Proof-Theoretic Background for Linear Grammar

Carl Pollard
The Ohio State University

July 4, 2011

1 Introduction

Linear grammar (hereafter LG) is a practical framework for linguistic analysis closely similar to frameworks such as abstract categorial grammar (ACG, de Groote 2001) and lambda grammar (Muskens 2003, 2007), which in turn are inspired by some programmatic ideas of Curry (1961) and some technical proposals due to Oehrle (1994). A LG for a fragment of a particular natural language (NL) recursively defines a set of ordered triples called **signs**, each of which is taken to represent one of the expressions of the NL in question. Some of these triples, called **lexical entries**, represent the (syntactic) words of the NL in question. The set of lexical entries, called the **lexicon** is given 'in advance' and forms the base of the recursive definition. The recursive part of the definition is in the form of a few (at first, just two) **rules**, with whose help we can analyze more complex expressions. As we soon will see, the lexical entries and rules can be thought of, respectively, as axioms and inference rules of a natural-deduction proof system, and the expressions analyzed by the grammar can be thought of as the theorems proved by the system.

The three components of LG signs are called the **phenogrammatical**, the **tectogrammatical**, and the **semantic** components. These can be described roughly as follows:

- The phenogrammatical component, usually abbreviated to **pheno**, is the expression's contribution to (audible or visible) surface form, including (at minimum) the phonologies of the words, and their linear order. Pheno in LG corresponds to what is called the **concrete syntax** in ACG, and simply **syntax** in lambda grammar (although from a lingust's point of view it seems more closely related to phonology than to syntax).

- The tectogrammatical component, usually abbreviated to **tecto**, is the expression's syntactic combinatory potential: what it combines with and what results from the combination. Here, as in other kinds of categorial grammar (such as combinatory categorial grammar (CCG) and type-logical grammar (TLG)), it is assumed that syntactic combination drives semantic composition (in a sense that will be made precise). Tecto in LG corresponds to what is called the **abstract syntax** in ACG, and **combinatorics** in lambda grammar.
- The semantic component is the expression's meaning. This could be a **intension**, as in Montague's (1974) PTQ, though we will use as meanings things called **hyperintensions** (Pollard 2008), which are more fine-grained than intensions. (Later, we'll also consider **dynamic** semantics, in which a sentences don't merely express propositions, but also affect the utterance context.)

Signs are notated in the form

$$\vdash \phi; \tau; \sigma$$

where:

- ϕ is a term in a typed lambda calculus (more precisely, a higher-order logical theory) called the **pheno calculus**. Such a term, called a **pheno term**, is interpreted (in the simplest case) as a string of phonological words, i or (in more complex cases) as a higher-order function over such strings.
- τ is a formula of linear logic, the **tecto type** of the sign. Intuitively, a tecto type can be thought of as the name of a syntactic category.
- σ is a term in a typed lambda calculus (again, more precisely, a higher-order theory) called the **semantic calculus**. Such a term is interpreted (in the simplest case) as an entity or a proposition, or (in more complex cases) as a higher-order function over such thing.

Here's a typical lexical entry:

$$\vdash \text{PIG}; \text{N}; \mathsf{pig}$$

In this lexical entry:

- PIG is a constant (of type s) of the pheno calculus, which denotes a string (of length 1) whose only element is the phonological word pIg/.
- N is a (basic) tectotype, namely the type of nouns.
- pig is a constant (of type e $\rightarrow$ p) of the semantic calculus, which denotes a certain property, namely the function that maps any entity to the proposition that that entity is a pig.

In addition to lexical entries, which will play the role of **nonlogical axioms** when we think of a grammar as a proof system, there are also **logical axioms**, which correspond to the notion of 'movement traces' in mainstream generative grammar. (hereafter MGG). Here's a typical trace (more specifically, an NP-trace):

$$s; \mathrm{NP}; x \vdash s; \mathrm{NP}; x$$

Here:

- s is a variable (of type s) of the pheno calculus, which denotes a hypothetical string,
- NP is a (basic) tectotype, namely the type of noun phrases, and
- x is a variable (of type e) of the semantic calculus, which denotes a hypothetical entity.

Traces will play a central role in the analysis of scope of quantificational NPs, and also in the analysis of constructions usually analyzed in terms of movement in MGG (such as constituent questions, relative clauses, and clefts). Intuitively, a trace can be thought of as a 'hypothetical' syntactic consituent.

In its simplest form, LG has only two (schematic) rules for for deducing 'new' signs from 'old' ones. In proof-theoretic terms, these two schemas correspond to the inference rules Modus Ponens and Hypothetical Proof. Their closest analogs in mainstream generative grammar are the two rules Merge and Move. Thus, Modus Ponens is used to combine two signs (roughly, a 'function' sign and its argument), whereas Hypothetical Proof is used to 'discharge' a hypothetical constituent (similar to 'binding a trace with an empty operator' in MGG).

Before we can lay out the details of LG and start using it for linguistic analysis, we need to develop some a few technical tools, especially linear

logic, intuitionistic logic, typed lambda calculus, and higher order logic. To this end, we start by introducing a style of proof theory called **(sequent-style) natural deduction**, or just ND for short.

2 Proof Theory Basics and Linear Logic

Proof theory is the part of logic concerned with purely syntactic methods for determining whether a formula is deducible from a collection of formulas. Here what counts as a 'formula' varies from one theory to the next. The key thing is that the theory only cares about the syntactic form of formulas and not their interpretation (the part of logic that worries about that is **model theory**). What counts as a 'collection' also varies from one proof theory to the next. In some proof theories, collections are taken to be sets; in some they are taken to be strings; in the proof theories we will be concerned with, collections will be taken to be *finite multisets*.

Roughly speaking, finite multisets are a sort of compromise between strings and finite sets: they are stringlike in the sense that repetitions matter, but they are setlike in the sense that order does not matter. Technically, for any set S, we can think of a finite S-multiset as an equivalence class of S-strings, where two strings count as equivalent if they are permutations of each other. Alternatively, we can think of a finite S-multiset as a function from a finite subset of S to the positive natural numbers. (So if we indicate multisets between square brackets, then $[A]$ is a different multiset from $[A, A]$, but $[A, B]$ and $[B, A]$ are the same multiset.)

To define a proof theory, we first need a recursively defined set of **formulas**. Usually the base of the recursion provides some **basic** formulas, and then the recursion clauses tell how to get additional formulas using **connectives**. For example, in linear logic[1] (LL), which we will use as a running example in this section, we start with a set of basic formulas (say S, NP, and N), and then define the set of formulas as follows:

- Any basic formula is a formula.
- if A and B are formulas, so is $A \multimap B$.

Here the connective $\multimap$ is called **linear implication** (informally called 'lollipop'). Soon we will see that it works much like the familiar implication $\rightarrow$ of ordinary propositional logic, but with more limited options. A couple of

[1] Actually, there are many linear logics. The one we describe here, whose only connective is the **linear implication** $\multimap$, is linear implicative intuitionistic propositional logic.

standard notational conventions we will adopt are these: (1) we use uppercase italic roman letters as metavariables ranging over formulas; and (2) $\multimap$ associates to the right, e.g. $A \multimap B \multimap C$ abbreviates $A \multimap (B \multimap C)$.

A finite multiset of formulas is called a **context**. (Careful: this is a distinct usage from the notion of context as linguistically relevant features of the environment in which an expression is uttered.) We use capital Greek letters (usually Γ or Δ) as metavariables ranging over contexts. An ordered pair $\langle \Gamma, A \rangle$ of a context and a formula is called a **sequent**; then Γ is called the **context** of the sequent and A is called the **statement** of the sequent. The formula occurences in the context of a sequent are called **hypotheses** or **assumptions**.

The proof theory itself recursively defines a set of sequents, i.e. it recursively defines a relation between contexts and formulas. The relation defined by the theory is called **deducibility**, **derivability**, or **provability**, and is denoted by $\vdash$ (read 'deduces', 'derives', or 'proves'). The symbol '$\vdash$' itself is called 'turnstile'. The metalanguage assertion that $\langle \Gamma, A \rangle \in \vdash$ is usually written $\Gamma \vdash A$. Such an assertion is called a **judgment**. If Γ is the null multiset, we usually just write $\vdash A$. If Γ is the singleton multiset with one occurrence of B, we write $B \vdash A$. Commas in the contexts of judgments represent multiset union, so that, e.g. if $\Gamma = A, B$ and $\Delta = B$, then $\Gamma, \Delta = A, B, B$.

The base clauses of the proof theory are called **axioms**, and the recursion clauses are called **(inference) rules**. Axioms are just certain judgments, while rules are metalanguage conditional assertions, whose antecedents are conjunctions of judgments and whose consequent is a judgment. The judgments in the antecedent of a rule are called the **premisses** of the rule, and the consequent is called the **conclusion** of the rule.

For example, the proof theory for (pure) linear logic has one schema of axioms, and two schemas of rules. The axiom schema, variously called Refl (Reflexive Axioms), Hyp (Hypotheses), or simply Ax (Axioms) looks like this:

$$A \vdash A$$

To call this an axiom **schema** is just to say that upon replacing the metavariable A by any (not necessarily basic) formula, we get an axiom, such as

$$\mathrm{NP} \vdash \mathrm{NP}$$

(In LG, hypothesis axioms will be used as (the tecto part of) traces, which in turn will be a crucial ingredient in the analysis of certain linguistic phenomena known to phrase-structure grammarians as **unbounded dependencies**, and to mainstream generative grammarians as $\bar{\text{A}}$-**movement**.)

The two rule schemas of linear logic are as follows:

Modus Ponens, also called $\multimap$-Elimination ($\multimap$E):

$$\frac{\Gamma \vdash A \multimap B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B} \multimap\text{E}$$

Hypothetical Proof, also called $\multimap$-Introduction ($\multimap$I):

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \multimap\text{I}$$

Notice that Modus Ponens **eliminates** the connective $\multimap$, in the sense that there is an occurrence of $\multimap$ in one of the premisses (called the **major** premiss; the other premiss is called the **minor** premiss). Whereas, Hypothetical Proof **introduces** $\multimap$, in the sense that there is an occurrence of $\multimap$ in the conclusion but not in the (single) premiss. The presence of pairs of rules that eliminate and introduce connectives are characteristic of the natural-deduction style of proof theory.

If in fact $\Gamma \vdash A$, then we call the sequent $\langle \Gamma, A \rangle$ a **theorem** (in the present case, of linear logic). It is not hard to see that $\Gamma \vdash A$ if and only if there is a **proof tree** whose root is labelled with the sequent $\langle \Gamma, A \rangle$. Here, by a proof tree we mean an ordered tree whose nodes are labelled by sequents, such that

- the label of each leaf node is the sequent of an axiom; and
- the label of each nonleaf node is the sequent of the conclusion of a rule such that the sequents of the premisses of the rule are the labels of the node's daughters.

By convention, in displaying a proof tree, the root appears at the bottom and the leaves at the top (so from a logician's point of view, linguist's trees are upside down). Even though technically the labels are sequents, we conventionally write the corresponding judgments (metalanguage assertions that the sequents are deducible). And instead of edges connecting mothers to

daughters as in linguist's trees, we write horizontal lines with the label of the mother below and the labels of the daughters above (just as in inference rules). Sometimes as a mnemonic we label the horizontal line with the name of the rule schema that was instantiated there.

The simplest possible proof tree in linear logic has just one leaf which is also the root. In this case the only option is for the node to be labelled by a Hypothesis, e.g.:

$$\mathrm{NP} \vdash \mathrm{NP}$$

Intuitively, this can be thought of as: suppose you had an NP; then, sure enough, you'd have an NP. Don't worry if this doesn't seem to make any sense yet; it will become clear what this means when we use an elaborated form of Hypothesis (namely, trace) in a linguistic analysis.

Here is a (very) slightly less trivial proof tree:

$$\dfrac{\mathrm{NP} \vdash \mathrm{NP}}{\vdash \mathrm{NP} \multimap \mathrm{NP}} \multimap\mathrm{I}$$

One more example:

$$\dfrac{\dfrac{\dfrac{\mathrm{NP} \vdash \mathrm{NP} \qquad \mathrm{NP} \multimap \mathrm{S} \vdash \mathrm{NP} \multimap \mathrm{S}}{\mathrm{NP}, \mathrm{NP} \multimap \mathrm{S} \vdash \mathrm{S}} \multimap\mathrm{E}}{\mathrm{NP} \vdash (\mathrm{NP} \multimap \mathrm{S}) \multimap \mathrm{S}} \multimap\mathrm{I}}{\vdash \mathrm{NP} \multimap (\mathrm{NP} \multimap \mathrm{S}) \multimap \mathrm{S}} \multimap\mathrm{I}$$

3 Positive Intuitionistic Propositional Logic

Like most contemporary semantic theories, LG uses higher-order logic (HOL) as a notation for meanings. One of the hallmarks of LG (and related frameworks such as λ-grammar and ACG is that it also uses HOL as a notation for phenos. HOL is an elaboration of a system of notation for functions called **typed lambda calculus** (TLC). And TLC in turn is based on the proof theory of positive intuitionistic propositional logic (PIPL), which we now consider.

PIPL is just like the familiar propositional logic studied in introductory logic courses, except that (1) it has fewer connectives, and (2) it has fewer

rules. To be more specific, PIPL has the nullary connective (i.e. distinguished formula) T (read 'true') and the two binary connectives $\rightarrow$ (intuitionistic implication) and $\wedge$ (conjunction), but **not** F (false), $\vee$ (disjunction), or $\neg$ (negation)

The axioms of PIPL include the Hypothesis schema

$$A \vdash A$$

just as in linear logic, as well as the Truth axiom

$$\vdash \mathrm{T}$$

The rules of PIPL include elimination and introduction schemas for the two binary connectives, as well as two schemas of **structural** rules which only affect the contexts of sequents.

The rules for intuitionistic implication are exactly the same as for linear implication (except with $\multimap$ replaced by $\rightarrow$:

Modus Ponens, also called $\rightarrow$-Elimination ($\rightarrow$E):

$$\frac{\Gamma \vdash A \rightarrow B \qquad \Delta \vdash A}{\Gamma, \Delta \vdash B} \rightarrow\mathrm{E}$$

Hypothetical Proof, also called $\rightarrow$-Introduction ($\rightarrow$I):

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \rightarrow B} \rightarrow\mathrm{I}$$

The rules for conjunction include *two* elimination rules (for eliminating the left and right conjunct respectively):

$\wedge$-Elimination 1 ($\wedge$E1):

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge\mathrm{E1}$$

$\wedge$-Elimination 2 ($\wedge$E2):

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge\text{E2}$$

$\wedge$-Introduction ($\wedge$I):

$$\frac{\Gamma \vdash A \qquad \Delta \vdash B}{\Gamma, \Delta \vdash A \wedge B} \wedge\text{I}$$

Finally, the two structural rule schemas are as follows:

Weakening (W):

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A} \text{W}$$

Contraction (C):

$$\frac{\Gamma, B, B \vdash A}{\Gamma, B \vdash A} \text{C}$$

Weakening says that if we can prove something from certain assumptions, we can also prove it with more assumptions. Contraction says that repeated assumptions can be eliminated. These may seem too obvious to be worth stating, but in fact they *must* be stated, because in some logics (such as LL) they are not available!

As exercises, show that the following are provable in PIPL (their counterparts with $\to$ replaced by $\multimap$ are *not* provable in LL!): (1) $A \vdash B \to A$, and (2) $A \to A \to B \vdash A \to B$.