Introduction to Linear Categorial Grammar (LCG)

Carl Pollard

November 18, 2014

Linear Categorial Grammar (LCG)

LCG is a practical framework for linguistic analysis influenced by three traditions in linguistic theory:

- **categorial grammar (CG)**, a kind of syntactic analysis founded by Joachim Lambek (1958) that treats grammar rules as inference rules of a **proof theory**.

- **Montague semantics (MS)**, a higher-order theory of sentence meaning founded by Richard Montague (late 1960s), influenced by Frege, Carnap, and Kripke.

- **dynamic semantics**, theories of discourse meaning proposed by Lauri Karttunen, Hans Kamp, Irene Heim, and others (1970s-1980s), influenced by Hamblin, Lewis, and Stalnaker, emphasizing the interaction between utterance context and utterance meaning.

Sources of LCG

LCG builds on recent developments in all three of these traditions:

- **curryesque** CG, which analyzes syntax using **linear logic** (LL). Inspired by programmatic ideas of Curry (1961) and technical innovations of Oehrle (1994).

- **agnostic hyperintensional semantics (AHS)**. a higher-order semantic theory which is *weaker* than MS and makes *finer* meaning distinctions (already discussed).

- the line of work in dynamic semantics based on mainstream type theory/HOL established by Muskens (1996), Beaver (2001), de Groote (2006), and others.

For now, we focus in the first of these.

Pheno, Tecto, and Semantics

In a 1948 lecture, published in expanded form in 1961, Curry proposed that a linguistic expression should be analyzed as consisting of:

1. a **phenogrammatical** component: specifies the expression's surface form

2. a **tectogrammatical** component: specifies the the expression's combinatory potential

3. a **semantic** component: specifies the expression's meaning

The Phenogrammatical Component

- Usually abbreviated to just **pheno**

- Corresponds roughly to what linguists call **phonology**, broadly construed to include word order and intonation.

- relates to what the expression sounds like (or in the case of sign language, looks like)

The Tectogrammatical Component

- Usually abbreviated to just **tecto**.
- Corresponds roughly to what linguists call **syntactic category**.
- Relates to what other expressions the expression can combine with, and what results from the combination.

Syntactic Analyses as Proofs (1/2)

- In 1958, Lambek invented his **syntactic calculus**, later called **Lambek calculus**.
- A Lambek calculus is a grammar written in the form of a **logical proof system**.
- This is also true of other kinds of CG, including LCG; the main difference is that the underlying logic is different (linear logic for LCG, vs. bilinear logic for the Lambek calculus).

Syntactic Analyses as Proofs (2/2)

- The role of linguist's trees is taken over by **proof trees**.
- Words correspond to **axioms**.
- Grammar rules are replaced by **inference rules**.

- Linguistic expressions correspond to **theorems** of the proof system.

- Analyses of linguistic expressions correspond to **proofs** of the corresponding theorems.

Montague Grammar (1/3)

- A Montague grammar (MG) recursively defines a set of triples, each consisting of a **word string**, a **syntactic type**, and a HOL term denoting a meaning.

- In retrospect, we can relate Montague's string to Curry's pheno, and Montague's syntactic type to Curry's tecto.

- Some of the triples (**lexical entries**) are given at the outset, while the **rules** of the grammar produce new triples from old ones.

Montague Grammar (2/3)

- Each MG rule includes 'recipes' specifying how to construct the string and meaning of a new expression, respectively, from the strings and meanings of the expression's immediate constituents.

- On the semantic side, this last point is a version of Frege's notion of semantic *compositionality*.

- The operation involved in constructing the new expression's string is usually *concatenation*.

Montague Grammar (3/3)

- In MG, the operation involved in constructing the new expression's meaning is usually *function application*.

- Compared with Lambek calculus (or LCG), MG was impoverished, lacking any rule corresponding to Hypothetical Proof.

- In the 1980s van Benthem pointed out that Lambek calculus could be interfaced with Montague semantics by having Hypothetical Proof correspond to *lambda abstraction* in the semantics.

Oehrle

Oehrle (1994) introduced three technical innovations in categorial grammar.

1. Replace the Lambek calculus with a simpler logic, namely **linear logic (LL)**.

2. Allow phenos to be not just strings, but also (possibly higher-order) functions over strings.

3. Use type raising and function application in the pheno component to get the effect of Montague's 'quantifying in'. (We will extend this technique to deal with various kinds of focus constructions, '*wh*-movement', etc.)

LCG Overview

- An LCG for a natural language is a proof system that recursively defines a set of ordered triples called **signs**, each of which is taken to represent an expression of the language.

- Signs are notated in the form

$$a : A; B; c : C$$

  where

  - $a : A$, the pheno, is a typed term of the pheno theory.
  - B, the tecto, is a formula of LL.
  - $c : C$, the semantics, is a typed term of (static) AHS. (Later, we'll switch to dynamic AHS.)

Note: pheno and semantic types are omitted when they can be inferred from the term.

LCG Architecture

An LCG consists of:

- Two kinds of **axioms**:
  - **logical** axioms, also called **traces**
  - **nonlogical** axioms, also called **lexical entries**

- Two logical rule schemas:
  - **Modus Ponens**, also called **$\multimap$-Elimination**
  - **Hypothetical Proof**, also called **$\multimap$-Introduction**

- A few *nonlogical* rules will be added later.

Before considering the precise form of the axioms and rules, we need to discuss the form of LCG **sequents**.

LCG Sequents (1/2)

- A sign is called **hypothetical** provided its pheno and semantics are both variables.
- An LCG **sequent** is an ordered pair $\langle \Gamma, A \rangle$ where Γ, the **context**, is a set of hypothetical signs; and A, the **statement**, is a sign.
- The hypothetical signs in Γ are called the **hypotheses** of the sequent.
- Any two hypotheses in a context must have distinct pheno variables and distinct semantic variables.

LCG Sequents (2/2)

- The grammar, made up of axioms and rules, recursively defines a set of sequents.
- The notation

  $$\Gamma \vdash A$$

  asserts that the sequent $\langle \Gamma, A \rangle$ belongs to the set of sequents defined by the grammar.
- And so the turnstile symbol '$\vdash$' denotes a relation between contexts and signs.
- When Γ is empty, we write '$\vdash A$' rather than '$\emptyset \vdash A$'.
- If $\vdash A$, we say that the sign A is **generated** by the grammar.

The Trace Axiom Schema

Full form:

$$p : P; A; z : B \vdash p : P; A; z : B$$

Short form (when types of variables are known):

$$p; A; z \vdash p; A; z$$

Note: Soon we'll see that these axioms play a role analogous to that of traces in mainstream generative grammar.

Two Lexical Entries to Get Started

$\vdash \text{it}; \text{It}; *$ (dummy pronoun *it*)

$\vdash \lambda_s.s \cdot \text{rained}; \text{It} \multimap \text{S}; \lambda_o.\mathsf{rain} : \text{T} \to \text{p}$

Here the **unit type** T denotes a singleton set, whose only member, denoted by $*$, is used for vacuous meanings (following Carpenter 1997).

We treat dummy 'pronoun' *it* as belonging to its own syntactic category It.

The Two LCG Rule Schemas (Full Form)

- Modus Ponens

$$\frac{\Gamma \vdash f : A \to D; B \multimap E; g : C \to F \qquad \Delta \vdash a : A; B; c : C}{\Gamma, \Delta \vdash f\ a : D; E; g\ c : F}$$

- Hypothetical Proof

$$\frac{\Gamma, p : P; A; z : B \vdash a : C; D; b : E}{\Gamma \vdash \lambda_p.a : P \to C; A \multimap D; \lambda_z.b : B \to E}$$

The Two LCG Rule Schemas (Short Form)

These are used when the types of the terms are known.

- Modus Ponens

$$\frac{\Gamma \vdash f; B \multimap E; g \qquad \Delta \vdash a; B; c}{\Gamma, \Delta \vdash f\ a; E; g\ c}$$

  This is LCG's counterpart of Merge in mainstream generative grammar. It corresponds to *function application* in pheno and semantics.

- Hypothetical Proof

$$\frac{\Gamma, p; A; z \vdash a; D; b}{\Gamma \vdash \lambda_p.a; A \multimap D; \lambda_z.b}$$

  This is LCG's counterpart of Move in mainstream generative grammar. It corresponds to *lambda abstraction* in pheno and semantics.

An LCG Analysis

With the terms unsimplified:

$$\frac{\vdash \lambda_s.s \cdot \text{rained}; \text{It} \multimap \text{S}; \lambda_o.\mathsf{rain} \qquad \vdash \text{it}; \text{It}; *}{\vdash (\lambda_s.s \cdot \text{rained})\ \text{it}; \text{S}; (\lambda_o.\mathsf{rain})\ *}$$

With the terms simplified:

$$\frac{\vdash \lambda_s.s \cdot \text{rained}; \text{It} \multimap \text{S}; \lambda_o.\mathsf{rain} \qquad \vdash \text{it}; \text{It}; *}{\vdash \text{it} \cdot \text{rained}; \text{S}; \mathsf{rain}}$$

Note: Without comment, we use provable equalities (usually, rule (β)) of the pheno and semantic theories to simplify terms in intermediate conclusions before using them as premisses for later rule instances.

More Lexical Entries

$\vdash \text{pedro}; \text{NP}; \mathsf{p}$

$\vdash \text{chiqita}; \text{NP}; \mathsf{c}$

$\vdash \text{maria}; \text{NP}; \mathsf{m}$

$\vdash \lambda_s.s \cdot \text{brayed}; \text{NP} \multimap \text{S}; \mathsf{bray}$

$\vdash \lambda_{st}.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{NP} \multimap \text{S}; \mathsf{beat}$

$\vdash \lambda_{stu}.s \cdot \text{gave} \cdot t \cdot u; \text{NP} \multimap \text{NP} \multimap \text{NP} \multimap \text{S}; \mathsf{give}$

$\vdash \lambda_{st}.s \cdot \text{believed} \cdot t; \text{NP} \multimap \bar{\text{S}} \multimap \text{S}; \mathsf{believe}$

$\vdash \lambda_{stu}.s \cdot \text{persuaded} \cdot t \cdot u; \text{NP} \multimap \text{NP} \multimap \bar{\text{S}} \multimap \text{S}; \mathsf{persuade}$

Note: The finite verb entries are written to combine the verb first with the subject, then with the complements (the reverse of how things are traditionally done!)

Still More Lexical Entries

$\vdash \text{donkey}; \text{N}; \mathsf{donkey}$

$\vdash \text{farmer}; \text{N}; \mathsf{farmer}$

$\vdash \lambda_s.\text{that} \cdot s; \text{S} \multimap \bar{\text{S}}; \lambda_p.p$ (complementizer *that*)

$\vdash \lambda_{fs}.s \cdot \text{that} \cdot (f\ \mathbf{e}); (\text{NP} \multimap \text{S}) \multimap \text{N} \multimap \text{N}; \mathsf{that}$
(relativizer *that*)

$\vdash \lambda_{sf}.f\ (\text{every} \cdot s); \text{N} \multimap \text{QP}; \mathsf{every}$

$\vdash \lambda_{sf}.f\ (\text{some} \cdot s); \text{N} \multimap \text{QP}; \mathsf{some}$

Note: 'QP' (for 'quantified NP') abbreviates $(\text{NP} \multimap \text{S}) \multimap \text{S}$.

The motivation for giving these expressions a 'raised' tecto type is semantic and will be explained in due course.

Another LCG Analysis

$$\frac{\vdash \lambda_s.s \cdot \text{brayed}; \text{NP} \multimap \text{S}; \mathsf{bray} \qquad \vdash \text{chiqita}; \text{NP}; \mathsf{c}}{\vdash \text{chiqita} \cdot \text{brayed}; \text{S}; \mathsf{bray\ c}}$$

Yet Another LCG Analysis

$$\dfrac{\dfrac{\vdash \lambda_{st}.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{NP} \multimap \text{S}; \mathsf{beat} \qquad \vdash \text{pedro}; \text{NP}; \mathsf{p}}{\lambda_t.\text{pedro} \cdot \text{beat} \cdot t; \text{NP} \multimap \text{S}; \mathsf{beat\ p}} \qquad \vdash \text{chiqita}; \text{NP}; \mathsf{c}}{\text{pedro} \cdot \text{beat} \cdot \text{chiquita}; \text{S}; \mathsf{beat\ p\ c}}$$

Sorry, had to shrink this to scriptsize to fit it on the slide!

The Same Analysis with Semantics Omitted

Alternatively, if we don't care about semantics, we can sometimes overcome the space problem by omitting the semantic components of the signs:

$$\dfrac{\dfrac{\vdash \lambda_{st}.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{NP} \multimap \text{S} \qquad \vdash \text{pedro}; \text{NP}}{\lambda_t.\text{pedro} \cdot \text{beat} \cdot t; \text{NP} \multimap \text{S}} \qquad \vdash \text{chiqita}; \text{NP}}{\text{pedro} \cdot \text{beat} \cdot \text{chiquita}; \text{S}}$$

This approach has its limits.

An Oversized LCG Analysis

I shrunk this one to tiny and it still won't fit!

$$\dfrac{\dfrac{\vdash \lambda_{st}.s \cdot \text{believed} \cdot t; \text{NP} \multimap \bar{\text{S}} \multimap \text{S} \qquad \vdash \text{pedro}; \text{NP}}{\vdash \lambda_t.\text{pedro} \cdot \text{believed} \cdot t; \bar{\text{S}} \multimap \text{S}} \qquad \dfrac{\vdash \lambda_s.\text{that} \cdot s; \text{S} \multimap \bar{\text{S}} \qquad \dfrac{\vdash \lambda_s.s \cdot \text{brayed}; \text{NP} \multimap \text{S} \qquad \vdash \text{chiqita}; \text{NP}}{\vdash \text{chiquita} \cdot \text{brayed}; \text{S}}}{\vdash \text{that} \cdot \text{chiquita} \cdot \text{brayed}; \bar{\text{S}}}}{\vdash \text{pedro} \cdot \text{believed} \cdot \text{that} \cdot \text{chiquita} \cdot \text{brayed}; \text{S}}$$

Another Solution to the Space Problem

[1]:

$$\dfrac{\vdash \lambda_{st}.s \cdot \text{believed} \cdot t; \text{NP} \multimap \bar{\text{S}} \multimap \text{S}; \mathsf{believe} \qquad \vdash \text{pedro}; \text{NP}; \mathsf{p}}{\vdash \lambda_t.\text{pedro} \cdot \text{believed} \cdot t; \bar{\text{S}} \multimap \text{S}; \mathsf{believe\ p}}$$

[2]:

$$\dfrac{\vdash \lambda_s.\text{that} \cdot s; \text{S} \multimap \bar{\text{S}}; \lambda_p.p \qquad \dfrac{\vdash \lambda_s.s \cdot \text{brayed}; \text{NP} \multimap \text{S}; \mathsf{bray} \qquad \vdash \text{chiqita}; \text{NP}; \mathsf{c}}{\vdash \text{chiquita} \cdot \text{brayed}; \text{S}; \mathsf{bray\ c}}}{\vdash \text{that} \cdot \text{chiquita} \cdot \text{brayed}; \bar{\text{S}}; \mathsf{bray\ c}}$$

$$\dfrac{[1] \qquad [2]}{\vdash \text{pedro} \cdot \text{believed} \cdot \text{that} \cdot \text{chiquita} \cdot \text{brayed}; \text{S}; \mathsf{believe\ p\ (bray\ c)}}$$

Quantified Noun Phrases (QPs)

[1]:

$$\frac{\vdash \lambda_{sf}.f\ (\text{some} \cdot s); \text{N} \multimap \text{QP}; \mathsf{some} \qquad \vdash \text{farmer}; \text{N}; \mathsf{farmer}}{\vdash \lambda_f.f\ (\text{some} \cdot \text{farmer}); \text{QP}; \mathsf{some\ farmer}}$$

[2]:

$$\frac{\vdash \lambda_{sf}.f\ (\text{every} \cdot s); \text{N} \multimap \text{QP}; \mathsf{every} \qquad \vdash \text{donkey}; \text{N}; \mathsf{donkey}}{\vdash \lambda_f.f\ (\text{every} \cdot \text{donkey}); \text{QP}; \mathsf{every\ donkey}}$$

This analysis of QPs is due to Oehrle (1994).

Note that the phenos are not strings, but rather have the raised type $(\text{s} \to \text{s}) \to \text{s}$.

These are used to get the effect of Montague's 'quantifying-in'.

Quantifying In

$$\frac{[2] \qquad \vdash \lambda_s.s \cdot \text{brayed}; \text{NP} \multimap \text{S}; \mathsf{bray}}{\vdash \text{every} \cdot \text{donkey} \cdot \text{brayed}; \text{S}; \mathsf{every\ donkey\ bray}}$$

Quantifier Scope Ambiguity (1/2)

To get the 'surface scope' reading, we start by introducing a trace into the subject position:

[3]:

$$\frac{\vdash \lambda_{st}.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{NP} \multimap \text{S}; \mathsf{beat} \qquad s; \text{NP}; x \vdash s; \text{NP}; x}{s; \text{NP}; x \vdash \lambda_t.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{S}; \mathsf{beat}\ x}$$

Then we quantify in the object QP; bind the subject trace; and finally quantify in the subject QP:

$$\dfrac{[1] \qquad \dfrac{\dfrac{[2] \qquad [3]}{s; \text{NP}; x \vdash s \cdot \text{beat} \cdot \text{every} \cdot \text{donkey}; \text{NP} \multimap \text{S}; \mathsf{every\ donkey\ (beat}\ x)}}{\vdash \lambda_s.s \cdot \text{beat} \cdot \text{every} \cdot \text{donkey}; \text{NP} \multimap \text{S}; \lambda_x.\mathsf{every\ donkey\ (beat}\ x)}}{\vdash \text{some} \cdot \text{farmer} \cdot \text{beat} \cdot \text{every} \cdot \text{donkey}; \text{S}; \mathsf{some\ farmer}\ (\lambda_x.\mathsf{every\ donkey\ (beat}\ x))}$$

Quantifier Scope Ambiguity (2/2)

To get the 'crossed scope' reading, we start as before, introducing a trace into the subject position:

[3]:

$$\frac{\vdash \lambda_{st}.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{NP} \multimap \text{S}; \mathsf{beat} \qquad s; \text{NP}; x \vdash s; \text{NP}; x}{s; \text{NP}; x \vdash \lambda_t.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{S}; \mathsf{beat}\ x}$$

But then, we introduce a *second* trace in the object position; bind the subject trace and quantify in the subject QP; and finally, bind the *object* trace and quantify in the *object* QP:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[2]\quad \vdash \lambda_t.\text{some} \cdot \text{farmer} \cdot \text{beat} \cdot t; \text{NP} \multimap \text{S}; \lambda_y.\mathsf{some\ farmer}\ (\lambda_x.\mathsf{beat}\ x\ y)}}{t; \text{NP}; y \vdash \text{some} \cdot \text{farmer} \cdot \text{beat} \cdot t; \text{S}; \mathsf{some\ farmer}\ (\lambda_x.\mathsf{beat}\ x\ y)}}{[1]\quad t; \text{NP}; y \vdash \lambda_s.s \cdot \text{beat} \cdot t; \text{NP} \multimap \text{S}; \lambda_x.\mathsf{beat}\ x\ y}}{t; \text{NP}; y, s; \text{NP}; x \vdash s \cdot \text{beat} \cdot t; \text{S}; \mathsf{beat}\ x\ y}}{[3]\quad t; \text{NP}; y \vdash t; \text{NP}; y}}{\vdash \text{some} \cdot \text{farmer} \cdot \text{beat} \cdot \text{every} \cdot \text{donkey}; \text{S}; \mathsf{every\ donkey}\ (\lambda_y.\mathsf{some\ farmer}\ (\lambda_x.\mathsf{beat}\ x\ y))}$$