Categorial grammars for computing the correspondence between syntax and semantics

An example of interplay between mathematical logic and computational linguistics

Christian Retoré

Université de Bordeaux, LaBRI-CNRS & INRIA

EALING ENS, Paris, Monday September 22nd, 2008

Warning

This lecture is intended to be nothing more than a lecture:

I assume that the prototypic attendant is a student possibly interested in mathematical logic.

Colleagues will be left wanting for more, sorry.

Objective: provide a rather detailed example of relevant use of mathematical logic in formal or computational linguistics by a study of the simplest deductive system handling both syntax and the convergence with semantics: AB grammars, Lambek grammars.

Introduction: the mathematics of computational linguistics

1. Statistics and probability
   - if plain statistics, linguistically frustrating,
   - needs to combine with symbolic methods
   - only makes use of mathematically closed questions (afaik)
2. Formal Language theory, logic (model theory) (Stabler's lecture)
   - slightly beyond context free languages
   - polynomial parsing
   - learnable from positive examples (often left out)
3. ***Logic, proof theory and (typed) lambda calculus***
   - for semantics as expected
   - but also for syntax (intriguing connections with 2)
   - for the correspondence between syntax and semantics

Guidelines

Can some fine mathematics be relevant to formal linguistics?

Can linguistics raise some fruitful motivations to logic?

If language is a computation of some kind can it be depicted by computational logic?

Principles of categorial grammars (CG)

1. Lexical items are mapped into complex categories (lexicon, a.k.a. dictionnary).

2. Universal rules regulate how categories combine.

3. Rules are deductive rules.

1,2: lexicalised grammars

3: ***logical*** CG (Lambek, Moortgat) $\neq$ combinatory CG (Steedman).

Standard history

Rather issued from the logical, philosophical side:

Aristotle, Husserl, ...

Ajdukiewicz (1931) fractions, type checking for formula wellformedness

Bar-Hillel (1953): AB grammars, directionality for depicting word order

Lambek (1958, happy birthday): rules completed into a logical system

van Benthem, Moortgat (1986): models and modal extensions

Subjective history

I would add:

Montague (1970): although he thought that intermediate structures, CG analyses, should disappear (syntax $\longrightarrow$ CG analyses $\longrightarrow$ truth valued models)

Girard (1987), Abrusci (1991): non commutative linear logic, relation to intuitonistic logic, proof nets

Outcomes of categorial grammars

In this order:

- Relation to semantics (in between syntax and semantics)
- Learning algorithms (Buszkowski, Penn, Kanazawa)

More anecdotic, but interesting: Proof nets and human processing (Johnson, Morrill)

Some beautiful results about categorial grammars

1992 Pentus Lambek grammars are context free

1994 Kanazawa convergence of the learning algorithm for AB categorial grammars

2003 Pentus Lambek calculus is NP complete

2007 Salvati Lambek analyses are an Hyper Egde replacement graph language

Current issues in categorial grammars

Extending CG syntax beyond context-freeness, according to linguistic theory e.g. categorial minimalist grammars. (Lecomte, Retoré, Amblard,..),

Practical development of CG for Natural language Processing. (Moortgat, Moot, Steedman, Hockenmaier,..)

Learning classes of categorial grammars from structured data (Tellier, Foret, Bechet,...) or from corpora (Hockenmaier, Moot, ...)

Current issues in categorial grammars Cont’ed

Formal grammar in a type theoretical frame work with a mapping to semantics: abstract categorial grammars (de Groote, Pogodalla, Salvati,...)

Deductions as trees of formal language theory (Tiede, Retoré, Salvati,..)

Various kinds of non commutative or partially commutative linear logic. (Abrusci, Retoré, Ruet,...)

Lexical semantics in a compositional framework

Move from sentence semantics to (short) discourses, compositional DRT (Muskens, de Groote)

Technical contents

- AB grammars
  - definition, examples
  - relation to CFG
  - learnability
- Lambek calculus
  - (without product) definition, examples, properties
  - relation to Montague semantics

Classical categorial grammars: AB grammars

Categories a.k.a. types a.k.a fractions:

$$\mathsf{L} ::= \mathsf{P} \quad | \quad \mathsf{L} \setminus \mathsf{L} \quad | \quad \mathsf{L} / \mathsf{L}$$

P base categories

S (for sentences)

np (for noun phrases)

n (for nouns),

if you wish pp (for prepositional phrase), vp (for verb phrase) etc.

Lexicon/grammar generated language

$\mathrm{Lex}:$ $\quad m$: word /terminal $\longmapsto \mathrm{Lex}(m)$ finite subset of $\mathcal{T}$.

$w_1 \cdots w_n$, is of type u whenever there exists for each w_i a type t_i in $\mathrm{Lex}(w_i)$ such that $t_1 \cdots t_n \longrightarrow u$ with the following reduction patterns:

$$\forall u, v \in \mathsf{L} \qquad \begin{array}{lll} u(u \setminus v) \longrightarrow v & & (\setminus_e) \\ (v \,/\, u)u \longrightarrow v & & (/_e) \end{array}$$

The set of sentences or the language generated by the grammar is the set of word sequences of type S.

Lexicalised grammar with structured non-terminals
(coherent with modern linguistic theories)

The universal rules are called residuation laws, or simplifications, or elimination rules modus ponens. What do they do?

- If $y : A \setminus B$ then adding any expression $a : A$ on its left yields an expression $ay : B$.

- Symmetrically, if $z : B\ /\ A$ and $a : A$ then $za : B$;

The derivation tree is simply a binary tree whose leaves are the t_i and whose nodes are labeled by rules $/_e$ and $\setminus_e$.

Reduced form for AB grammars

Proposition 1 (Gaifman) *Every AB grammar is equivalent to an AB grammar containing only types of the form*

$$p \qquad (p \,/\, q) \qquad ((p \,/\, q) \,/\, r)$$

where p, q, r *stand for primitive types.*

PROOF : This theorem is an immediate consequence of propositions 3 and 2 to be proved below using Greibach normal form theorem that is now famous. This enables a simpler proof. ◇

Example: a tiny AB grammar

Word	Type(s)
$cosa$	$(S\,/\,(S\,/\,np))$
$guarda$	$(S\,/\,vp)$
$passare$	$(vp\,/\,np)$
il	$(np\,/\,n)$
$treno$	n

$guarda\ passare\ il\ treno : S$ indeed:

$$\begin{array}{l} (S\,/\,vp) \quad (vp\,/\,np) \quad (np\,/\,n) \quad n \\ \longrightarrow (S\,/\,vp) \quad (vp\,/\,np) \quad np \\ \longrightarrow (S\,/\,vp) \quad vp \\ \longrightarrow S \end{array}$$

The derivation tree for this analysis can be written as:

$$[_{/_e}(S \,/\, vp) \quad [_{/_e}(vp \,/\, np) \quad [_{/_e}(np \,/\, n) \quad n]]]$$

$cosa\ guarda\ passare$ not ok, since no rule can reduce the sequence

$$(S \,/\, (S \,/\, np))(S \,/\, vp)(vp \,/\, np)$$

From CFG to AB grammars

Proposition 2 *Every ε-free Context-Free Grammar in Greibach normal form is strongly equivalent to an AB categorial grammar. Thus every ε-free Context-Free grammar is weakly equivalent to an AB categorial grammar.*

PROOF : Let us consider the following AB grammar:

- Its words are the terminals of the CFG.
- Its primitive types are the non terminals of the CFG.
- $\mathrm{Lex}(a)$, the finite set of types associated with a terminal a contains the formulae $((\cdots((X \,/\, X_n) \,/\, X_{n-1}) \,/\, \cdots) \,/\, X_2) \,/\, X_1$ such that there are non terminals $X, X_1, \ldots, X_n$ such that $X \longrightarrow aX_1 \cdots X_n$ is a production rule.

It is then easily observed that the derivation trees of both grammars are isomorphic. ◇

From AB grammars to CFG

Proposition 3 *Every AB grammar is strongly equivalent to a CFG in Chomsky normal form.*

PROOF : Let G be the CFG defined by:

- Terminals T are the words of the AB grammar.

- Non Terminals NT are all the subtypes of the types appearing in the lexicon of the AB grammar — a type is considered to be a subtype of itself.

- The production rules are of two kinds:
  - $X \longrightarrow a$ whenever $X \in \mathrm{Lex}(a)$
  - $X \longrightarrow (X \mathbin{/} Z)Z$ and $X \longrightarrow Z(Z \setminus X)$ for all $X, Z \in NT$ — beware that from the CFG viewpoint $(Z \setminus X)$ or $(X \mathbin{/} Z)$ is a ***single*** non terminal.

◇

Learnability of rigid AB grammars

Rigid: one type per word. Contains non regular languages, but does not contain all of them. Some context free languages, but not even all regular languages.

Rigid AB grammars are learnable from structures from positive examples in Gold sense.

Buskowski & Penn algorithm 1990

Proof of convergence (in Gold sense) by Kanazawa 1994

A learning function maps finite sets of examples to a grammar which generates the examples in the set. Here examples are parse tree as the one produced by an AB, grammar with rules but no type associated with words.

Given any tree language T generated by an AB grammar and any enumeration of it $t_1, t_2,$ ..., there exists an integer N such that for all $n > N$, the language generated by the AB grammar $\phi(\{t_1, \ldots, t_N, \ldots, t_n\})$ is L.

Here language are parse trees, hence examples are trees as well.

- Assign types to the leaves of the examples $\{t_1, \ldots, t_n\}$
- Unify them (make the several types per word one).
- The resulting grammar if any is $\phi(\{t_1, \ldots, t_n\})$

If the examples are from an AB tree language, ϕ of a finite set of examples always exists and can be easily computed.

ϕ converges in the aforementioned sense.

Learning example

(1) $[_{\backslash e}[_{/e}$ a man $]$ swims $]$

(2) $[_{\backslash e}[_{/e}$ a fish $][_{\backslash e}$ swims fast $]]$

Typing:

(3) $[^{S}_{/e}[^{x_2}_{\backslash e}$ a:$(x_2 \, / \, x_1)$ man:$x_1]$ swims:$(x_2 \backslash S)]$

(4) $[^{S}_{\backslash e}[^{y_2}_{/e}$ a:$(y_2 \, / \, y_3)$ fish:$y_3][^{(y_2 \backslash S)}_{\backslash e}$ swims:y_1 fast:$(y_1 \backslash (y_2 \backslash S))]]$

We end up from the previous steps with several types per word. For instance the examples above yields:

(5)

$word$	$type1$	$type2$
a:	$x_2 \,/\, x_1$	$y_2 \,/\, y_3$
fast:		$y_1 \setminus (y_2 \setminus S)$
man:	x_1	
fish:		y_3
swims:	$x_2 \setminus S$	y_1

Unification:

(6)

$$\begin{array}{rcl} \sigma_u(x_1) & = & z_1 \\ \sigma_u(x_2) & = & z_2 \\ \sigma_u(y_1) & = & z_2 \setminus S \\ \sigma_u(y_2) & = & z_2 \\ \sigma_u(y_3) & = & z_1 \end{array}$$

which yields the rigid grammar/lexicon:

(7)

a:	$z_2 \,/\, z_1$
fast:	$(z_2 \setminus S) \setminus (z_2 \setminus S)$
man:	z_1
fish:	z_1
swims:	$z_2 \setminus S$

Key idea for the convergence

Define $G \sqsubset G'$ as the reflexive relation bewteen G and G' which holds when there exists a substitution σ such that $\sigma(G) \subset G'$ which dos not identify different types of a given word, but this is always the case when the grammar is rigid — $\sigma(G) \subset G'$ is a reflexive relation bewteen G and G' which holds whenever every assignment $a : T$ in G is in G' as well — in particular when G' is rigid, so is G, and they are equal. $\sqsubset$ is an order between grammars.

If the examples arise from a language, unification never fails, but increase the size of the grammar, and there are finitely many grammars below the one to reach.

As opposed to human learning, the language increases instead of specialising to the target language (except using semantics, variant by Tellier).

Limitations of AB-grammars

$(t \,/\, u)$ and $(u \,/\, v)$ does not yield $(t \,/\, v)$

Cosa guarda passare?.

$$(S \,/\, (S \,/\, np))\ (S \,/\, vp)\ (vp \,/\, np) \xrightarrow{(trans.)} (S \,/\, (S \,/\, np)) \quad (S \,/\, np)$$
$$\longrightarrow S$$

that/whom, should be $(n \setminus n) \,/\, (S \,/\, np)$ hence we would need a verb $np \setminus (S \,/\, np)$, unnatural

Maths question: categories = subsets of a free monoid, could there be completeness? (No, see later)

Product free Lambek grammars and calculus

Grammar

With the same notion of lexicon and of grammar,

$w_1 \cdots w_n$, is of type u whenever there exists for each w_i a type t_i in $\mathrm{Lex}(w_i)$ such that $t_1 \cdots t_n \longrightarrow u$

$t_1 \cdots t_n \longrightarrow u$ will be defined by adding extra rules and will be denoted by $t_1 \cdots t_n \vdash u$

Rules

We have two more rules:

this rule requires at least two free hyp.

A left most free hyp.

$$\dfrac{\begin{array}{c}\ldots[A]\ldots\ldots\\ \vdots\\ B\end{array}}{A\backslash B}\backslash_i$$ binding A

$$\dfrac{\begin{array}{c}\Delta\\ \vdots\\ A\end{array}\quad\begin{array}{c}\Gamma\\ \vdots\\ A\backslash B\end{array}}{B}\backslash_e$$

this rule requires at least two free hyp.

A right most free hyp.

$$\begin{array}{c} \ldots\ldots\ldots[A]\ldots \\ \vdots \\ \dfrac{B}{B\ /\ A}\ /_i \text{ binding } A \end{array}$$

$$\dfrac{\begin{array}{c}\Gamma\\ \vdots\\ B\ /\ A\end{array} \qquad \begin{array}{c}\Delta\\ \vdots\\ A\end{array}}{B}\ /_e$$

Natural deduction in Gentzen style $\neq$ sequent calculus

$$\frac{\Gamma \vdash A \quad \Delta \vdash A \setminus B}{\Gamma, \Delta \vdash B} \setminus_e \qquad \frac{A, \Gamma \vdash C}{\Gamma \vdash A \setminus C} \setminus_i \quad \Gamma \neq \varepsilon$$

$$\frac{\Delta \vdash B / A \quad \Gamma \vdash A}{\Delta, \Gamma \vdash B} /_e \qquad \frac{\Gamma, A \vdash C}{\Gamma \vdash C / A} /_i \quad \Gamma \neq \varepsilon$$

$$\frac{}{A \vdash A}\, axiom$$

An example

Here we take up again our small example of an Italian lexicon:

Word	Type(s)
$cosa$	$(S\ /\ (S\ /\ np))$
$guarda$	$(S\ /\ vp)$
$passare$	$(vp\ /\ np)$
il	$(np\ /\ n)$
$treno$	n

Transitivity of $/$, ***Cosa guarda passare*** with the Lambek calculus (we use Natural Deduction in Gentzen style):

$$\dfrac{(S / (S / np)) \vdash (S / (S / np)) \qquad \dfrac{\dfrac{(S / vp) \vdash (S / vp) \qquad \dfrac{(vp / np) \vdash (vp / n}{(vp / np), n}}{(S / vp), (vp / np), np \vdash S}\;/_i}{(S / vp), (vp / np) \vdash S / np}}{(S / (S / np)), (S / vp), (vp / np) \vdash S}\;/_e$$

Variables $\sim$ ***traces*** in chomskyan terms?

Some changes

Allows to reaarrange brackets in the Lambek calculus:
$(a \setminus b) / c \vdash a \setminus (b / c)$, etc.

Fine with object relatives introduced by ***whom/that*** having the type $(n \setminus n) / (S / np)$ with the unique category $(np \setminus S) / np$ for a transitive verb.

$x \vdash (z / x) \setminus z$ and $x \vdash z / (x \setminus z)$ hold for every categories x and z.

From a semantic viewpoint: an np (an individual) can be viewed as a $(S / np) \setminus S$ or $S / (np \setminus S)$ (a function form one place predicates to truth values) that is the set of all the properties of this individual.

The empty sequence

$A \setminus B$ requires an A object before

if $\vdash A$ then the empty sequence can be provided

a: $np \,/\, n$

rather $(n \,/\, n) \,/\, (n \,/\, n)$ ... rather boring: $(n \,/\, n)$

adj: $n \,/\, n$ (provable)

a rather book ???

$$\dfrac{\text{a} : np \,/\, n \qquad \dfrac{\dfrac{\text{rather} : (n \,/\, n) \,/\, (n \,/\, n) \qquad \dfrac{[n]_\alpha}{n \,/\, n}\,/_i{-\alpha}}{n \,/\, n}\,/_e \qquad \text{book} : n}{n}\,/_e}{np}\,/_e$$

Normalization of natural deduction

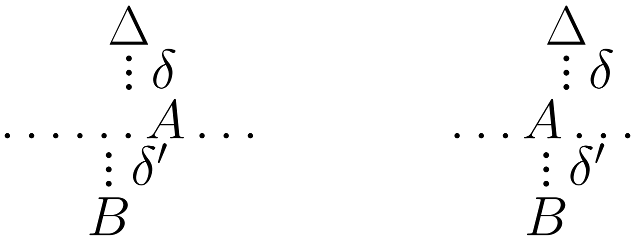
$$\dfrac{\dfrac{\begin{array}{c}\ldots\ldots\ldots[A]_\alpha\ldots\\ \vdots\;\delta'\\ B\end{array}}{B\,/\,A}\;/_i{-}\alpha \qquad \begin{array}{c}\Delta\\ \vdots\;\delta\\ A\end{array}}{B}\;/_e \qquad\qquad \dfrac{\begin{array}{c}\Delta\\ \vdots\;\delta\\ A\end{array} \qquad \dfrac{\begin{array}{c}\ldots[A]_\alpha\ldots\ldots\ldots\\ \vdots\;\delta'\\ B\end{array}}{A\,\backslash\,B}\;\backslash_i{-}\alpha}{B}\;\backslash_e$$

Whenever such a configuration appears, it can be reduced as follows:

1. find the hypothesis A which has been cancelled in the proof δ' of B under some hypotheses including A

2. replace this hypothesis with the proof δ of A

So the configurations above reduce to:



Proposition 4 *Natural deduction for L without product enjoys strong normalization, that is there are no infinite reduction sequences. (size decreases)*

Proposition 5 *Normalization is a locally confluent process. (they do not overlap)*

A ***principal branch*** leading to F a sequence $H_0, \ldots, H_n = F$ of formulae of a natural deduction tree such that:

- H_0 is a free hypothesis
- H_i is the principal premise — the one carrying the eliminated symbol — of an elimination rule whose conclusion is H_{i+1}
- H_n is F

Using this notion, by induction, one obtains:

Proposition 6 *Let d be a normal natural deduction (without product), then:*

1. *if d ends with an elimination then there is a principal branch leading to its conclusion*
2. *each formula in d is the sub-formula of a free hypothesis or of the conclusion*

Here is a proposition of Cohen 1967 needed to prove that every context free grammar is weakly equivalent to a Lambek grammar. It can be easily obtained using the normalisation result, and the subformula property for normal deduction.

Let us call the order $o(A)$ of a formula A the number of alternating implications:

- $o(p) = 0$ when p is a primitive type
- $o(A \setminus B) = \max(o(A) + 1, o(B))$
- $o(B \mathbin{/} A) = \max(o(A) + 1, o(B))$

Proposition 7 *A provable sequent $A_1, \ldots, A_n \vdash p$ of the product free Lambek calculus with $o(A_i) \leq 1$ and p a primitive type is provable with $\setminus_e$ and $/_e$ only — in other words AB derivations and L derivations coincide when types are of order at most one.*

Normalisation

Normal proof without an $\backslash_i$ rule followed by an $\backslash_e$ rule and without an $/_i$ rule followed by an $/_e$ rule.

Proposition 8 *In a normal proof of $H_1, \ldots, H_n \vdash C$ very formula is a sub formula of the conclusion C or of some hypothesis H_i. (induction using principal branch)*

Proposition 9 *Every proof of a given sequent $\Gamma \vdash A$ can be turned into a normal proof. (straightforward induction)*

Thsi entails:

Proposition 10 *Proof search in Lambek calculus and therefore parsing in Lambek grammars is decidable.*

Lambek calculus and Montague semantics

Semantic types: e: entities, t: truth values

$$types ::= e \quad | \quad t \quad | \quad types{\rightarrow}types$$

Constant	Type
$\exists$	$(e{\rightarrow}t){\rightarrow}t$
$\forall$	$(e{\rightarrow}t){\rightarrow}t$
$\wedge$	$t{\rightarrow}(t{\rightarrow}t)$
$\vee$	$t{\rightarrow}(t{\rightarrow}t)$
$\supset$	$t{\rightarrow}(t{\rightarrow}t)$

and proper constants for the denotation of the words in the lexicon:

$likes$	$\lambda x \lambda y\ (\text{likes}\ x)\ y$	$x : e,\ y : e, \text{likes} : e {\longrightarrow} (e {\longrightarrow} t)$
`<< likes >> is a two-place predicate`		
$Pierre$	$\lambda P\ (P\ \text{Pierre})$	$P : e {\longrightarrow} t,\ \text{Pierre} : e$

`<< Pierre >> is viewed as`
`the properties that << Pierre >> holds`		

Higher order logic in simply typed lambda calculus as Church did.

$e {\longrightarrow} t$: e common noun like ***chair*** or an intransitive verb like ***sleep***

$e {\longrightarrow} (e {\longrightarrow} t)$ a transitive verb like ***takes***

Morphism from syntactic types to semantic types:

(Syntactic type)*	=	Semantic type	
S^*	$=$	t	a sentence is a proposition
np^*	$=$	e	a noun phrase is an entity
n^*	$=$	$e{\longrightarrow}t$	a noun is a subset of the set of entities
$(a \setminus b)^* = (b \,/\, a)^*$	$=$	$a^* \longrightarrow b^*$	extends $(\_)^*$ to all syntactic types

The lexicon associates to each syntactic type $t_k \in \mathrm{Lex}(m)$ of a word m a λ-term τ_k ***whose type is precisely*** t_k^*, the semantic counter part of the syntactic type t_k.

Input for the algorithm computing the semantics

- ***a syntactic analysis of*** $m_1 \ldots m_n$ in Lambek calculus, that is a proof $\mathcal{D}$ of
  $t_1, \ldots, t_m \vdash S$ and

- ***the semantics of each word*** m_1***,. . . et*** m_n, that are λ-terms
  $\tau_i : t_i^*$,

The algorithm computing the semantics

1. Replace every syntactic type in $\mathcal{D}$ with its semantic counterpart; since intuitionistic logic extends the Lambek calculus the result $\mathcal{D}^*$ of this operation is a proof in intuitionistic logic of $t_1^*, \ldots, t_n^* \vdash t = S^*$.
2. Via the Curry-Howard isomorphism, this proof in intuitionistic logic can be viewed as a simply typed λ-term $\mathcal{D}_\lambda^*$ which contains one free variable x_i of type t_i^* per word m_i.
3. Replace in $\mathcal{D}_\lambda^*$. each variable x_i by the λ-term τ_i — whose type is also type t_i^*, so this is a correct substitution.
4. Reduce the resulting λ-term: this provides the semantics of the sentence (another syntactic analysis of the same sentence can lead to a different semantics).

Every normal λ-term of type t without free variables (with solely bound variables and constants) correspond to a formula of predicate calculus.

The previous algorithm relies on;

- embedding of L in intuitionistic logic,
- normalisation of type lambda terms
- predicate logic in typed lambda calculus

Let us see this at work:

word	***syntactic type*** u ***semantic type*** u^* ***semantics :*** λ***-term of type*** u^* x_v ***means that the variable or constant*** x ***is of type*** v
some	$(S \,/\, (np \setminus S)) \,/\, n$ $(e{\to}t){\to}((e{\to}t){\to}t)$ $\lambda P_{e\to t}\ \lambda Q_{e\to t}\ (\exists_{(e\to t)\to t}\ (\lambda x_e(\wedge_{t\to(t\to t)}(P\ x)(Q\ x))))$
statements	n $e{\to}t$ $\lambda x_e(\texttt{statement}_{e\to t}\ x)$
speak_about	$(np \setminus S) \,/\, np$ $e{\to}(e{\to}t)$ $\lambda y_e\ \lambda x_e\ ((\texttt{speak\_about}_{e\to(e\to t)}\ x)y)$
themselves	$((np \setminus S) \,/\, np) \setminus (np \setminus S)$ $(e{\to}(e{\to}t)){\to}(e{\to}t)$ $\lambda P_{e\to(e\to t)}\ \lambda x_e\ ((P\ x)x)$

Let us first show that ***Some statements speak about themselves.*** belongs to the language generated by this lexicon. So let us prove (in natural deduction) the following :

$$(S\,/\,(np\,\backslash\,S))\,/\,n\ ,\ n\ ,\ (np\,\backslash\,S)\,/\,np\ ,\ ((np\,\backslash\,S)\,/\,np)\,\backslash\,(np\,\backslash\,S) \vdash S$$

using the abbreviations So (some) Sta (statements) SpA (speak about) $Refl$ (themselves) for the syntactic types :

$$\dfrac{\dfrac{So \vdash (S/(np\backslash S))/n \quad Sta \vdash n}{So, Sta \vdash (S/(np\backslash S))}\,/_e \qquad \dfrac{SpA \vdash (np\backslash S)/np \quad Refl \vdash ((np\backslash S)/np}{SpA, Refl \vdash (np\backslash S)}}{So, Sta, SpA, Refl \vdash S}$$

Using the homomorphism from syntactic types to semantic types we obtain the following intuitionistic deduction, where $So^*, Sta^*, SpA^*, Refl^*$ are abbreviations for the semantic types respectively associated with the syntactic types: $So, Sta, SpA, Refl$:

$$\cfrac{\cfrac{So^* \vdash (e{\to}t){\to}(e{\to}t){\to}t \quad Sta^* \vdash e{\to}t}{So^*, Sta^* \vdash (e{\to}t){\to}t}\to_e \quad \cfrac{SpA^* \vdash e{\to}e{\to}t \quad Refl^* \vdash (e-}{SpA^*, Refl^* \vdash e-}}{So^*, Sta^*, SpA^*, Refl^* \vdash t}$$

The λ-term representing this deduction simply is

((some statements) (themsleves speak_about)) of type t

where *some,statements,themselves,speak_about* are variables with respective types

$$So^*, Sta^*, Refl^*, SpA^*$$

Let us replace these variables with the semantic λ-terms (of the same type) which are given in the lexicon. We obtain the following λ-term of type t (written on two lines) that we reduce:

$$\Big(\big(\lambda P_{e\to t}\,\lambda Q_{e\to t}\,(\exists_{(e\to t)\to t}\,(\lambda x_e(\wedge(P\,x)(Q\,x))))\big)\big(\lambda x_e(\texttt{statement}_{e\to t}\,x$$

$$\Big(\big(\lambda P_{e\to(e\to t)}\,\lambda x_e\,((P\,x)x)\big)\big(\lambda y_e\,\lambda x_e\,((\texttt{speak\_about}_{e\to(e\to t)}\,x)y)\big)\Big)$$

$$\downarrow\beta$$

$$\big(\lambda Q_{e\to t}\,(\exists_{(e\to t)\to t}\,(\lambda x_e(\wedge_{t\to(t\to t)}(\texttt{statement}_{e\to t}\,x)(Q\,x))))\big)$$

$$\big(\lambda x_e\,((\texttt{speak\_about}_{e\to(e\to t)}\,x)x)\big)$$

$$\downarrow\beta$$

$$\big(\exists_{(e\to t)\to t}\,(\lambda x_e(\wedge(\texttt{statement}_{e\to t}\,x)((\texttt{speak\_about}_{e\to(e\to t)}\,x)x)))\big)$$

The previous term represent the following formula of predicate calculus (in a more pleasant format) :

$$\exists x : e\,(\texttt{statement}(x) \wedge \texttt{speak\_about}(x, x))$$

This is the semantics of the analysed sentence.

Discussion

Fine point: compositionality at work with neat logical tools.

Technical difficulty: how to extend the syntax while keeping this correspondence? Abstract Categorial Grammars (for tree grammars) or Categorial Minimalist Grammars.

IMHO Intermediate language (despised by Montague) much more exciting than models and possible worlds.

What would be a good interpretation?

Discourse and DRT like objections solved by λ DRT or de Groote with contexts as argument.

IMHO main problem is lexical semantics: how to integrate relation between the predicates and constants in the model. It is related to a good interpretation without too much unfolding without too much knowledge representation.

Outcomes of this restricted CGs

Here we have only seen the simplest logical system for grammar as a deductive system, a purely logical one: Lambek calculus and grammars.

A grammar system quite different from standard formal language theory. (Evidence for the difference: the proofs are already in HR graph/tree grammars, incomparable with context free tree languages).

Because of its relation to ordinary logic, in particular intuitionistic logic, perfect for computing logical semantic representation in a way that impements compositionality).

Because of the lexicalisation and of the structure of the deductions/parse structure, easy to learn from structure, with a convergent algorithm.

Discussion

Pretty mathematical results from formal linguistics motivation.

Assuming the linguistic relevance of the model, are the mathematical results used?

- The connection with usual logic is clearly used especially for computing semantic representations.
- The existence of normal proof is used every where, but not the process of normalisation.
- Free semi group model is appealing, but not really used.
- Learning technique could be used for grammar construction from corpora (e.g. also works for Lambek grammars Bonato, Retoré)

Discussion

Is the linguistic model relevant?

The implicit claim is that deduction in a resource logic is related to linguistic processing and for some of us that the first includes the second.

- Orthodox view of deduction as parsing Multi Modal Categorial Grammars (Moortgat) Various connectives assosciative or not, commutative or not, modality and postulates for relation among them.

- Valency consumption(for semantic applciation) by commutative linear logic Intermediate out and word order as a separate process:
  - completely separate (IG Perrier, ACG de Groote)
  - partially separate, with the connectives commutative and not commutative à la de Groote Abrusci Ruet (CMG Lecomte, Retoré)

References

Two-part survey in the journal *La Gazette des mathématiciens* (Société Mathématique de France).
Ch. Retoré Les mathématiques de la linguistique computationnelle.
Premier volet: la théorie des langages. Volume 115 janvier 2008 pp. 35-62.
http://smf.emath.fr/Publications/Gazette/2008/115/
Second volet: Logique. Volume 116 avril 2008 pp. 29-63.
http://smf.emath.fr/Publications/Gazette/2008/116/

Lecture notes Ch. Retoré The logic of categorial grammars - Lecture Notes - Rapport de Recherche INRIA 5703
http://www.inria.fr/rrrt/rr-5703.html