ohio State University.

Mark Steedman. 1985. Dependency and coördination in the grammar of dutch and english. *Language*, pages 523–568.