Typological Grammar

First published Tue Sep 7, 2010

Typological grammars are substructural logics, designed for reasoning about the composition of form and meaning in natural language. At the core of these grammars are residuated families of type-forming operations; a hierarchy of typological grammars results from the choices one makes with respect to the structural properties of the type-forming operations, and the means one introduces to control the grammatical resource management.

Computational semantics is obtained from the Curry-Howard interpretation of categorial derivations. Parsing and natural language processing is modeled in terms of suitably refined versions of the proof nets of linear logic.

1. A bit of history

Typological grammar has its roots in two seminal papers written by Jim Lambek half a century ago (Lambek 1958, 1961). In these papers, the author set himself the aim “to obtain an effective rule (or algorithm) for distinguishing sentences from non-sentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages […]”. To realize this goal, the familiar parts of speech (nouns, verbs, …) are turned into formulas of a logic — a logic designed to reason about grammatical composition. The judgement whether a phrase is wellformed, under this view, is the outcome of a process of deduction. A decision procedure for such judgements is obtained by casting the grammar logic in the format of a Gentzen-style sequent calculus. The sequent presentation is extremely simple: the logic of grammar is a logic without structural rules. Contraction and Weakening are dropped; their presence would entail that wellformedness is unaffected by arbitrary copying or deletion of grammatical material. To take into account also word order and phrase structure
information, Lambek further removes the structural rules of Exchange and (in the 1961 paper) Associativity.

At the time of their publication, these ideas did not resonate; their impact on the field of computational linguistics dates from the 1980s. Two factors have played an important role in this belated recognition. The first was the addition of a computational semantics for categorial derivations along the lines of the Curry-Howard proofs-as-programs interpretation in van Benthem 1983, reprinted in Buszkowski et al. 1988. Lambek's typological grammars, so viewed, could be seen to realize Montague's compositionality program in an uncompromising way, with the lambda calculus and type theory providing powerful tools to study derivational and lexical semantics. The second factor was the introduction of linear logic (Girard 1987), and with it, the surge of interest in substructural logics. A key insight from linear logic injected into typelogical grammar is the idea that structural rules can be reintroduced in a controlled form by means of so-called modalities: in moving to more discriminating type logics, no expressivity is lost. From a computational point of view, variants of the proof nets of linear logic have provided the basis for typelogical natural language processing.

Typological grammar, in its current incarnations, keeps the general architecture of Lambek's original calculi, but extends this to a more articulate vocabulary of type-forming operations. Of central importance are the multiplicative operations, used to model grammatical composition; these form the focus of this article. Next to the multiplicatives one can consider additive or Boolean operations and first or second order quantification to handle phenomena of lexical ambiguity, type polymorphism and the management of feature information. Morrill 1994 is a good source of examples for such extensions.

Outline of this article. We start with the standard Lambek systems. We study their model-theoretic and proof-theoretic aspects (frame semantics, sequent calculus), and the relation between the two (soundness, completeness). We characterize compositional interpretation as a homomorphism relating a syntactic source calculus and a target calculus for meaning assembly. The mapping associates syntactic derivations with semantic readings, expressed as terms of the simply typed linear lambda calculus. In §3 we turn to the expressive limitations of the standard Lambek systems, in syntax and semantics, and study how they can be addressed by systematically extending the categorial architecture in a modular way. Generalizations concern the arity of the type-forming operations (binary composition operations versus unary control operators); multimodal extensions where multiple families of type-forming operations co-exist and communicate via structural interaction principles; one-sorted vs multi-sorted logics (discontinuous calculi); single-conclusion vs multiple-conclusion systems (symmetric calculi); and more structured views of the syntax- semantics interface (continuation semantics). To close the panoramic tour, we present proof nets for the various typelogical systems studied in §3, and we compare these systems with respect to expressivity and computational tractability.

2. The Lambek systems

2.1 Modelling grammatical composition

The type language to be considered in this section consists of a small set of atomic types, and is closed under the binary operations product, left and right division. Some notational conventions: in the specification below, lower case \( p \) ranges over atomic types, upper case \( A, B \) over arbitrary types. We pronounce \( A \, \mathbin{\&} \, B \) as ‘\( A \) under ‘\( B \)’ and \( B \, / \, A \) as ‘\( B \) over ‘\( A \)’ thus to make clear which part of the ‘fraction’ is the denominator (\( A \)), and which is the numerator (\( B \)).

- Types: \( A \, \mathbin{\&} \, B := p \mid A \, \mathbin{\&} \, B \mid A \, \mathbin{\&} \, (B \, / \, A) \)

In categorizing linguistic expressions, atomic types stand for phrases that one can think of as grammatically ‘complete’. Examples, for English, could be \( s \) for declarative sentences (‘Mary dreams’,...), \( np \) for noun phrases (‘Mary’, ‘the girl’,...), \( n \) for common nouns (‘girl’, ‘smart girl’,...). Division expresses incompleteness: an expression with type \( A \, \mathbin{\&} \, B \) will produce a phrase of type \( B \) when it is put together with a phrase of type \( A \) to its left (right). Product types \( A \, \mathbin{\&} \, B \) explicitly express the formation of a complex phrase out of constituent parts of type \( A \) and \( B \) in that order. So if ‘Mary’ is typed as \( np \) and the verb ‘dreams’ as \( np \, s \), we obtain \( s \) for the combination ‘Mary dreams’ by multiplying out these types: \( np \, \mathbin{\&} \, (np \, s) \). Similarly for the combination ‘the girl dreams’, where one would start from a type \( np \, n \) for the determiner and \( n \) for ‘girl’: \( ((np \, n) \, \mathbin{\&} \, n) \, \mathbin{\&} \, (np \, s) \).
To make this informal description of the interpretation of the type language precise, we turn to modal logic. Frames $F = (W, R)$, in the categorial setting, consist of a set $W$ of linguistic resources (expressions, ‘signs’), structured in terms of a ternary relation $R$. This ternary relation stands for grammatical composition, or ‘Merge’ as it is known in the generative tradition: read $Rxyz$ as “the expression $x$ is obtained by putting together the subexpressions $y$ and $z$”. For comparison: in the Routley-Meyer semantics for relevance logic, $R$ would be the accessibility relation interpreting the fusion operation.

A model $M$ is a pair $(F, V)$ where $V$ is an interpretation (valuation) sending atomic types to subsets of $W$. For complex types, the valuation respects the clauses below.

- $x \vdash A \otimes B$ iff $\exists yz$ such that $Rxyz$ and $y \vdash A$ and $z \vdash B$
- $y \vdash C/B$ iff $\forall xz$, if $Rxyz$ and $z \vdash B$, then $x \vdash C$
- $z \vdash A\setminus C$ iff $\forall xy$, if $Rxyz$ and $y \vdash A$, then $x \vdash C$

A syntactic calculus for a categorial type language is a deductive system producing statements of the form $A \vdash B$ (‘$B$ is derivable from $A$’). A statement $A \vdash B$ holds in a model ($M \models A \vdash B$) if $V(A) \subseteq V(B)$; it is valid ($\models A \vdash B$) if it holds in all models. For the minimal grammar logic, the set of derivations in the syntactic calculus is freely generated from the axiom and rules of inference below. This minimal system, for historical reasons, is known as $NL$ (for ‘Non-associative Lambek calculus’). The first line states that derivability is a reflexive, transitive relation, i.e., a preorder. The bidirectional (‘if and only if’) inference rules of the second line characterize the product and division operations as a residuated triple as it is known in residuation theory. (Galatos et al. 2007 provides good background reading.) The syntactic calculus, so defined, precisely fits the intended interpretation of the type language, in the sense of the soundness and completeness result below.

- preorder laws:
  - $A \vdash A$ (reflexivity)
  - from $A \vdash B$ and $B \vdash C$ infer $A \vdash C$ (transitivity)
- res budiation laws: $A \vdash C/B$ iff $A \otimes B \vdash C$ iff $B \vdash A\setminus C$
- soundness and completeness: $A \vdash B$ is provable in the grammatical base logic $NL$ iff $\models A \vdash B$, i.e., iff $V(A) \subseteq V(B)$ for every valuation $V$ on every frame $F$ (Došen 1992).

The completeness result for $NL$ does not impose any restrictions on the interpretation of the Merge relation $R$. This means that the theorems and inference rules of the minimal grammar logic have the status of grammatical invariants: properties of type combination that hold no matter what the structural particularities of individual languages may be. Here are some examples of such universally valid principles. They come in pairs, because of the left-right symmetry relating slash and backslash.

- application:
  - $(A/B) \otimes B \vdash A$
  - $B \otimes (BA) \vdash A$
- co-application:
  - $A \vdash (A \otimes B)/B$
  - $A \vdash B(B \otimes A)$
- lifting:
  - $A \vdash B/(AB)$
  - $A \vdash (B/A)B$
- monotonicity: from $A \vdash B$ and $C \vdash D$ infer any of the following:
  - $A \otimes C \vdash B \otimes D$
Given the universal freely generated syntactic calculus discussed above, the task of providing a categorial grammar for a particular language is reduced to specifying its lexicon: the typological framework, in this sense, represents the culminating point of ‘lexicalizing’ grammar formalisms. A categorial lexicon is a relation associating each word with a finite number of types. Given a lexicon $\text{Lex}$, a categorial grammar assigns type $B$ to a string of words $w_1 \cdots w_n$ iff for $1 \leq i \leq n$, $(w_i, A_i) \in \text{Lex}$, and $X \vdash B$ is provable in the syntactic calculus where $X$ is a product formula with yield $A_1 \cdots A_n$. The grammar can be considered adequate if the strings $w_1 \cdots w_n$ are indeed judged to be the wellformed phrases of type $B$.

Lexical type assignments don’t have to be handcrafted: Buszkowski and Penn (1990) model the acquisition of the lexicon as a process of solving type equations. Their unification-based algorithms take function-argument structures as input (binary trees with a distinguished head daughter); one obtains variations depending on whether the solution should assign a unique type to every vocabulary item (so-called rigid grammars), or whether one accepts multiple assignments ($k$-valued grammars, with $k$ as the maximal lexical ambiguity factor). Kanazawa (1998) studies learnable classes of grammars from this perspective, in the sense of Gold’s notion of identifiability ‘in the limit’. This line of research is by now well established, for typological grammars proper, and for the related pregroup formalism discussed under §6.

The radical lexicalism of typological grammar means there is no room for construction specific rule stipulations: central grammatical notions, rather than being postulated, must emerge from the type structure. Here are some examples.

- **Valency.** Intransitive verbs (‘dreams’, ‘sleeps’) form a complete sentence with just a subject noun phrase to their left; transitive verbs (‘sees’, ‘admires’) require both a subject to their left and a direct object to their right. Such selectional requirements are expressed in terms of the directional implications: $n p s$ for intransitive, $(n p s) / n p$ for transitive verbs. In a context-free grammar, these distinctions require the postulation of new non-terminals.

- **Case.** The distinction between phrases that can fulfill any noun phrase selectional requirement (say, proper names) versus case-marked pronouns insisting on playing the subject or direct object role is expressed through higher-order type assignment: proper names can be typed simply as $n p$, subject pronouns (‘he’, ‘she’) as $s / (n p s)$, direct object pronouns (and reflexives) as $((n p s) / n p) / (n p s)$. Under such typing, one correctly distinguishes between the grammatical ‘he sees her’ versus the ill-formed ‘him likes she’.

- **Complements vs modifiers.** Compare exocentric types $A / B$ or $B A$ with $A \neq B$ versus endocentric types $A / A$, $A A$. The latter express modification; optionality of modifier phrases follows. Compare ‘the girl’, ‘the smart girl’, ‘the girl (who teases Bill)’, with ‘girl’ typed as $n$, ‘smart’ as $n / n$ and the relative clause ‘who teases Bill’ as $n / n$.

- **Filler-gap dependencies.** Types with nested implications $A / (C / B)$, $A / (B / C)$, etc., signal the withdrawal of a gap hypothesis of type $B$ in a domain of type $C$. The relative pronoun ‘who’ in ‘the girl who teases Bill’ is typed as $(n / n n) / (n p s)$; it combines with a sentence of which the subject must remain unexpressed, as one sees from the ill-formed ‘the girl who Mary teases Bill’.

**Sequent calculus, decidability.** How can we decide whether a statement $A \vdash B$ is indeed a theorem of the syntactic calculus? In the presence of expanding rules, such as Lifting, this is not immediately obvious: the transitivity inference from $A \vdash B$ and $B \vdash C$ to $A \vdash C$ involves a ‘lemma’ formula $B$ which disappears from the conclusion, and there is an infinite number of candidate lemma formulas to consider. A key point of Lambek’s original papers consists in the reformulation of the syntactic calculus in an equivalent Gentzen-style sequent format, which enjoys cut-elimination.

Sequents for the grammatical base logic $\text{NL}$ are statements $X \Rightarrow A$, where $X$ (the antecedent) is a structure, and $A$ (the succedent) a type formula. The antecedent structure is required to be non-empty. A structure is either a single formula, or a binary branching tree with formulas at the leaves. In the sequent rules below, the tree structure of the antecedent is indicated by bracketing $(\cdots)$. The notation $X[Y]$ refers to a structure $X$ containing a distinguished substructure $Y$. For every connective (type-forming operation), there are two inference rules, introducing the connective to the left or to the right of the derivability arrow. In this sequent presentation, as in Gentzen’s propositional logics, Cut is an admissible rule: every proof of a theorem that makes use of Cut inferences can be transformed into

\[
A / D \vdash B / C \\
D A \vdash C B
\]
an equivalent cut-free derivation. Backward-chaining proof search in the cut-free system respects the subformula property and immediately yields a decision procedure.

\[
\frac{A \Rightarrow A}{A \Rightarrow A} \quad \text{Ax}
\]

\[
\frac{Y \Rightarrow A \quad X[A] \Rightarrow B}{X[Y] \Rightarrow B} \quad \text{Cut}
\]

\[
\frac{X \Rightarrow A \quad Y \Rightarrow B}{(X,Y) \Rightarrow A \otimes B} \quad \text{(\otimes R)}
\]

\[
\frac{X[(A,B)] \Rightarrow C}{X[A \otimes B] \Rightarrow C} \quad \text{(\otimes L)}
\]

\[
\frac{Y \Rightarrow B \quad X[A] \Rightarrow C}{X[(Y,B,A)] \Rightarrow C} \quad \text{(L)}
\]

\[
\frac{(B,X) \Rightarrow A}{X \Rightarrow B A} \quad \text{(R)}
\]

\[
\frac{Y \Rightarrow B \quad X[A] \Rightarrow C}{X[(A/B,Y)] \Rightarrow C} \quad \text{(/L)}
\]

\[
\frac{(X,B) \Rightarrow A}{X \Rightarrow A/B} \quad \text{(/R)}
\]

**NL:** Sequent Calculus

**A landscape of calculi.** From the pure residuation logic, one obtains a hierarchy of calculi by attributing associativity and/or commutativity properties to the composition operation \(\otimes\). These calculi reflect different views on how the grammatical material is structured. The table below summarizes the options.

<table>
<thead>
<tr>
<th>logic</th>
<th>structure</th>
<th>associative</th>
<th>commutative</th>
</tr>
</thead>
<tbody>
<tr>
<td>NL</td>
<td>tree</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>L</td>
<td>string</td>
<td>✓</td>
<td>–</td>
</tr>
<tr>
<td>NLP</td>
<td>mobile</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>LP (=MILL)</td>
<td>multiset</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

For reasons of historical precedence, the system of Lambek 1958, with an associative composition operation, is referred to as \(L\); the pure residuation logic \(NL\), i.e., the non-associative version of \(L\), was introduced later in Lambek 1961. Addition of commutativity turns \(L\) and \(NL\) into \(LP\) and \(NLP\), respectively. \(LP\), Lambek calculus with Permutation, was introduced in van Benthem 1983; in retrospect, this is the implication/fusion fragment of Multiplicative Intuitionistic Linear Logic (MILL). The commutative variant of \(NL\) has been studied by Kandulska; it has not found significant linguistic applications. For the four systems in the hierarchy, one writes \((N)L(P)^*\) for the (non-conservative) extension which allows empty antecedents.

In the presentation of the syntactic calculus, the structural options take the form of *non-logical axioms* (postulates), which one adds to the residuation laws. In the sequent presentation, one has corresponding structural rules. The sequent calculus for \(LP\) also allows for a sugared presentation where the antecedent is treated as a list or a multiset of formulas; the inference steps for the structural rules can then be left implicit. With respect to the frame semantics, each of the
structural options introduces a constraint on the interpretation of the Merge relation $R_\otimes$. Completeness for the calculi with structural options is then with respect to all frames respecting the relevant constraints (Došen 1992). The associative calculus $L$ in fact allows more specific models where we read $R_{xyz}$ as $x = y \cdot z$, with $\cdot$ a binary concatenation operation. Pentus (1995) presents a quite intricate proof that $L$ is indeed complete with respect to such free semigroup models (or language models, as they are also called).

- structural postulates:
  - $A \otimes B \vdash B \otimes A$ (commutativity)
  - $A \otimes (B \otimes C) = (A \otimes B) \otimes C$ (associativity)
- corresponding structural rules for the sequent presentation:
  \[
  \frac{W[(Y,X)] \Rightarrow C}{W[(X,Y)] \Rightarrow C} \quad \text{(comm.)} \quad \frac{W[(X,Y),Z] \Rightarrow C}{W[(X,Y,Z)] \Rightarrow C} \quad \text{(assoc.)}
  \]

- frame constraint for commutativity: $\forall xyz (R_{xyz} \implies R_{xzy})$
- frame constraint for associativity (rebracketing to the left): $\forall xyzrs (R_{xys} \text{ and } R_{xyz} \implies \exists t (R_{rtz} \text{ and } R_{txy}))$; similarly for the other direction

A characteristic property of $LP$ is the collapse of the distinction between left and right division: $A\setminus B$ and $B/A$ become interderivable. From a syntactic point of view, the global availability of associativity and/or commutativity in $LP$ would entail that arbitrary changes in constituent structure and/or word order cannot affect the well-formedness of an expression — a position that few linguists will subscribe to. Whether global associativity as we find it in $L$ is a viable option for syntax is open to dispute. Lambek points to cases of overgeneration resulting from free rebracketing in his 1961 paper, but later reverts to the associative view. In the following section, we will see that $LP$ is perfectly legitimate as a calculus of meaning composition. In §3, we discuss controlled forms of structural reasoning, anchored in lexical type assignment, as an alternative to the problematic all-or-nothing options.

2.2 The syntax-semantics interface

In order to give equal weight to syntactic and semantic considerations, we can set up a categorial grammar as a pair of calculi linked by a compositional interpretation: a syntactic source calculus, a semantic target calculus, and a homomorphism mapping types and derivations of the source to the corresponding types and derivations of the target. In this section, we work out this architecture for the standard Lambek calculi. In the following section, we will see how the extended categorial type logics reflect different views on the division of labour between syntax and semantics, and hence on the amount of work that has to be done to achieve a compositional mapping between the two.

The semantic study of categorial deduction started with van Benthem 1983. In line with the Curry-Howard ‘formulas-as-types, proofs-as-programs’ method, derivations in the various categorial calculi are associated with terms of suitable fragments of the lambda calculus. For the sake of simplicity, we restrict attention to the implicational vocabulary: the directional slashes for $(N)L$, and the non-directional implication of $LP$.

Let us start with the Curry-Howard correspondence for $LP$. The choice of $LP$ as the calculus of natural language semantics captures the fact that meaning
composition is a resource-sensitive process: in interpreting a derivation, no material can be copied or thrown away. Let us refer to the terms of the simply typed linear lambda calculus as $\Lambda(Lin)$. In the absence of Contraction and Weakening, this is the smallest set such that, given a denumerably infinite set of variables $\text{Var}$,

- $x \in \Lambda(Lin)$ if $x \in \text{Var}$;
- $\lambda x. M \in \Lambda(Lin)$ if $x \in \text{Var}, M \in \Lambda(Lin)$, and there is exactly one free variable occurrence of $x$ in $M$;
- $(M \ N) \in \Lambda(Lin)$ if $M \in \Lambda(Lin), N \in \Lambda(Lin)$, and the sets of free variables of $M$ and $N$ are disjoint.

From $\Lambda(Lin)$, one obtains $\Lambda(LP)$, the terms in Curry-Howard correspondence with $\text{LP}$ derivations, by adding one extra restriction, reflecting the requirement that sequent antecedents must be non-empty: every subterm contains at least one free variable. The derivations of $\text{LP}$ are now read as judgements of a type inference system for $\Lambda(LP)$. Derivations are given in the natural deduction style, with elimination and introduction rules for the linear implication $\rightarrow$.

Judgements then are sequents of the form $\Gamma \vdash M : B$. $\Gamma$ is a non-empty multiset of typing declarations $x_1 : A_1, \ldots, x_n : A_n$; the $A_i$ and $B$ are linear implicative types, and $M$ is a $\Lambda(LP)$ term of type $B$ built out of the $x_i$ of type $A_i$. Axiom sequents are of the form $x : A \vdash x : A$. Elimination and Introduction rules for the linear implication are given below. In the Introduction rule, $x \not\in \text{dom}(\Gamma)$; in the Elimination rule $\text{dom}(\Gamma) \cap \text{dom}(\Delta) = \emptyset$.

The syntactic calculi $(\text{NL})L$, similarly, are in Curry-Howard correspondence with directional linear lambda terms (Wansing 1992). There is no standard notation. In the directional typing rules below, we use $\lambda^\ell$ vs $\lambda^l$ for the image of the slash and backslash Introduction rules; in the case of the Elimination rules, we have right and left function application, with the triangle pointing to the function. The antecedent is now a (bracketed) string of typing declarations $x_i : A_i$.

\[
\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda^\ell x. M : B/A} \quad \frac{\Gamma \vdash M : A \rightarrow B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash (M \ N) : B} \quad \frac{\Gamma \vdash M : B/A \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash (M \triangleright N) : B} \quad \frac{\Gamma \vdash N : A \quad \Delta \vdash M : A\setminus B}{\Gamma, \Delta \vdash (N \triangleright M) : B}
\]

The source and target calculi each have their own set of basic types, motivated by syntactic and semantic considerations respectively. At the syntactic end, one could discriminate between sentences $s$, noun phrases $np$, common nouns $n$, for example. At the semantic end, for a simple extensional interpretation, one could introduce two basic types, $e$ and $t$. Set-theoretic interpretation for the semantic types is then given in terms of a non-empty set of individuals/entities $E$ and the truth values \{0,1\}. Denotation domains $\text{Dom}(A)$ for semantic types $A$ are set up such that $\text{Dom}(e) = E$, $\text{Dom}(t) = \{0,1\}$ and $\text{Dom}(A\setminus B) = \text{Dom}(B/A)$ is the set of functions from $\text{Dom}(A)$ to $\text{Dom}(B)$.

Consider next the homomorphic mapping from the syntactic source calculus to the semantic target calculus. The mapping is specified at the level of types and at the level of proofs (terms). Setting up the mapping at the level of atomic types as below, sentences denote truth values; (proper) noun phrases individuals; common nouns functions from individuals to truth values. For complex syntactic types, the interpretation function sends the two directional implications to the linear implication of the target calculus. At the level of proofs (terms), likewise, the directional constructs of the syntactic source calculus are identified in the interpretation. (We use a primed version for the target variable corresponding to a source variable, as a reminder of the fact that the type of these expressions is
In the previous sections, we have seen that the theorems of NL, L and LP form a proper inclusion hierarchy. The semantic counterpart is that in moving to more discriminating syntactic calculi, more and more LP terms are lost in translation: desirable recipes for meaning assembly are often unobtainable as the image of (NL)L proofs. For the three calculi, the table below gives a characteristic type transition. The second row has the directional proof terms associated with the (NL)L type transitions; the third row gives the corresponding LP term, obtained via the translation homomorphism, in the case of (NL)L, or directly for the LP case.

<table>
<thead>
<tr>
<th>NL: argument lowering</th>
<th>L: composition</th>
<th>LP: argument raising</th>
</tr>
</thead>
<tbody>
<tr>
<td>((B/(A\Lambda B))\Lambda C) \vdash A\Lambda C</td>
<td>(A\Lambda B \vdash (A\Lambda C)/(B\Lambda C))</td>
<td>(A \rightarrow (B \rightarrow C) \vdash)</td>
</tr>
<tr>
<td>(z \vdash \lambda x.((\lambda y.(x \rightarrow y)) \rightarrow z))</td>
<td>(y \vdash \lambda z.\lambda y.((x \rightarrow y) \rightarrow z))</td>
<td>(x \vdash \lambda w.\lambda z.((w \lambda y.((x \rightarrow y) \rightarrow z)))</td>
</tr>
<tr>
<td>(z' \vdash \lambda x'.(z' \lambda y'.(y' x')))</td>
<td>(y' \vdash \lambda z'\lambda x'.(z' (y' x')))</td>
<td>(x \vdash \lambda w.\lambda z.((w \lambda y.((x \rightarrow y) \rightarrow z)))</td>
</tr>
</tbody>
</table>

Argument lowering is valid in NL, hence also in L, LP. Function composition depends on associativity: it is invalid in NL, valid in L(P). Function composition has been used to build up partial semantic representations, running in parallel with the incremental (left-to-right) processing of a string. Argument raising, finally, is valid only in LP; i.e., this form of meaning assembly cannot be obtained as the image of a derivation in any of the syntactic calculi (NL)L. But, as we will see below, this transition plays an important role in the analysis of natural language scope ambiguities.

**Derivational versus lexical semantics.** The constraints imposed by resource-sensitivity put severe limitations on the expressivity of the derivational semantics. To some extent, these can be overcome at the level of lexical semantics. The Curry-Howard term associated with a derivation, seen as a program for meaning assembly, abstracts from the contribution of particular lexical items. The semantic specification for a lexical item of type \(A\) takes the form of a lambda term of the corresponding semantic type \(A'\), but this lambda term is no longer required to be linear. Below some examples of non-linear lexical meaning recipes. The specification for the reflexive pronoun ‘himself’ is a pure combinator (a closed term); it identifies the first and second argument of a binary relation. The terms for the relative pronoun ‘that’ or for the determiners ‘a, some’ have an abstraction that binds two occurrences of a variable, so as to compute the intersection of their two \((e \rightarrow t)\) arguments (noun and verb phrase), and to test the intersection for non-emptiness in the case of ‘some’.

<table>
<thead>
<tr>
<th>NL type</th>
<th>LP type</th>
<th>Lexical recipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>himself</td>
<td>(((np\Lambda s)/(np\Lambda s)))</td>
<td>(e \rightarrow e \rightarrow t)</td>
</tr>
<tr>
<td>a, some</td>
<td>((s/(np\Lambda s)))</td>
<td>((e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow t)</td>
</tr>
<tr>
<td>that</td>
<td>((n\Lambda n)/(np\Lambda s))</td>
<td>((e \rightarrow t) \rightarrow (e \rightarrow t) \rightarrow e \rightarrow t)</td>
</tr>
</tbody>
</table>

The interplay between lexical and derivational aspects of meaning assembly is illustrated with the example below. Notice that the linear proof term reflects the derivational history (modulo directionality); after substitution of the lexical recipes and \(\beta\) conversion this transparency is lost. The full encapsulation of lexical semantics is one of the strong attractions of the categorial approach.

http://plato.stanford.edu/entries/typological-grammar/
Typological Grammar (Stanford Encyclopedia of Philosophy)

- string: ‘a boy hurt himself’
- NL derivation: \((x : (s/np\backslash s)/n,y : n),(z : ((np\backslash s)/np, w : ((np\backslash s)/np)\backslash (np\backslash s))) \vdash M : s\)
- corresponding LP proof term: \((M)' = ((x' \ y') (w' \ z'))\)
- lexical insertion: \(x' \mapsto \lambda P. Q. (\exists \lambda x. ((P \ x) \land (Q \ x))), y' \mapsto \text{boy}, z' \mapsto \text{hurt}, w' \mapsto \lambda R. x. ((R \ x) \ x)\)
- final result after \(\beta\) conversion: \((\exists \lambda x. ((\text{boy} \ x) \land (\text{hurt} \ x \ x)))\)

3. Extended typelogical grammars

The strategy of combining resource-sensitivity at the level of derivational semantics with non-linear lexical meaning assignments is not a general solution for the expressive limitations of the syntactic calculi originally conceived by Lambek. A type of problem that necessitates extensions of the derivational machinery itself has to do with the inability of \((\text{N})\text{L}\) to establish discontinuous dependencies between grammatical resources in a satisfactory way. Such dependencies manifest themselves in two situations.

**Extraction.** These are phenomena which generative grammar refers to as ‘overt displacement’, as we find it in \(wh\) ‘movement’ constructions. Compare the \(wh\) phrases ‘what — annoys Mary’ versus ‘what Mary put — there’, as they would occur in a context ‘I know what annoys Mary’ or ‘I know what Mary put there’. The dash marks the spot where a noun phrase would fit in a normal declarative sentence: ‘this record annoys Mary’, ‘Mary put the record there’.

Assigning the \(wh/(np\backslash s)\) to ‘what’, we can derive \(wh\) for the first example, with a \(np\) hypothesis in the left-peripheral subject position. If this hypothesis (the ‘gap’) occurs internally, as in the second example, it is unreachable for \((\text{N})\text{L}: one cannot derive \(np\backslash s\) for ‘Mary — put there’ (nor \(s/np\), for that matter, which would require a right-peripheral gap).

**Infraction.** Typical examples would be cases of non-local semantic construal, as in ‘Alice knows someone is cheating’, with a wide-scope reading for ‘someone’: ‘there is a particular person \(x\) such that Alice thinks \(x\) is cheating’. The quantifier phrase ‘someone’ occupies the structural position where in fact the \(np\)-type hypothesis is needed, and realizes its semantic effect at some higher sentential level. With a type-assignment \(s/(np\backslash s)\) for ‘someone’, there is no derivation producing the non-local reading. In generative grammar, one would call the non-local construal an instance of ‘covert’ movement.

The extensions of the basic calculus to be discussed in the sections below represent the strategies that are being pursued to find principled solutions to these problems of discontinuous dependencies.

3.1 Multimodal systems, structural control

The key idea of Multimodal Categorial Grammars, introduced in the 1990s, is simple: instead of a single family of residuated type-forming operations, one considers multiple families, co-existing in one logic. To discriminate between these families, type-forming operations are decorated with a mode index \(i, \otimes, \odot\); likewise for the interpreting Merge relations \(R_i\). Each family has the same logical rules (the residuation laws). But they can differ in their structural properties. In particular, they can interact in terms of structural rules that mix different modes. At the level of the interpretation, such interaction principles introduce frame constraints, similar to the constraints that went with the move from NL to L and LP.

For an example, let us return to the case of non-peripheral extraction ‘(what) Mary put — there’. Assume we have two modes of composition: \(\otimes_c\) and \(\otimes_d\). Regular valency requirements are expressed in terms of the slashes for the \(\otimes_c\) product. Discontinuous dependencies are handled by structural postulates of mixed associativity and commutativity, and an inclusion postulate expressing the relative ‘strength’ of \(\otimes_c\) and \(\otimes_d\). The type-assignment to the pronoun ‘what’ provides access to these postulates, so that it can establish a link with a non-subject \(np\) hypothesis, also in a sentence-internal position.
mixed associativity: \((A \otimes_c B) \otimes_d C \vdash A \otimes_c (B \otimes_d C)\)

mixed commutativity: \((A \otimes_c B) \otimes_d C \vdash (A \otimes_d C) \otimes_c B\)

inclusion: \(A \otimes_d B \vdash A \otimes_c B\)

type for ‘what’ related to a non-subject \(np\) hypothesis: \(\text{wh}_c(s/d np)\)

We can now successfully demonstrate that the phrase ‘Mary put — there’ is of type \(s/d np\), as shown in the sketch of a derivation below. By means of the mixed interaction postulates, the \(np\) hypothesis finds its way to the non-peripheral position where it is required by the verb ‘put’. Once it has reached its home position, the inclusion step switches from the structurally permissive mode \(d\) to the \(c\) mode expressing regular subcategorization.

Through multimodal interaction principles, one avoids the overgeneration that would result from global Associativity and/or Commutativity options for a single composition operation \(\otimes\). Undisciplined use of multimodality, however, could easily lead to the introduction of construction-specific mode distinctions, thus compromising one of the main attractions of the typological approach. The theory of structural control operators to be discussed below offers a more principled solution to the type of problem multimodal approaches address.

**Control operators.** Linear Logic, by dropping the structural rules of Contraction and Weakening, splits the conjunction ‘\&’ of classical logic in a multiplicative and an additive connective. This move to a resource-sensitive logic can be obtained without sacrificing expressivity: the unary modalities ‘!’; ‘?’ allow one to reintroduce Contraction/Weakening in a controlled form and thus to recover classical logic from within the linear setting.

A similar strategy can be followed within typological grammar (Moortgat 1996, Kurtonina 1995). The type language is extended with a pair of unary operators which we will write \(\Diamond, \Box\); formulas now are built out of atomic formulas \(p\) with the unary and binary type-forming operations. As with the binary vocabulary, we start with a minimal logic for \(\Diamond, \Box\); facilities for structural control can then be added in the form of extra postulates. The truth conditions below characterize the control operators \(\Diamond\) and \(\Box\) as inverse duals with respect to a binary accessibility relation \(R_\Diamond\). This interpretation turns them into a residuated pair, just like composition and the left and right division operations.

- Formulas: \(A, B ::= p \mid \Diamond A \mid \Box A \mid A\Box A \mid A \otimes B \mid A/B\)
- Interpretation:
  - \(x \vdash \Diamond A\) iff \(\exists y(R_\Diamond xy \text{ and } y \vdash A)\)
  - \(x \vdash \Box A\) iff \(\forall y(R_\Box yx \text{ implies } y \vdash A)\)
- Residuation laws: \(\Diamond A \vdash B\) iff \(A \vdash \Box B\)

We saw that in the minimal logic \(\text{NL}\) completeness with respect to the frame semantics for composition and its residuals doesn't impose restrictions on the interpretation of the merge relation \(R_\otimes\). The same holds for \(R_\Diamond\) in the pure residuation logic of \(\Diamond, \Box\). From the residuation laws, one easily derives monotonicity of
the control operators \((A \vdash B \text{ implies } \lozenge A \vdash \lozenge B \text{ and } \Box A \vdash \Box B)\); their compositions satisfy \(\lozenge \Box A \vdash A \text{ and } A \vdash \lozenge \Box A\). These properties can be put to use in refining lexical type assignment so that \textit{selectional dependencies} are taken into account. Compare the effect of an assignment \(A/B\) versus \(A/\lozenge \Box B\). The former will produce an expression of type \(A\) in composition both with expressions of type \(B\) and \(\lozenge \Box B\), the latter only with the more specific of these two, \(\Box B\). An expression typed as \(\Box \lozenge B\) will \textit{resist} composition with either \(A/B\) or \(A/\lozenge \Box B\). Bernardi and Szabolcsi (2008) present accounts of licensing of polarity-sensitive expressions and scope construal based on (a generalization of) this modalisation strategy.

For sequent presentation of \(\lozenge, \Box\), antecedent tree structures now are built out of formulas by means of unary and binary tree-building operations, \((-\) and \((-,-)\). The residuation pattern then gives rise to the following left and right introduction rules. Cut elimination carries over straightforwardly to the extended system, and with it decidability and the subformula property.

\[
\begin{align*}
\Gamma[(A)] & \Rightarrow B & B \Rightarrow L & \Gamma \Rightarrow A & \lozenge R & \Gamma \Rightarrow A & \Box L & \Gamma \Rightarrow [\Box A] & \Rightarrow B & \Box R & \Gamma \Rightarrow A & \Box R \\
\end{align*}
\]

**Controlling structural resource management.** Let us turn then to the use of \(\lozenge, \Box\) as control devices. The control can take two forms: one is to \textit{license} access to a structural option that would be overgenerating if globally available; the second is to occasionally \textit{block} a structural capability of the grammar. For the licensing type of control, compare the multimodal treatment of extraction in terms of a distinction \(\otimes_c\) vs \(\otimes_d\), with the \(\lozenge\)-controlled version below, which relies on a single binary composition operation, and narrows down the ‘movement’ potential entirely to \(\lozenge\)-marked resources.

- \(\lozenge\)-controlled mixed associativity: \((A \otimes B) \otimes C \vdash A \otimes (B \otimes C)\)
- \(\lozenge\)-controlled mixed commutativity: \((A \otimes B) \otimes C \vdash (A \otimes (C \otimes B))\)
- type for ‘what’ related to a non-subject \(np\) hypothesis: \(wh/(s/\lozenge \Box np)\)

The derivation sketch below illustrates the interplay between logical and structural reasoning. As long as the gap subformula \(\lozenge \Box np\) carries the licensing \(\lozenge\), the structural rules are applicable; as soon as it has found the appropriate structural position where it is selected by the transitive verb, it can be used as a regular \(np\), given the type transition \(\lozenge \Box np \vdash np\) which obviates the need for the inclusion postulate of the earlier account. Related analyses of extraction based on the ‘!’ modality of Linear Logic can be found in (Barry et al. 1991), with the dereliction rule \(!A \vdash A\) taking the place of \(\lozenge \Box A \vdash A\) for gap introduction.

\[
\begin{align*}
np \otimes (((((np \Box s)/pp)/np) \otimes np) \otimes pp) & \vdash s & \text{ since } \lozenge \Box A \vdash A \\
& \text{ \(\lozenge\) mixed comm.} & \lozenge \text{ \(\Box\) mixed assoc.} \\
np \otimes (((((np \Box s)/pp)/np) \otimes np) \otimes pp) & \otimes np \vdash s \\
(np \otimes (((((np \Box s)/pp)/np) \otimes pp)) \otimes np) & \vdash s \\
np \otimes (((((np \Box s)/pp)/np) \otimes pp)) & \otimes np \vdash s \\
\end{align*}
\]

Mary put there

For the type of control where a structural transformation has to be \textit{blocked}, we can turn to extraction islands: phrases that do not allow a gap, such as adjuncts. Compare ‘Mary fell asleep during the concert’, with the adjunct phrase ‘during the concert’ of type \(s/\Box\), with the ill-formed ‘I know what Mary fell asleep during \(-\)‘. To make the adjunct inaccessible, one can assign ‘during’ the type \((\Box(\Box s))/np)\) for the \(\Box\) to be eliminated, the whole adjunct phrase has to be enclosed by a structural \(\lozenge\), which then acts as a barrier for extraction. Without such a barrier, the extraction postulates as formulated above would allow the ill-formed sentence to be derived. This strategy of projecting island barriers from lexical type assignments was introduced by Morrill; see the discussion of ‘structural inhibition’ in
Morrill 1994.

The two uses of the control operators illustrated here give rise to a general theory of substructural communication relating the different type logics in the categorial landscape. Let Source and Target be neighbouring logics differing in some structural option, for example NL versus L, or L versus LP. Kurtonina and Moortgat (1997) establish a set of embedding theorems of the following form:

- translation: $\mu : \text{Source}(/, \otimes, \setminus) \to \text{Target}(\Diamond, \Box, /, \otimes, \setminus)$, such that $A \vdash B$ is derivable in the Source logic iff $\mu(A) \vdash \mu(B)$ is derivable in the Target logic

In the case where the Target logic is more discriminating than the Source, the translation implements control of the licensing type. The other direction of communication obtains when the Target system is less discriminating than the Source. The modal decoration effected by the translation in this case blocks the applicability of structural rules.

**Frame constraints, term assignment.** The frame semantics for the pure residuation logic does not impose restrictions on the interpretation of the $R_\Diamond$ and $R_\Box$ relations. In the case of extended versions with modally controlled structural postulates, there is a frame constraint for each structural postulate, and completeness holds with respect to appropriately restricted frames.

```
   z                 z
     |                 |
   x   y   z   y   t'   x   t'
  \ /   \ /   \ /   \ /   \ /   \ /
 s   t   x   s'   s'   y
  \ /   \ /   \ /   \ /   \ /   \ /
 r   r   r   r
```

Depicting $R_\otimes$ with a branching and $R_\Diamond$ with a non-branching node, in the case of the modally controlled extraction postulates discussed here, we have the constraint that for all $x,y,z,r,s,t$ composed as in the input configuration below, there are alternative internal points $s',t'$ connecting the root $r$ to the leaves $x,y,z$.

For the mapping between the modally extended syntactic source calculus and the semantic target calculus LP, we have two options. The first is to treat $\Diamond, \Box$ purely as syntactic control devices. One then sets $(\Diamond A)' = (\Box A)' = A'$, and the inference rules affecting the modalities leave no trace in the LP term associated with a derivation. The second is to actually provide denotation domains for the new types, and to extend the term language accordingly. For the minimal logic of $\Diamond, \Box$, one can turn to Wansing (2002), who develops a set-theoretic interpretation of minimal temporal intuitionistic logic. The temporal modalities of future possibility and past necessity are indistinguishable from the control operators $\Diamond, \Box$, prooftheoretically and as far as their relational interpretation is concerned. Morrill (1990) gives an interpretation in intensional lambda calculus for a stronger S5 modality, which assumes a universal accessibility relation at the level of the frame semantics. Denotations for modalized formulas $A$, in this setting, are functions from indices (situations, reference points) to $A$ denotations.

**Discussion.** The control operators discussed here bear a strong resemblance to the syntactic control features used in Stabler's algebraic formulation of Minimalist Grammars (Stabler 1997, 1999). These features, like $\Diamond, \Box$, come in matching pairs of a licensor ‘+$t$’ and a licensee ‘$-t$’ that have to cancel for a
derivation to be successful; movement is triggered by the presence of an appropriate licensor feature. For an explicit comparison of the minimalist and the typological view on structural control, see Vermaat (2004).

Whether one wants the full expressivity of modally controlled structural rules is a matter of debate. Vermaat (2006) takes the strong position that the cross-linguistic possibilities for displacement can be fully captured in terms of the right branch extraction postulates discussed above, together with their mirror images for extraction from left branches. Under this view, which is consistent with the radical lexicalism of the original categorial systems, this postulate set is fixed at the level of Universal Grammar, and variation is reduced to the language-specific lexica that determine which of the available extraction patterns are activated.

### 3.2 The logic of discontinuity

The discontinuous Lambek calculi that have been developed by Morrill and co-workers (see Morrill et al. 2007, 2009) and Chapter 6 of the forthcoming monograph (Morrill 2010) extend the associative Lambek calculus $L$. We saw that $L$ is the logic of strings composed by concatenation. The discontinuous calculi enrich the ontology with a notion of *split* strings: expressions consisting of detached parts, as in the idiom ‘take — to task’. To build the phrase ‘take someone to task’, one *wraps* the discontinuous expression around its object. In this particular example, there is a single point of discontinuity, but one can also think of cases with more than one split point. This naturally leads to two notions of discontinuous composition: a deterministic view, where wrapping targets a particular split point, and a non-deterministic view where the targeted split point is arbitrary. In the case of expressions with a single split point, these two notions coincide.

The vocabulary of $DL$ (Discontinuous Lambek Calculus) consists of residuated families of unary and binary type-forming operations. A representative sample is given below. For the binary case, in addition to the concatenation product of $L$ and the residual slash operations, we have a discontinuous (wrapping) product $\odot$, with residual infixation $\downarrow$ and extraction $\uparrow$ operations. For the deterministic interpretation, the discontinuous type-forming operations have an indexed form $\uparrow_k$, $\odot_k$, $\downarrow_k$ explicitly referring to the $k$-th split point of their interpretants. The function of the unary operations is to control the creation and removal of split points.

As in the binary case, we have non-deterministic operations (bridge $\wedge$, split $\lor$) and indexed forms ($\wedge_k, \lor_k$) with a deterministic interpretation.

$$A,B ::= \ldots \mid A \odot B \mid A \downarrow B \mid B \uparrow \mid A \wedge A \lor A$$

On the model-theoretic side, an innovative feature of $DL$ is the move to a *multi-sorted* interpretation. The key notion is that of a graded algebra: a freely generated algebra $(W, \cdot, 1, \_)$ where the monoid $(W, \cdot, 1)$ of the interpretation of $L^*$ is augmented with a distinguished generator called the separator. The *sort* of an expression $s$, $\sigma(s)$, is given by the number of occurrences of the separator in it. Expressions of the nullary sort are the familiar strings for the language models of $L$. Expressions of sort $n > 0$ are split strings, with $n$ marked positions where other expressions can be substituted.

Interpretation of types is now relativized to sorted domains $W_i = \{s \mid \sigma(s) = i\}$ for $i \geq 0$. Frames, accordingly, are sorted structures $\{(W_i)_{i \in N} \cdot R_{bridge} \cdot R_{wrap}, \{bridge_k\}_{k \in N^+}, \{wrap_k\}_{k \in N^+}\}$ with $n + 1$-ary relations for the $n$-ary type-forming operations with a non-deterministic interpretation and $n$-ary operations (functions) for the $n$-place deterministic vocabulary items. The operation $\cdot : W_i \times W_j \rightarrow W_{i+j}$ here is the sorted version of the concatenation operation of $L$.

<table>
<thead>
<tr>
<th>relation/operation</th>
<th>interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{\text{wrap}} \subseteq W_{i+1} \times W_j \times W_{i+j}$</td>
<td>the smallest relation s.t. $R_{\text{wrap}}(u \wedge w, v, uvw)$</td>
</tr>
<tr>
<td>$R_{\text{bridge}} \subseteq W_{i+1} \times W_i$</td>
<td>the smallest relation s.t. $R_{\text{bridge}}(u \wedge v, uv)$</td>
</tr>
<tr>
<td>$\text{wrap}<em>k : W</em>{i+1} \times W_j \rightarrow W_{i+j}$</td>
<td>$\text{wrap}_k(s,t)$ is the result of replacing the $k$-th separator in $s$ by $t$</td>
</tr>
</tbody>
</table>
bridge_k : W_{i+1} \rightarrow W_i  \quad \text{bridge}_k(s) \text{ is the result of replacing the } k\text{-th separator in } s \text{ by } 1

An interpretation for DL associates atomic types of sort } i \text{ to subsets of } W_i. \text{ Interpretation clauses for the new complex types are standard. In the light of the illustrations below, we give the clauses for non-deterministic bridge/split, and the wrapping family. The sort of the types can be readily computed from the sort information for the interpreting operations/relations.}

- s ⊨ ∧A iff ∃(R_{bridge}(t,s) \text{ and } t ⊨ A)
- t ⊨ ∨B iff ∀s(R_{bridge}(t,s) \text{ implies } s ⊨ B)
- s ⊨ A \otimes B iff ∃s''(R_{wrap}(s'',s''') \text{ and } s'' ⊨ A \text{ and } s''' ⊨ B)
- s'' ⊨ A ↓ C iff ∀s''s(\text{if } R_{wrap}(s'',s''') \text{ and } s'' ⊨ A, \text{ then } s ⊨ C)
- s'' ⊨ C ↑ B iff ∀s''s(\text{if } R_{wrap}(s'',s''') \text{ and } s'' ⊨ B, \text{ then } s ⊨ C)

On the proof-theoretic side, a cut-elimination theorem for a sequent presentation of DL establishes decidability. The DL sequent rules are shown to be sound with respect to the intended interpretation; a completeness result so far has not been obtained. For the mapping from the syntactic source calculus DL to the semantic target calculus LP, the unary type-forming operations are considered inert: the inference rules for these connectives, consequently, leave no trace in the LP proof term associated with a derivation in the syntactic source calculus. The continuous and discontinuous families, for the rest, are treated exactly alike. Specifically, the infixation and extraction operations are mapped to LP function types, like the slashes.

**Illustration.** DL has been successfully applied to a great number of discontinuous dependencies, both of the overt and of the covert type. The non-deterministic operations have been used to model particle shift and complement alternation constructions. The deterministic operations of sort 1 (single split point) are used in the analysis of non-peripheral extraction, discontinuous idioms, gapping and ellipsis, quantifier scope construal, reflexivisation, pied-piping and the Dutch cross-sentential dependencies, among others.

We illustrate the non-deterministic use of DL with English particle shift, using labeled natural deduction format to display derivations, with items *Form-Meaning-Type*. A verb-particle combination ‘call — up’ can be lexically typed as 𝜔(np\#s) ↑ np of sort 2, with an internal split point, and a right-peripheral one. Elimination of the non-deterministic extraction operation ↑ offers a choice as to whether wrapping will affect the first or the second split point. The first option is displayed below. The remaining separator is removed in the elimination step for ∧, with the continuous verb phrases ‘call Mary up’ or ‘call up Mary’ as result.

\[
\begin{align*}
\text{called · Mary · up · phone : } & \omega(np\#s) \uparrow np \quad \text{Mary} - m : np \quad E\uparrow \\
& \quad \text{called · Mary · up · (phone } m) : \omega(np\#s) \quad E\wedge \\
& \quad \text{called · Mary · up · (phone } m) : np\#s
\end{align*}
\]

For an example involving covert discontinuity, consider quantifier scope construal. DL provides a uniform type assignment to generalized quantifier expressions such as ‘everyone’, ‘someone’: (s ↑ np) ↓ s. In the syntactic source calculus, this type assignment allows a quantifier phrase QP to occupy any position that could be occupied by a regular non-quantificational noun phrase. Semantically, the image of the ↑ Introduction rule at the level of the semantic target calculus LP binds an np type hypothesis at the position that was occupied by the QP (the a – x : np premise, with a and x structural and semantic variables for the np hypothesis). The image of the ↓ Elimination rule applies the term representing the QP meaning to this abstract. Scope ambiguities arise from derivational ambiguity in the source calculus DL. The derivation below results in a non-local reading ‘there is a particular x such that Mary thinks x left’. Looking upward from the conclusion, the last rule applied is ↓ Elimination, which means the QP takes scope at the main clause level. An alternative derivation, producing the local scope reading, would have the / Elimination rule for ‘thinks’: (np\#s)/s as the last step.
\[ \ldots a \rightarrow x : np \ldots \]
\[
\vdots
\]
\[
\text{Mary} \cdot \text{thinks} \cdot a \cdot \text{left} - ((\text{thinks} (\text{left} x)) m) : s
\]
\[
\text{someone} - \exists : (s \uparrow np) \downarrow s \text{Mary} \cdot \text{thinks} \cdot \_ \cdot \text{left} - \lambda x.((\text{thinks} (\text{left} x)) m) : s \uparrow np \downarrow E
\]

**Discussion.** The basis for the DL extensions is the associative calculus L. As we saw above, a global insensitivity to phrase structure is a source of overgeneration, unless blocked by explicit island modalities. In the development of DL nothing seems to hinge on the associativity of the base system: it would seem entirely feasible, in other words, to develop DL as an extension of a non-associative basis, allowing for the representation of constituent-structure information. In the interpreting frames, one would then start from a graded *groupoid* rather than a monoid. The fact that the DL framework can readily accommodate a string or tree-based point of view testifies to the versatility of the approach.

### 3.3 Symmetric categorial grammar

The extensions of the Syntactic Calculus that we studied in the previous sections all obey an “intuitionistic” restriction: in statements \( A_1 \ldots A_n \vdash B \) the antecedent can consist of multiple formulas (configured as a \( \bigotimes \) tree in the non-associative case, a list or a multiset in the case of L, LP), the succedent is a single formula.

In a remarkable paper antedating Linear Logic by five years, Grishin (1983) presents a number of extensions of the Lambek calculus with the common characteristic that derivability holds between hypotheses \( A_1 \ldots A_n \), taken together by means of a multiplicative conjunction, and *multiple* conclusions \( B_1 \ldots B_m \), combined by means of a multiplicative disjunction. Linguistic exploration of Grishin’s ideas has started in recent years. In this section, we introduce a multiple-conclusion categorial system that is already well-studied: LG (for Lambek-Grishin calculus). We present syntax and relational semantics of the system; in the next section, we turn to its computational interpretation.

| Preorder laws | \( A \vdash A \) | \( A \vdash B \quad B \vdash C \) | \( A \vdash C \) |
| (dual) Residuation laws | \( A \otimes B \vdash C \) | \( B \vdash A \) | \( C \vdash B \otimes A \) | \( C \otimes A \vdash B \) |
| Interaction principles | \( A \otimes B \vdash C \oplus D \) | \( A \oplus B \vdash C \oplus D \) | \( C \oplus A \vdash D \) | \( B \otimes D \vdash A \setminus C \) |

The Lambek-Grishin calculus **LG**

**LG** consists of a symmetric version of the pure residuation logic **NL** together with structure-preserving interaction principles relating the conjunctive and disjunctive operations. We discuss these components in turn. For symmetry, the inventory of type-forming operations is doubled: in addition to the **NL**
operations $\otimes, \setminus, \backslash$ (product, left and right division), there is a second family $\oplus, \otimes, \oslash$: coproduct, right and left difference. The two families are related by an arrow reversal symmetry $\delta$, which translates type formulas according to the table below.

$$
\begin{array}{c|ccc}
\delta & C/B & A \otimes B & A \oslash C \\
\hline
B \otimes C & B \oplus A & C \ominus A
\end{array}
$$

At the level of derivability, we then have $A \vdash B$ iff $\delta(B) \vdash \delta(A)$: for every theorem or rule of NL, we also find its image under $\delta$ in LG. A note about the notation: we read $B \oslash A$ as ‘$B$ from $A$’ and $A \ominus B$ as ‘$A$ less $B$’, i.e., the quantity that is subtracted is put under the circled (back)slash, just as we have the denominator under the (back)slash in the case of left and right division types. In a formulas-as-types spirit, we will feel free to refer to the division operations also as implications, and to the difference operations as coimplications.

Communication between the product and coproduct families requires the addition of interaction principles to the (dual) residuation laws. The principles above take the form of inference rules obtained from the following recipe: from $A \otimes B \vdash C \oplus D$ in the premise, one selects a product and a coproduct term; in the conclusion, one simultaneously introduces the residual operations for the remaining two terms. Using the (dual) residuation laws, one derives the following patterns from the interaction principles. Alternatively, one can take (P1)–(P4) as primitive postulates, and obtain the interaction principles as derived inference rules, using transitivity and the (dual) residuation laws.

(P1) $$(A \otimes B) \otimes C \vdash A \otimes (B \otimes C)$$
(P2) $$C \otimes (A \otimes B) \vdash A \otimes (C \otimes B)$$
(P3) $$C \otimes (B \otimes A) \vdash (C \otimes B) \oplus A$$
(P4) $$(B \otimes A) \otimes C \vdash (B \otimes C) \oplus A$$

Derivability patterns of the form (P1)–(P4) have been called linear distributivity principles — linear, because they do not duplicate any of the terms involved. In LG, in addition to being linear, they respect the word order and phrase structure information encoded in the non-commutative, non-associative type-forming operations.

**Illustration.** The interaction principles of LG provide a particularly direct way of capturing phenomena that depend on infixation rather than concatenation. The derivations below (using monotonicity, and (dual) residuation steps) show how the same pieces of information (the same premises) can be used either to introduce an implication $B \setminus C$ on the left of the turnstile, or a coimplication $B \oslash C$ on the right. The first option leads to a rule of Application — the central composition operation of standard Lambek calculus. The second option leads to a Co-Application variant. Although these two rules are derived from the same premises, there is an important difference between them. When the implication $B \setminus C$ composes with its argument, it must stay external to $X$. In the case of the coimplication, when $X$ is some product of factors $A_1, \ldots, A_n$, the conditions for the interaction principles are met. This means that the coimplication $B \oslash C$ will be able to descend into the phrase $X$ and associate with any of the constituent parts $A_i$ into a formula $(B \oslash C) \oslash A_i$.

$$
\frac{X \vdash B \quad C \vdash Y}{B \setminus C \vdash XY} \quad \frac{X \vdash B \quad C \vdash Y}{X \setminus Y \vdash B \oslash C} \\
\frac{X \otimes (B \setminus C) \vdash Y}{X \otimes Y \vdash (B \setminus C) \oslash Y} \quad \frac{X \vdash (B \oslash C) \oslash Y}{X \vdash \oslash (B \oslash C) \oslash Y}
$$

In general, an expression of type $(B \oslash C) \oslash A$ behaves locally as an $A$ within a context of type $B$; it then acts as a function transforming $B$ into $C$. We illustrate

http://plato.stanford.edu/entries/typological-grammar/
with the example of non-local scope construal for which we have seen the DL analysis in the previous section. The key point is the lexical type assignment to the generalized quantifier expression, instantiating \((B \otimes C) \otimes A\) as \((s \otimes s) \otimes np\). The semantic interpretation of this derivation will be discussed below.

\[
\begin{align*}
\text{np} \otimes ((\text{np}s)/s) \otimes (\text{np}s)) & \vdash s, s \vdash s \\
\text{np} \otimes ((\text{np}s)/s) \otimes ([np] \otimes (np)s)) & \vdash (s \otimes s) \oplus s \\
\vdots \\
\text{np} \otimes ((\text{np}s)/s) & \otimes ((s \otimes s) \otimes np)) \vdash s
\end{align*}
\]

Alice  thinks  someone  left

Completeness, decidability. Relational models for LG are given in terms of two interpreting relations: \(R_{\ominus}\) for multiplicative conjunction (merge, fusion), and \(R_{\oplus}\) for multiplicative disjunction (fission). The truth conditions for co-product and the difference operations are given below.

- \(x \vdash A \ominus B \iff \forall yz (\text{if } R_{\ominus}xyz, \text{then either } y \vdash A \text{ or } z \vdash B)\)
- \(y \vdash C \ominus B \iff \exists xz (R_{\ominus}xyz \text{ and not-(} z \vdash B \text{) and } x \vdash C)\)
- \(z \vdash A \ominus C \iff \exists xy (R_{\ominus}xyz \text{ and not-(} y \vdash A \text{) and } x \vdash C)\)

Completeness of LG with respect to this interpretation is proved in Kurtonina and Moortgat 2010. The minimal symmetric system (without interaction principles) has fission \(R_{\ominus}\) and merge \(R_{\oplus}\) as distinct relations, without imposing restrictions on their interpretation. In the presence of the interaction principles, their interpretation is related in terms of frame constraints. The distributivity principle \((A \ominus B) \otimes C \vdash A \ominus (B \otimes C)\), for example, corresponds to the constraint that for every \(x,y,z,w,v\), if we have a configuration \(R_{\ominus}xyz\) and \(R_{\oplus}wvy\), then there exists an alternative internal point \(t\) such that \(R_{\ominus}twx\) and \(R_{\oplus}tvz\). For decidability, Moortgat (2009) gives a sequent presentation of LG in the format of a Display Calculus allowing cut-free proof search.

Discussion. LG is a weaker logic than CNL, the Classical Non-Associative Lambek Calculus of de Groote and Lamarche 2002. In the latter system, like in classical linear logic, we have an involutive negation and De Morgan dualities turning multiplicative conjunction (times) and disjunction (par) into interexpressible operations. With respect to the linguistic applications, one can compare LG with the Discontinuous Calculus of §3.2. Whereas the latter provides uniform analyses of both the extraction and the infixation type of discontinuities, the distributivity principles of LG are primarily addressing the infixation variety. It may be interesting in this respect to note that Grishin 1983 proposes a second group of distributivity principles — converses of the ones discussed above. Whether these could play a role in the analysis of overt displacement remains to be seen. From a formal point of view, each of the groups of distributivities is a conservative extension of the basic symmetric calculus. But the combination of the two (i.e., the distributivity principles as invertible rules) induces partial associativity/commutativity of the (co)product operations, i.e., structure-preservation is lost.

3.4 Flexible interpretation, continuation semantics

In the previous sections, the emphasis was on extending the capabilities of the syntactic source calculi. In this section, we look at developments that put more structure in the mapping between the source and target calculi.

The syntax-semantics mapping, as discussed in §2.2, is rigid: once we have determined its action on the atomic types of the syntactic source calculus, everything is fixed. This rigidity is problematic as soon as we have a class of syntactic expressions with a non-uniform contribution to meaning assembly. Noun phrases are a case in point. Mapping \(np\) to semantic type \(e\) is appropriate for proper nouns and definite descriptions whose denotations can be taken to be individuals. But for quantifying expressions like ‘someone’, ‘no student’, an interpretation of type \(e\) is inadequate. The rigid syntax-semantics mapping thus forces one to assign these expressions a higher-order type in the syntax, for example \(s/(np)s\), so as to obtain a semantic type with the right kind of denotation as

http://plato.stanford.edu/entries/typological-grammar/
the translation image, \((e \rightarrow t) \rightarrow t\). If one wants to avoid this semantically motivated complication of the syntax, one could set \(np' = (e \rightarrow t) \rightarrow t\). But now the effect will be that simple transitive verbs \((np's)/np\), which one would like to treat as binary relations \(e \rightarrow e \rightarrow t\) by default, are associated with third-order interpretations instead.

**Flexible interpretation.** An attractive alternative to this rigid view of the syntax-semantics interface, is the flexible interpretation of Hendriks 1993. In this approach, a type of the syntactic source calculus is associated with an infinite set of \(LP\) types: one of these is the default semantic type associated with the syntactic source type, the others are derived from this default by means of type shifting laws. Similarly, a source calculus term is translated into an infinite set of target semantic terms; one of these is the basic translation (not necessarily of the default semantic type for the relevant syntactic category), the others are derived by the Curry-Howard term image of the typing shifting laws. The type shifting principles used are (i) argument lowering, (ii) value raising, and (iii) argument raising. The first two correspond to valid type transitions of \(NL\); argument raising is the crucial principle that goes beyond the expressivity of the syntactic source calculi, as we saw in §2.2.

As an example of flexible interpretation, compare ‘John loves Mary’ and ‘Everyone loves someone’. Syntactically, all noun phrases are typed as \(np\), and the verb as \((np's)/np\). ‘John’ and ‘Mary’ are interpreted with constants of type \(e\), ‘loves’ with a constant of type \(e \rightarrow e \rightarrow t\), the default semantic types for the syntactic source types. But ‘everyone’ and ‘someone’ are mapped to constants of type \((e \rightarrow t) \rightarrow t\). Interpreting the syntactic derivation for ‘Everyone loves someone’, the semantic types no longer match for simple function application: the verb expects two \(e\) type arguments, but instead finds two arguments of type \((e \rightarrow t) \rightarrow t\). There are two ways of accommodating these higher-order arguments by means of argument raising: depending on whether one raises first the object argument and then the subject, or vice versa, one obtains two readings for one and the same syntactic derivation: a reading corresponding to the surface order, with \(\forall\) taking scope over \(\exists\), and a reading with the inverted scope order. The correspondence between syntax and semantics, under this flexible view, has become a relation, rather than being functionally determined by the translation homomorphism of §2.2.

**Continuations.** It has become clear in recent years, that the core ideas of Hendriks' type-shifting account of derivational ambiguity can be insightfully recast in terms of a continuation-passing-style interpretation (CPS), as developed within computer science, and the different evaluation strategies available for such interpretation. The use of continuations in natural language semantics has been pioneered by de Groote (2001b) and Barker (2002). In the theory of programming languages, a continuation is a representation of the control state, i.e., the future of the computation to be performed. By adding the control state as an explicit parameter in the interpretation, it becomes possible for a program to manipulate its continuation, and thus to express control constructs that would otherwise be unavailable. Technically, continuation semantics makes use of a designated response type for the ultimate result of a computation; a continuation for an expression of type \(A\) is a function that takes an \(A\) value to the response type. In the application to natural language semantics, the response type is generally identified with the type of truth values \(t\), i.e., the type assigned to complete sentences.

Barker 2004 shows how to obtain the type-shifting scope ambiguity for ‘Everyone loves someone’ in terms of a continuation-passing-style translation. The key idea is that in mapping the source calculus to the semantic target calculus, all types are parameterized with an extra continuation argument. We use a new translation function \((-)^*\) for this, which composes the mapping from syntactic to semantic types with the continuization. A source language type \(A\) is now associated with a so-called computation in the target language, i.e., a function acting on its own continuation: \(A^* = (A' \rightarrow t) \rightarrow t\).

At the level of proofs, given the above interpretation of types, the task is to find an \(LP\) proof for the sequent below, the image of the \(\backslash\) and / elimination rules for \(A, A'B \vdash B\) and \(B/A, A \vdash B\). Whereas in the source calculus there is only one way of putting together an \(A/B\) function with its \(A\) argument, in the target calculus there is a choice as to the evaluation order: do we want to first evaluate the translation image of the argument, then that of the function, or the other way around. We write \(-^\lor\) for the first option (call-by-value) and \(-^\land\) for the second (call-by-name). In the target language, \(m\) and \(n\) are variables of type \(A' \rightarrow B'\) and \(A'\) respectively; \(k\) is the resulting \(B' \rightarrow t\) continuation.

- \(LP\) translation of Application:
  \((A' \rightarrow t), ((A' \rightarrow B') \rightarrow t) \vdash (B' \rightarrow t) \rightarrow t\)
The continuation-passing-style interpretation, like the type-shifting approach, makes it possible to assign syntactic type \( np \) both to proper names and to quantificational noun phrases: in the target calculus, the translation of \( np \) has the appropriate semantic type \( (e \rightarrow t) \rightarrow t \). But the lifting strategy is now generalized to all source types: a transitive verb \((np)s/np)\) is mapped to \((e \rightarrow e \rightarrow t) \rightarrow t\), etc. For the translation of lexical constants of type \( A \), the default recipe is \( \lambda k. (c) \), \( c \) a non-logical constant of type \( A' \). The default translation simply passes the semantic value for these lexical items to the continuation parameter \( k \). But quantificational noun phrases effectively exploit the continuation parameter: they take scope over their continuation, leading to lexical recipes \( \lambda k. (\forall x. (k x)) \), \( \lambda k. (\exists x. (k x)) \) for ‘everyone’ and ‘someone’. The choice between these evaluation strategies, in combination with these lexical recipes, then results in different interpretations for a single derivation in the source calculus \( M = (\text{everyone} \Rightarrow (\text{loves} \Rightarrow \text{someone})) \), with \( \forall \) producing the surface scope construal, and \( \exists \) the inverted scope reading.

\[
\begin{align*}
M^\forall &= \lambda k. (\forall x. (\exists y. (k ((\text{loves} y) x))) \\
M^n &= \lambda k. (\exists y. (\forall x. (k ((\text{loves} y) x)))
\end{align*}
\]

The above example only uses the slash Elimination of \( \text{NL} \) in the syntactic source calculus. An important motivation for the introduction of continuations is that they make it possible to give a constructive interpretation to classical (as opposed to intuitionistic) logic; see Sørensen and Urzyczyn 2006 for discussion. It will be no surprise, then, that also the multiple-conclusion symmetric categorial grammar \( \text{LG} \) has a natural interpretation in the continuation-passing-style (Bernardi and Moortgat 2010, Moortgat 2009). In the example we gave above, source types \( A \) are provided with a single continuation parameter. In the CPS translation for \( \text{LG} \) types, the continuization of syntactic source types is performed recursively. Let us write \( V(A) \) for target language values of type \( A \), \( K(A) \) for continuations, i.e., functions \( V(A) \rightarrow R \), and \( C(A) \) for computations, i.e., functions \( K(A) \rightarrow R \), where \( R \) is the response type. For an \( \text{LG} \) syntactic source type \( A \), the call-by-value CPS translation yields an \( \text{LP} \) value \( V(A) \) as follows. \( V(p) = p \) for atomic types,

- implications: \( V(A \mid B) = V(B/A) = K(B) \rightarrow K(A) \)
- complications (dual to implications): \( V(A \otimes B) = V(B \otimes A) = K(A \mid B) \)

At the level of proofs (and the terms in Curry-Howard correspondence with them), the CPS translation turns the multiple-conclusion source derivations into single-conclusion \( \text{LP} \) derivations. The translation respects the invariant below. An active output formula \( A \) (marked off by the vertical bar in the box below) maps to a computation \( C(A) \), an active input formula to a continuation \( K(A) \). A cut then is interpreted as the application of \( C(A) \) to \( K(A) \).
Discussion. Continuation-based approaches are now available for a number of recalcitrant phenomena that would seem to defy a compositional treatment. Examples, at the sentential level, include the treatment of in situ scope construal and wh questions of Shan and Barker 2006, where crossover and superiority violations are explained in terms of a preference of the human processor for a left-to-right evaluation strategy; donkey anaphora (Barker and Shan 2008); quantifier scope ambiguities for the call-by-value and call-by-name interpretation of LG are investigated in Bernardi and Moortgat 2010. At the discourse level, de Groote (2006) gives a type-theoretic analysis of dynamic phenomena, modeling propositions as functions over a sentence's left and right context (continuation).

4. Proof nets and processing

In the previous sections, we have seen how the different categorial calculi can be presented as proof systems with a sequent-based decision procedure. Naive proof search in these systems, however, is inefficient from a computational point of view: in general, there will be multiple ways to construct alternative proofs that differ only in the order of the application of the inference rules, but produce the same interpretation (the LP term associated with a derivation under the interpretation homomorphism). This issue of ‘spurious’ ambiguity can be tackled by the introduction of normal form derivations (as in Hepple 1990, Hendriks 1993), compare focused proof search regimes in linear logic), combined with the use of chart-based parsing methods, as in Hepple 1999, Capelletti 2007. An alternative for categorical ‘parsing-as-deduction’ is to leave the sequent calculus for what it is, and to switch to a proof net approach. Proof nets, originally developed in the context of linear logic, use a representation of derivations that is inherently free of ‘spurious ambiguity’, i.e., the issue of irrelevant rule orderings simply doesn't arise. For a general treatment of proof nets, we refer to the entry on linear Logic. Below we comment on aspects that are specifically addressing issues in categorial grammar.

Proof nets for the Lambek calculi. Proof nets for the associative calculus L and the commutative variant LP have been studied initially by Roorda (1991, 1992). To capture the ‘intuitionistic’ nature of these systems, one works with formulas with polarities: input (‘given’) polarity for antecedent occurrences of a formula, versus output (‘to prove’) polarity for succedent occurrences. Correctness criteria identify proof nets among a wider class of graphs: to the criteria of acyclicity and connectedness, applicable for linear logic in general, one adds planarity to capture word order sensitivity: in the proof nets for L, axioms links cannot cross. With respect to the syntax-semantics interface, de Groote and Retoré (1996) show that the lambda terms in Curry-Howard correspondence with LP derivations in the semantic target calculus can be read off from a proof net by specifying a set of ‘travel instructions’ for traversing a net; these instructions then correspond step-by-step with the construction of the associated lambda term.

Incremental processing. Proof nets, considered statically as graphs satisfying certain correctness criteria, remove spurious choices relating to the order of rule applications in sequent calculi: in this respect, they represent a purely ‘declarative’ view on categorial deductions. Johnson (1998) and Morrill (2000) have pointed out that an alternative, ‘procedural’ view on the actual process of constructing a net makes perfect sense as well, and offers an attractive perspective on performance phenomena. Under this interpretation, a net is built in a left-to-right incremental fashion by establishing possible linkings between the input/output literals of the partial proof nets associated with lexical items as they occur in real time. This suggests a simple complexity measure on an incremental traversal, given by the number of unresolved dependencies between literals. This complexity measure correlates nicely with a number of well-attested processing issues, such as the difficulty of center embedding, garden path effects, attachment preferences, and preferred scope construals in ambiguous constructions.

First-order quantifiers. The planarity condition singles out the non-commutative proof nets for L among the LP nets. To deal with the more structured categorial calculi discussed here, the correctness criteria have to be refined. One strategy of doing this is via a translation into MILL1 (first-order multiplicative intuitionistic linear logic) where one has proof nets with extra links for existential and universal quantification over first order variables. One can then use these variables in a way very similar to the use of position arguments in Definite Clause Grammars as used in logic programming. Moot and Piazza (2001) work out such translations for L and NL. For the concatenation operations of L, one replaces the proposition letters (atomic formulas) by two-place terms, marking beginning and end of a continuous string. For non-associative NL, one adds an extra variable to keep track of the nesting depth of subformulas. For wrapping operations in the simple discontinuous calculus DL (allowing a single split point), Morrill and Fadda (2008) use four-place predicates. In general, we find a correlation here between the syntactic expressivity of the calculi and the number of variables needed to encode their structural resource management.
Nets and rewriting. The multimodal and symmetric calculi of §3.1 and §3.3 pose a challenge to the proof net methods as originally developed for linear logic. In these systems we typically find one-way structural rules, such as the extraction postulates for overt displacement, or the Grishin distributivity laws in the case of LG: these one-way rules naturally suggest a notion of graph rewriting. A completely general proof net framework for the extended Lambek calculi, with a correctness criterion based on rewriting, has been developed in Moot and Puite 2002 and Moot 2007.

The basic building block for the Moot-Puite nets is a generalized notion of a link, accommodating connectives of any arity. A link is determined by its type (tensor or cotensor), its premises (a sequence \( P_1 \ldots P_n \), \( 0 \leq n \)), its conclusions (a sequence \( C_1 \ldots C_m \), \( 0 \leq m \)), and its main formula (which can be empty, in the case of a neutral link, or one of the \( P_i \) or \( C_j \)). A proof structure is a set of links over a finite set of formulas such that every formula is at most once the premise of a link and at most once the conclusion. Formulas which are not the conclusion of any link are the hypotheses of the proof structures, whereas the formulas which are not the premise of any link are the conclusions. An axiom formula is a formula which is not the main formula of any link. Abstract proof structures are obtained by erasing all formulas on the internal nodes. A proof net is a proof structure for which the abstract proof structure converts to a tensor tree — a rooted tree in the case of the intuitionistic systems, possibly an unrooted tree in the case of symmetric LG. Proofs nets, so defined, can then be shown to be the graphs that correspond to valid derivations.

The rewriting steps transforming a candidate abstract proof structure into a proof net are of two kinds. The logical contractions correspond to identities \( A \vdash A \), for complex formulas \( A \); they reduce a configuration of a matching tensor and cotensor link to a point. The structural conversions perform an internal rewiring of a proof structure with hypotheses \( H_1 \ldots H_n \) and conclusions \( C_1 \ldots C_m \) to a structure with some permutation of the \( H_i \) as hypotheses and some permutation of the \( C_i \) as conclusions. Copying and deletion, in other words, are ruled out. As an example, the rewriting corresponding to one of the Grishin interaction principles discussed in §3.3, allowing us to infer \( C \otimes A \vdash D \mid B \) from \( A \otimes B \vdash C \oplus D \). Hypotheses \( A, B \) are on the left, conclusions \( C, D \) on the right. The net representation brings out in a particularly clear way that these principles are structure-preserving: they leave the order of hypotheses and conclusions untouched.

5. Recognizing capacity, complexity

Reflecting on the above, one can say that the extensions of the syntactic calculus reviewed here are motivated by the desire to find a proper balance between expressivity and computational tractability: on the side of expressivity, an adequate typological framework should be able to recognize patterns beyond the strictly context-free; ideally, one would like to keep such expressivity compatible with a polynomial derivability problem. Among contemporary ‘lexicalized’ grammar formalisms, there is a remarkable convergence on systems that meet these requirements: the so-called ‘mildly context-sensitive’ systems (Joshi, Vijay-Shanker, and Weir 1991). Where can we situate the typological systems discussed here with respect to recognizing capacity and complexity?

The minimal system in the typological hierarchy NL has a polynomial recognition problem (see de Groote 1999, and Capelletti 2007 for actual parsing
algorithms), but it is strictly context-free (Kandulski 1988). Extensions with global structural rules are unsatisfactory, both on the expressivity and on the complexity front. As for \textbf{L}, Pentus (1993b, 2006) shows that it remains strictly context-free, whereas the addition of global associativity makes the derivability problem NP complete. NP completeness already holds for the product-free fragment of \textbf{L} (Savateev 2009). Also for \textbf{LP}, i.e., multiplicative intuitionistic linear logic, we have NP completeness (Kanovich 1994). With regard to recognizing capacity, van Benthem (1995) shows that \textbf{LP} recognizes all permutation closures of context-free languages: a class which is too wide from the syntactic point of view. As the logic of meaning assembly, \textbf{LP} is a core component of the typelogical inventory. But as we saw in the discussion of the syntax-semantics interface, we can restrict attention to the sublanguage of \textbf{LP} that forms the image of derivations in syntactic calculi making interesting claims about word order and phrase structure.

The situation of the multimodal and symmetric extensions is more intricate. Expressivity here is directly related to the kind of restrictions one imposes on structural resource management. At one end of the spectrum, multimodality without structural rules does not lead us beyond context-free recognition: Jäger (2003) shows that the pure residuation logic for \textit{n}-ary families of type-forming operations stays strictly context-free. If one requires structural rules to be resource-sensitive (no copying or deletion) and, for the unary modalities, non-expanding, one obtains the full expressivity of context-sensitive grammars, and the PSPACE complexity that goes with it (Moot 2002). (PSPACE consists of those problems solvable using some polynomial amount of memory space. See the entry on computability and complexity. If one imposes no restrictions on structural rules (specifically, if one allows copying and deletion operations), unsurprisingly, one obtains the expressivity of unrestricted rewriting systems (Carpenter 1999). A controlled use of copying is used in the analysis of anaphora resolution in Jäger 2005.

The symmetric calculus \textbf{LG} without interaction principles is context-free, as shown in Bastenhof 2010. For the system with the interaction principles of §3.3, Melissen (2009) shows that all languages which are the intersection of a context-free language and the permutation closure of a context-free language are recognizable in \textbf{LG}. In this class, we find generalized forms of MIX (the language consisting of an equal number of symbols \(a, b, c\), in any order), with equal multiplicity of \(k\) alphabet symbols in any order, and counting dependencies \(a^k_1 \ldots a^k_k\) for any number \(k\) of alphabet symbols. Patterns of this type are recognized by Range Concatenation Grammars and Global Index Grammars; a comparison with these formalisms then might be useful to fix the upper bound of the recognizing capacity of \textbf{LG}, which is as yet unknown.

With respect to computational complexity, Moot (2008) establishes a correspondence between Lexicalized Tree Adjoining Grammars on the one hand, and categorial grammars with the multimodal extraction postulates of §3.1 and a restricted set of \textbf{LG} grammars on the other. For these grammars he obtains a polynomial parsability result via a translation into Hyperedge Replacement Grammars. In the case of \textbf{LG}, the restriction requires the complications \(\emptyset, \otimes\) to occur in matching pairs in lexical type assignments. The lexicon of the generalized MIX construction of Melissen 2009, and the type assignment used for quantifier phrases in the analysis of scope construal in Bernardi and Moortgat 2010, do not respect this restriction. For the general case of \textbf{LG} with the interaction principles of §3.3, Bransen (2010) establishes NP-completeness. The position of the discontinuous calculi of §3.2 in this hierarchy has to be determined: they recognize more than the context-free languages, but it is not clear whether they stay within the mildly context-sensitive family.

6. Related approaches

Typelogical grammars as discussed here share a number of characteristics with a number of related formal grammar frameworks where types also play a key role. We briefly discuss some of these relatives, and provide pointers to the literature for readers who want to explore the connections further.

\textbf{Combinatory Categorial Grammar} (CCG). The CCG framework (see Steedman 2000 for a comprehensive presentation) takes its name from the Combinatory Logic of Curry and Feys. Whereas the design of typelogical grammars follows a logical agenda (sequent calculus, soundness, completeness, etc), grammars in the CCG tradition are presented as finite sets of rules for type change and type combination. This rule inventory will typically include function application schemata, type lifting, and functional composition, both of the kind valid within associative \textbf{L} (composing implicational types with the same directionality), and of the mixed kind (composing functors with different directionality).
The development of CCG and Lambek style typological grammars in the 1980s initially followed separate courses. More recently, there are signs of convergence. An important factor has been the introduction of the typological technique of multimodal control to fine-tune the applicability of the CCG combinatory schemata (Baldrige 2002, Kruijff and Baldrige 2003). Multimodal typological grammar, from this perspective, plays the role of the underlying general logic which one uses to establish the validity of the CCG combinatory schemata, whereas the particular choice of primitive combinators is motivated by computational efficiency considerations (Hoyt and Baldrige 2008). In terms of generative capacity, CCG is a member of the mildly context sensitive family of grammar formalisms. Polynomial parsability is obtained by imposing a bound on the functional composition schemata.

**Pregroup grammars.** Pregroups are an algebraic version of compact bilinear logic (Lambek 1993) obtained by collapsing the tensor and cotensor operations. Pregroup grammars were introduced in Lambek 1999 as a simplification of the original Syntactic Calculus L. They have since been used to build computational fragments for a great variety of languages by Lambek and co-workers. A pregroup is a partially ordered monoid in which each element a has a left and a right adjoint, a′, satisfying a′a → 1 → aa′ and aa′r → 1 → a′a, respectively. Type assignment takes the form of associating a word with one or more elements from the free pregroup generated by a partially ordered set of basic types. For the connection with categorial type formulas, one can use the translations ab = a′b and b′a = a′a. Parsing, in the pregroup setting, is extremely straightforward. Lambek (1999) proves that one only has to perform the contractions replacing a′a and a′a by the multiplicative unit 1. This is essentially a check for well-bracketing — an operation that can be entrusted to a pushdown automaton. The expansions 1 → a′a and 1 → a′a are needed to prove equations like (ab)′ = b′a′. We have used the latter to obtain the pregroup version of the higher-order relative pronoun type (n\(\land\)n)/(s\(\land\)np) in the example below.

<table>
<thead>
<tr>
<th>book</th>
<th>that</th>
<th>Carroll wrote</th>
</tr>
</thead>
<tbody>
<tr>
<td>type assignment in L :</td>
<td>n (n(\land)n)/(s(\land)np)</td>
<td>np (np(\land)s)/np</td>
</tr>
<tr>
<td>pregroup type assignment :</td>
<td>n n′ r np(\land)l s′ l</td>
<td>np′ r s np′ l → n</td>
</tr>
</tbody>
</table>

Compact bilinear logic is not a conservative extension of the original Syntactic Calculus. Every sequent derivable in L has a translation derivable in the corresponding pregroup, but the converse is not true: the pregroup image of the types (a \(\otimes\) b)c and a \(\otimes\) (b/c), for example, is a b c′, but these two types are not interderivable in L.

With respect to generative capacity, Buszkowski (2001) shows that the pregroup grammars are equivalent to context-free grammars. They share, in other words, the expressive limitations of the original categorial grammars. To overcome these limitations different strategies have been pursued, including lexical rules (metarules), derivational constraints, controlled forms of commutativity, and products of pregroups. The *Studia Logica* special issue (Buszkowski and Preller 2007) and the monograph Lambek 2008 give a good picture of current research.

**Abstract Categorial Grammar.** The ACG framework (de Groote 2001a) is a meta-theory of compositional grammar architectures. ACGs are built on higher-order linear signatures \(\Sigma = (A,C,\tau)\), where A and C are finite sets of atomic types and constants respectively, and \(\tau\) a function assigning each constant a linear implicative type over A. Given a source signature \(\Sigma\) and a target signature \(\Sigma'\), an interpretation is a mapping form \(\Sigma\) to \(\Sigma'\) given by a pair of functions: \(\eta\) mapping the type atoms of \(\Sigma\) to linear implicative types of \(\Sigma'\) and \(\theta\) mapping the constants of \(\Sigma\) to well-typed linear lambda terms of \(\Sigma'\) in a way that is compatible with the mapping on types. Using the terminology of compiler theory, one refers to the source and target signatures as the abstract vocabulary and the concrete vocabulary, respectively, and to the interpretive mapping as the *lexicon*. An ACG is then obtained by specifying an atomic type of \(\Sigma\) as the distinguished type of the grammar.

In the ACG setting, one can model the syntax-semantics interface in terms of the abstract versus object vocabulary distinction. But one can also obtain surface form as the result of an interpretive mapping, using the canonical \(\lambda\) term encodings of strings and trees and operations on them. ACG has given rise to an interesting complexity hierarchy for rewriting grammar formalisms encoded as ACGs: context-free grammars, tree-adjoining grammars, etc.; see for example
Groote and Pogodalla 2004. Expressive power of these formalisms is measured in terms of the maximal order of the constants in the abstract vocabulary and of the object types interpreting the atomic abstract types. The study of ACG encodings of typelogical systems proper has started with Retoré and Salvati 2010; these authors present an ACG construction for (product-free) NL.

It will be clear from this description that the ACG architecture is closely related to the compositional interpretation for categorial type logics as discussed in this article. A key difference is the nature of the ‘abstract syntax’, i.e. the source calculus from which interpretations are homomorphically derived. In the case of the standard Lambek systems and the extended systems discussed in §3 above, the abstract syntax is a directional type logic; in the case of ACG, one finds LP and the linear lambda calculus both at the source and at the target end. The debate as to whether structural properties of language have to be accounted for at the level of the abstract syntax has a long history, starting with Curry 1961; see Muskens 2007 for discussion. The typological view accounts for word order universals at the level of its logical constants, i.e., the type-forming operations, and the laws that govern them. The ACG view is more liberal in this respect: the derivation of surface form can be specified on a word-by-word basis. Whether this more relaxed connection between abstract syntax and surface realisation is desirable, is a matter of debate. An attractive feature of ACG that has not been investigated systematically within the typelogical setting is the connection between expressivity and order restrictions on the source constants and on the interpretive mapping.

Bibliography

Note: In addition to the regular text references, the bibliography contains some items that can place the entry in a broader context.

- For general logical and mathematical background, see Galatos et al. 2007, Restall 2000, Sørensen and Urzyczyn 2006.


Carpenter, B. (1999). The Turing-completeness of multimodal categorial grammars. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema (Eds.), *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Amsterdam: Amsterdam University Press.


http://plato.stanford.edu/entries/typelogical-grammar/


and Information, 19(2). Special issue on New Directions in Type Theoretic Grammars.

Academic Tools

🔗 How to cite this entry.
🔗 Preview the PDF version of this entry at the Friends of the SEP Society.
🔗 Look up this entry topic at the Indiana Philosophy Ontology Project (InPhO).
🔗 Enhanced bibliography for this entry at PhilPapers, with links to its database.

Other Internet Resources

- Moot's typelogical theorem prover
- Symmetric Categorial Grammar
- Abstract Categorial Grammar
- Combinatory Categorial Grammar

Related Entries

compositionality | logic: linear | logic: modal | logic: relevance | logic: substructural | type theory