1

On rigid NL Lambek grammars inference from generalized functor-argument data

Denis Béchet and Annie Foret

Abstract

This paper is concerned with the inference of categorial grammars, a context-free grammar formalism in the field of computational linguistics. A recent result has shown that whereas they are not learnable from strings in the model of Gold, rigid and k-valued non-associative Lambek grammars are still learnable from generalized functor-argument structured sentences. We focus here on the algorithmic part of this result and provide an algorithm that can be seen as an extension of Buszkowski, Penn and Kanazawa's contributions for classical categorial grammars.

Keywords grammatical inference, categorial grammars, non-associative Lambek calculus, learning from positive examples, model of Gold

1.1 Introduction and background

This paper is concerned with the inference of categorial grammars, a context-free grammar formalism in the field of computational linguistics. We consider learning from positive data in the model of Gold. When the data has no structure, a recent result has shown that rigid and k-valued non-associative Lambek (NL) grammars admit no learning algorithm (in the model of Gold). By contrast, theses classes become theoretically learnable from generalized functor-argument structured sentences; the paper (Bechet and Foret (2003)) establishes the

FG-MoL 2005.
Organizing Committee:, Gerhard Jaeger, Paola Monachesi, Gerald Penn, James Rogers, Shuly Wintner.

existence of such learning algorithms, without explicitly defining one. We focus here on the algorithmic part for *NL* rigid grammars; we explicit an algorithm (for the rigid case) that can be seen as an extension to *NL* of Buszkowski, Penn and Kanazawa's contributions Buszkowski and Penn (1990), Kanazawa (1998a) for classical (AB) categorial grammars. The algorithm introduced here ouputs a grammar with variables and constraints, an alternative representation for *NL* rigid grammars, that is defined later in this article; this kind of representation can be seen as the necessary information used by parses to check sentence recognizability Aarts and Trautwein (1995).

Learning. The model of Gold (1967) used here consists in defining an algorithm on a finite set of structured sentences that converges to obtain a grammar in the class that generates the examples.

In a grammar system $\langle \mathbf{G}, \mathbf{S}, \mathbf{L} \rangle$ (that is $\mathbf{G}$ is a "hypothesis space", $\mathbf{S}$ is a "sample space", $\mathbf{L}$ is a function from $\mathbf{G}$ to subsets of $\mathbf{S}$) a function ϕ is said to learn $\mathbf{G}$ in Gold's model iff for any $G \in \mathbf{G}$ and for any enumeration $\langle e_i \rangle_{i \in \mathbf{N}}$ of $\mathbf{L}(G)$ there exists $n_0 \in \mathbf{N}$ and a grammar $G' \in \mathbf{G}$ such that $\mathbf{L}(G') = \mathbf{L}(G)$ and $\forall n \geq n_0, \phi(\langle e_0, \ldots, e_n \rangle) = G'$.

Categorial Grammars. The reader not familiar with Lambek Calculus and its non-associative version will find nice presentation in (Lambek (1961), Buszkowski (1997), Moortgat (1997), de Groote and Lamarche (2002)). We use in the paper non-associative Lambek calculus (written *NL*) without empty sequence and without product.

The *types Tp* are generated from a set of *primitive types Pr* by two binary connectives "/" (over) and "\" (under): $Tp ::= Pr \mid Tp \backslash Tp \mid Tp / Tp$.

A *categorial grammar* is a structure $G = (\Sigma, I, S)$ where: Σ is a finite alphabet (the words in the sentences); $I : \Sigma \mapsto \mathcal{P}^f(T)$ is a function that maps a set of types to each element of Σ (the possible categories of each word); $S \in Pr$ is the *main type* associated to correct sentences. A *k-valued categorial grammar* is a categorial grammar where, for every word $a \in \Sigma$, $I(a)$ has at most k elements. A *rigid categorial grammar* is a 1-valued categorial grammar.

A sentence belongs to the *language of* G, provided its words can be assigned types that derive S according to the rules of the type calculus.

The case of AB categorial grammars is defined by two rules :

$$/_e : A \,/\, B, B \Rightarrow A \quad \text{and} \quad \backslash_e : B, B \setminus A \Rightarrow A \qquad (AB \text{ rules})$$

In the case of *NL*, the derivation relation $\vdash_{NL}$ is defined by (left part belongs to $\mathcal{T}_{Tp}$ the set of binary trees over Tp) :

$$\frac{}{A \vdash A}\,\mathbf{Ax} \qquad \frac{(\Gamma, B) \vdash A}{\Gamma \vdash A/B}\,/R \qquad \frac{(A, \Gamma) \vdash B}{\Gamma \vdash A\backslash B}\,\backslash R$$

$$\frac{\Gamma \vdash A \quad \Delta[A] \vdash B}{\Delta[\Gamma] \vdash B}\,\mathbf{Cut} \qquad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(B/A, \Gamma)] \vdash C}\,/L \qquad \frac{\Gamma \vdash A \quad \Delta[B] \vdash C}{\Delta[(\Gamma, A\backslash B)] \vdash C}\,\backslash L$$

Learning categorial grammars. Lexicalized grammars of natural languages such as categorial grammars are adapted to learning perspectives: since the rules are known, the task amounts to determine the type assignment; this usually involves a unification phase.

In fact, the learnable or unlearnable problem for a class of grammars depends both of the information that the input structures carry and the model that defines the language associated to a given grammar. The input information can be just a string, the list of words of the input sentence. It can be a tree that describes the sub-components with or without the indication of the head of each sub-component. More complex input information give natural deduction structure or semantics information.

For k-valued categorial grammars: *AB* grammars are learnable from strings (Kanazawa (1998b)), associative Lambek grammars are learnable from natural deduction structures (elimination and introduction rules) (Bonato and Retoré (2001)) but not from strings (Foret and Le Nir (2002)), NL grammars are not learnable from well-bracketed strings but are from generalized functor argument structures (Bechet and Foret (2003)) .

Organization of the paper. Section 2 explains the notion of functor-argument and its generalized version in the case of *NL* (as recently introduced in (Bechet and Foret (2003))). Section 3 defines the algorithm (named *RGC*) on these structures. Section 4 defines grammars with variables and constraints underlying the *RGC* algorithm. Section 5 gives the main properties of the *RGC* algorithm. Section 6 concludes.

1.2 *FA* structures in *NL*

1.2.1 *FA* structures

Let Σ be an alphabet, a *FA structure* over Σ is a binary tree where each leaf is labelled by an element of Σ and each internal node is labelled by the name of the binary rule.

When needed we shall distinguish two kinds of *FA* structures, over the grammar alphabet (the default one) and over the types.

1.2.2 *GAB* Deduction

A *generalized AB deduction*, or GAB deduction, over *Tp* is a binary tree using the following conditional rules ($C \vdash_{NL} B$ must be valid in *NL*):

$$\frac{A/B \quad C}{A}\,/_{e^+} \qquad \frac{C \quad B\backslash A}{A}\,\backslash_{e^+} \qquad \left|\begin{array}{l} \textit{condition (both rules)} \\ C \vdash_{NL} B \text{ valid in } NL \end{array}\right.$$

In fact, a deduction must be justified, for each node, by a proof of the corresponding sequent in *NL*. Thus, a rule has three premises: the two sub-deductions and a *NL* derivation (later called *a constraint*).

Notation. For $\Gamma \in \mathcal{T}_{Tp}$ and $A \in Tp$, we write $\Gamma \vdash_{GAB} A$ when there is a *GAB* deduction $\mathcal{P}$ of A corresponding to the binary tree Γ.

There is a strong correspondence between *GAB* deductions and *NL* derivations. In particular Theorem 1 shows that the respective string languages and binary tree languages are the same.

Theorem 1 (Bechet and Foret (2003)) *If A is atomic, $\Gamma \vdash_{GAB} A$ iff $\Gamma \vdash_{NL} A$*

1.2.3 From *GAB* deductions to *FA* structures in *NL*

A *GAB* deduction $\mathcal{P}$ can be seen as an annotated *FA* structure where the leaves are types and the nodes need a logical justification.

We write $FA_{Tp}(\mathcal{P})$ for the *FA* structure over types that corresponds to $\mathcal{P}$ (internal types and *NL* derivations are forgotten). We then define *FA* structures over Σ instead of over *Tp* (these two notions are close).
Definition *$G = (\Sigma, I, S)$ generates a FA structure F over Σ* (in the *GAB* derivation model) iff there exists $c_1, \ldots, c_n \in \Sigma$, $A_1, \ldots, A_n \in Tp$ and a *GAB* derivation $\mathcal{P}$ such that:

$$\begin{cases} G : c_i \mapsto A_i \ (1 \leq i \leq n) \\ FA_{Tp}(\mathcal{P}) = F[c_1 \to A_1, \ldots, c_n \to A_n] \end{cases}$$

where $F[c_1 \to A_1, \ldots, c_n \to A_n]$ is obtained from F by replacing respectively the left to right occurrences of $c_1, \ldots, c_n$ by $A_1, \ldots, A_n$.

The *language of FA structures* corresponding to G, written $\mathcal{FL}_{GAB}(G)$, is the set of *FA* structures over Σ generated by G.

Example 1 *Let G_1: $\{John \mapsto N \ ; \ Mary \mapsto N \ ; \ likes \mapsto N\backslash(S/N)\}$ we get: $/_{e^+}(\backslash_{e^+}(John, likes), Mary) \in \mathcal{FL}_{GAB}(G_1)$*

1.3 Algorithm on functor-argument data

1.3.1 Background -RG algorithm

We first recall Buszkowski's Algorithm called RG as in Kanazawa (1998b), it is defined for *AB* grammars, based on $/_e$ and $\backslash_e$.

The RG algorithm takes a set D of functor-argument structures as positive examples and returns a rigid grammar $RG(D)$ compatible with the input if there is one (compatible means that D is in the set of functor-argument structures generated by the grammar).

Sketch of RG-algorithm, computing $RG(D)$:

1. assign S to the root of each structure
2. assign distinct variables to argument nodes
3. compute the other types on functor nodes according to $/_e$ and $\backslash_e$
4. collect the types assigned to each symbol, this provides $GF(D)$
5. unify (classical unification) the types assigned to the same symbol in $GF(D)$, and compute the most general unifier σ_{mgu} of this family of types.
6. The algorithm fails if unification fails, otherwise the result is the application of σ_{mgu} to the types of $GF(D)$: $RG(D) = \sigma_{mgu}(GF(D))$.

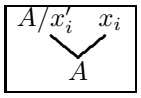
1.3.2 An algorithm for NL : a proposal

In the context of NL-grammars, we show how rules $/_{e+}$ and $\backslash_{e+}$ will play the role of classical forward and backward elimination $/_e$ and $\backslash_e$.

We now give an algorithm that given a set of positive examples D computes $RGC(D)$ composed of a general form of grammar together with derivation constraints on its type variables. This formalism is developped in next section. The main differences between RG and RGC appear in steps (3) and (6).

Algorithm for $RGC(D)$ **- rigid grammar with constraints**

1. assign S to root ;
2. assign distinct variables x_i to argument nodes ;
3. compute the other types on functor nodes and the derivation constraints according to $/_{e+}$ and $\backslash_{e+}$ rules as follows : for each functor node in a structure corresponding to an argument x_i and a conclusion A_i assign one of the types (according to the rule) A_i/x'_i or $x'_i\backslash A_i$, where the x'_i variables are all distinct and add the constraint $x_i \vdash x'_i$;



4. collect the types assigned to each symbol, this provides $GF^+(D)$; and collect the derivation constraints, this defines $GC^+(D)$; let $GFC(D) = \langle GF^+(D), GC^+(D)\rangle$
5. unify (classical unification) the types assigned to the same symbol in $GF^+(D)$, and compute the most general unifier σ_{mgu} of this family of types.

6. The algorithm fails if unification fails, otherwise the result is the application of σ_{mgu} to the types of $GF^+(D)$ and to the set of constraints, this defines $RGC(D) = \langle RG^+(D), RC^+(D)\rangle$ consisting in the rigid grammar $RG^+(D) = \sigma_{mgu}(GF^+(D))$ and the set of derivation constraints $RC^+(D) = \sigma_{mgu}(GC^+(D))$; the result is later written $RGC(D) = \sigma_{mgu}(GFC(D))$.

Example 2 *We consider the following functor-argument structures :*
$\backslash_{e^+}(/_{e^+}(a, man), swims)$
$\backslash_{e^+}(/_{e^+}(a, fish), \backslash_{e^+}(swims, fast))$
The algorithm computes the following grammar with constraints :

	general form	*unification*	*rigid grammar, constraints*
a	$X_1 \,/\, X'_2, X_3 \,/\, X'_4$ $X_2 \vdash X'_2, X_4 \vdash X'_4$	$X_1 = X_3, X'_2 = X'_4$	$X_1 \,/\, X'_2$ $X_2 \vdash X'_2, X_4 \vdash X'_2$
$fast$	$X'_5 \setminus (X'_3 \setminus S)$ $X_3 \vdash X'_3, X_5 \vdash X'_5$		$X'5 \setminus (X'_3 \setminus S)$ $X_1 \vdash X'_3, (X'_1 \setminus S) \vdash X'5$
$fish$	X_4		X_4
man	X_2		X_2
$swims$	$X'_1 \setminus S, X_5$ $X_1 \vdash X'_1$	$X_5 = X'_1 \setminus S$	$X'_1 \setminus S$ $X_1 \vdash X'_1$

1.4 Grammars with variables and constraints

Let Var denote a set of variables, we write Tp_{Var} the set of types that are constructed from the primitive types $Pr \cup Var$.

Definition. A *grammar with variables and constraints* on Var is a structure $\langle G, \mathcal{C}\rangle$ where G is a categorial grammar whose types are in Tp_{Var} and $\mathcal{C}$ is a set of sequents $t_i \vdash t'_i$ with types in Tp_{Var}.

We then define a structure language $\mathbf{FL}_{GAB}(\langle G, \mathcal{C}\rangle)$ based on derivations defined similarly to *GAB* , by replacing the conditions $C \vdash B$ valid in *NL* with conditions of the form $C \vdash B \in \mathcal{C}$ (constraints):

Definition. $\mathbf{FL}_{GAB}(\langle G, \mathcal{C}\rangle)$ is the set of functor-argument structures obtained from deductions using the following conditional rules :

$$\frac{A/B \quad C}{A}\,/_{e^+C} \qquad \frac{C \quad B\backslash A}{A}\,\backslash_{e^+C} \qquad \Big|\ \begin{array}{c}\textit{condition (both rules)}\\ C \vdash B \in \mathcal{C}\end{array}$$

Definition. For any NL-grammar G, $Constraints(G)$ is the set of NL-valid sequents $t_i \vdash t'_i$ where t_i is any head subtype[1] of a type assigned by G, and t'_i is any argument subtype[2] of any type assigned by G.

Property: the constraints of any rule $/_{e^+}$ or $\backslash_{e^+}$ in a GAB deduction for G belong to $Constraints(G)$ (proof by induction on a deduction).

[1] A is head subtype of A, a head subtype of A is head subtype of A/B and $B\backslash A$

[2] B is an argument subtype of A/B and of $B\backslash A$, any argument subtype of A is an argument subtype of A/B and of $B\backslash A$

A partial order[3] **on grammars with variables and constraints.**
Let us write $\langle G, C\rangle \subseteq \langle G', C'\rangle$ IFF both $G \subseteq G'$[4] and $C \subseteq C'$.
For a substitution σ, we write $\sigma\langle G, C\rangle = \langle \sigma(G), \sigma(C)\rangle$ and define:

$\langle G, C\rangle \sqsubseteq \langle G', C'\rangle$ IFF $\exists \sigma : \sigma\langle G, C\rangle \subseteq \langle G', C'\rangle$

$G \sqsubseteq G'$ IFF $\exists \sigma : \sigma(G) \subseteq G'$

Next properties follow easily.
Property:

$$\langle G, C\rangle \sqsubseteq \langle G', C'\rangle \text{ implies } \mathbf{FL}_{GAB}(\langle G, C\rangle) \subseteq \mathbf{FL}_{GAB}(\langle G', C'\rangle)$$

Property: $\mathbf{FL}_{GAB}(\langle G, Constraints(G)\rangle) = \mathcal{FL}_{GAB}(G)$, for G on *Tp*.

1.5 Properties of algorithm *RGC*.

Next property is analogue to properties by Buszkowski and Penn for RG; the first part concerns the general forms (defined in step (4) of *RGC*), the second concerns the rigid form computed in the final step.

Property 2 *Let D be a finite set of structures and G a rigid grammar:*
IF $D \subseteq \mathcal{FL}_{GAB}(G)$ THEN
(1) $\exists \sigma : \sigma(GFC(D)) \subseteq \langle G, Constraints(G)\rangle$
(2) $RGC(D)$ exists and $\exists \tau : \tau(RGC(D)) \subseteq \langle G, Constraints(G)\rangle$
(3) $D \subseteq \mathbf{FL}_{GAB}(GFC(D)) \subseteq \mathbf{FL}_{GAB}(RGC(D)) \subseteq \mathcal{FL}_{GAB}(G)$

Proof of property 2 (1) : each $F_i \in D$ corresponds to (at least one) *GAB*-derivation in G, we choose one such $\mathcal{P}_i$ for each F_i. Each $F_i \in D$ also corresponds to an annotated structure computed at step (3) of the *RGC* algorithm, that we write $\mathcal{Q}_i$. We first focus on these structures for a given i and define gradually a substitution σ. Each node of F_i labelled by rule $/_{e^+}$ ($\backslash_{e^+}$ is to be treated similarly) is such that :
- when considered in $\mathcal{Q}_i$, it is labelled by a type that we write t_j with successors labelled by t_j/x'_j and x_j (from left to right) for some j;
- when considered in $\mathcal{P}_i$, it is labelled by a type A_j with 2 successors labelled by some types A_j/B_j and Γ_j (from left to right) such that Γ_j is a formula and $\Gamma_j \vdash B_j$ is valid in NL.
We then take $\sigma(x_j) = \Gamma_j$ and $\sigma(x'_j) = B_j$. We get more generally :

$$\begin{cases} \text{(i) if } t \text{ labels a node in } \mathcal{Q}_i \\ \qquad\qquad \text{then } \sigma(t) \text{ labels the corresponding node in } \mathcal{P}_i \\ \text{(ii) } \sigma(x_j) \vdash \sigma(x'_j) \text{ is valid in NL and belongs to } Constraints(G) \end{cases}$$

where (ii) is clear by construction, and (i) is proved by induction on the length of t : as a basis, we consider the root node S that is invariant by σ, and the variable cases with the definition of $\sigma(x_j)$, we apply otherwise the hypothesis for t_j to $\sigma(t_j/x'_j) = \sigma(t_j)/\sigma(x'_j)$ or $\sigma(x'_j\backslash t_j) =$

[3] with equality up to renaming
[4] by $G \subseteq G'$, we mean the assignments in G are also in G'

$\sigma(x'_j)\backslash\sigma(t_j)$. Properties (i) (ii) give (1) ■
Proof of property 2 (2): we take σ as in (1), this shows that unification succeeds for the family of types of $GF^+(D)$ (it admits a unifier since $\sigma(GF^+(D)) \subseteq G$, where G is rigid); we consider a most general unifier σ_{mgu}, such that $RGC(D) = \sigma_{mgu}(GFC(D))$, which provides the existence of τ such that $\sigma = \tau \circ \sigma_{mgu}$; rewriting 2(1) gives 2(2) ■
Proof of property 2 (3): from $D \subseteq FL_{GAB}(GFC(D)$ (clear by construction) $RGC(D) = \sigma_{mgu}(GFC(D))$ and from property 2(2) ■

Lemma 3 (Incrementality) *IF* $D \subseteq D' \subseteq \mathcal{FL}_{GAB}(G)$ *where* G *is rigid THEN* $\exists\eta : \eta(RGC(D)) \subseteq RGC(D')$ *(written* $RGC(D) \sqsubseteq RGC(D')$*).*

Proof : suppose G is rigid with $D \subseteq D' \subseteq \mathcal{FL}_{GAB}(G)$; we already know from Proposition 2(2) that $RGC(D)$ and $RGC(D')$ are defined.
Let us show $RGC(D) \sqsubseteq RGC(D')$;
let σ_{mgu} and σ'_{mgu} denote the respective most general unifier computed by step 5 of the RGC algorithm for D and D';
since $D \subseteq D'$, σ'_{mgu} is also a unifier for D therefore
(i) $\exists\eta : \sigma'_{mgu} = \eta \circ \sigma_{mgu}$;
on the other hand, from D to D', we only add new generalized FA-structures, thus new variables in step 3 of the algorithm, new types and new constraints in step 4, thus:
(ii) $D \subseteq D'$ implies $GFC(D) \subseteq GFC(D')$ (modulo variable renaming)
also from the algorithm:
(iii) $RGC(D) = \sigma_{mgu}(GFC(D))$ and $RGC(D') = \sigma'_{mgu}(GFC(D'))$
all of which we get $RGC(D') = \eta \circ \sigma_{mgu}(GFC(D'))$;
hence (inclusion modulo variable renaming):
$\underbrace{\eta \circ \sigma_{mgu}(GFC(D))}_{=\eta(RGC(D))} \subseteq \underbrace{\eta \circ \sigma_{mgu}(GFC(D'))}_{=RGC(D')}$ ■

Theorem 4 (Convergence) *Let* G *be a rigid grammar, and* $\langle F_i\rangle_{i\in\mathbf{N}}$ *denote any enumeration of* $\mathcal{FL}_{GAB}(G)$*, RGC converges on* $\langle F_i\rangle_{i\in\mathbf{N}}$ *to a grammar with variables and constraints having the same language.*
$\exists p_0 \forall p > p_0 : RGC(D_p) = RGC(D_{p_0}), \mathbf{FL}_{GAB}(RGC(D_p)) = \mathcal{FL}_{GAB}(G)$
where $D_p = \{F_1, \ldots, F_p\}$

Proof : given G rigid $\{G', \exists\sigma : \sigma(G') \subseteq G\}$ is finite (up to renaming) and from the above lemma: $RG^+(D_p) \sqsubseteq RG^+(D_{p+1}) \sqsubseteq \ldots \sqsubseteq G$, (using property 2) ; therefore $\exists p'_0, \forall p > p'_0 : RG^+(D_{p+1}) = RG^+(D_p)$ (up to renaming). The constraints part of $\langle G, C\rangle$ computed by the algorithm is such that C only involves subformulas of the types in grammar G (this holds for the general form and its substitution instances as well). Since after p_0, the grammar part is stationary, and since its set of subformulas is finite, the constraint part must also converge after p'_0 :

$\exists p_0 > p'_0 : \forall p > p_0 : RC^+(D_{p+1}) = RC^+(D_p)$.
The language equality is then a corollary using property 2(3) ■

1.6 Conclusion and remarks

We have proposed a learning algorithm that is unification-based, polynomial according to the size of the input data (unification can be performed in linear time using a suitable data structure), and that can be performed incrementally with new structured sentences.

We recall that a sentence generated by any *NL* gammar can be recognized in polynomial time Aarts and Trautwein (1995). The output of the *RGC* algorithm is a grammar with variables and constraints that also represents the memoized information used to obtain a polynomial parsing algorithm from a grammar without constraint.

Another perspective is the extension of the result and the algorithm exposed here to other variants of categorial grammars; this should apply to NL allowing empty sequents ; another candidate is the associative version, where the forward and backward rules could be still generalized allowing a sequence of types instead of a single type as argument.

References

Aarts, E. and K. Trautwein. 1995. Non-associative Lambek categorial grammar in polynomial time. *Mathematical Logic Quaterly* 41:476–484.

Bechet, Denis and Annie Foret. 2003. k-valued non-associative Lambek grammars are learnable from function-argument structures. In *Proccedings of WoLLIC'2003, volume 85, Electronic Notes in Theoretical Computer Science*.

Bonato, Roberto and Christian Retoré. 2001. Learning rigid lambek grammars and minimalist grammars from structured sentences. *Third workshop on Learning Language in Logic, Strasbourg* .

Buszkowski, W. 1997. Mathematical linguistics and proof theory. In van Benthem and ter Meulen (1997), chap. 12, pages 683–736.

Buszkowski, Wojciech and Gerald Penn. 1990. Categorial grammars determined from linguistic data by unification. *Studia Logica* 49:431–454.

de Groote, Philippe and François Lamarche. 2002. Classical non-associative lambek calculus. *Studia Logica* 71.1 (2).

Foret, Annie and Yannick Le Nir. 2002. Lambek rigid grammars are not learnable from strings. In *COLING'2002, 19th International Conference on Computational Linguistics*. Taipei, Taiwan.

Gold, E.M. 1967. Language identification in the limit. *Information and control* 10:447–474.

Kanazawa, Makoto. 1998a. *Learnable Classes of Categorial Grammars*. Studies in Logic, Language and Information. Stanford, California: Center for the Study of Language and Information (CSLI) and The European association for Logic, Language and Information (FOLLI).

Kanazawa, Makoto. 1998b. *Learnable classes of categorial grammars*. Studies in Logic, Language and Information. FoLLI & CSLI. distributed by Cambridge University Press.

Lambek, Joachim. 1961. On the calculus of syntactic types. In R. Jakobson, ed., *Structure of language and its mathematical aspects*, pages 166–178. American Mathematical Society.

Moortgat, Michael. 1997. Categorial type logic. In van Benthem and ter Meulen (1997), chap. 2, pages 93–177.

van Benthem, J. and A. ter Meulen, eds. 1997. *Handbook of Logic and Language*. Amsterdam: North-Holland Elsevier.