PROOF-NETS AND DEPENDENCIES

Alain LECOMTE

GRIL
Université Blaise Pascal
34 Avenue Carnot
63037- CLERMONT-FERRAND cedex
(France)

Abstract

Proof-Nets (Roorda 1990) are a good device for processing with categorial grammars, mainly because they avoid spurious ambiguities. Nevertheless, they do not provide easily readable structures and they hide the true proximity between Categorial Grammars and Dependency Grammars. We give here an other kind of Proof-Nets which is much related to Dependency Structures similar to those we meet in, for instance (Hudson 1984). These new Proof-Nets are called Connection Nets. We show that Connection Nets provide not only easily interpretable structures, but also that processing with them is more efficient. [1]

1.Introduction

Nowadays, two formalisms are very attractive in Natural Language Processing:

- Categorial Grammars, and
- Dependency Grammars.

Numerous studies try to shed light on their similarities and differences. We may quote for instance works by Hudson (1984, 1990), Barry & Pickering(1990), Hausser (1990), Hepple (1991). One interesting particularity common to these two formalisms seems to be the capacity of leading to an incremental processing, which, in turn, leads to an on-line processing.

Moreover, these formalisms are now very well known. *Categorial Grammars* have been much studied recently, particularly since the article of Ades and Steedman (1982) and the re-discovering of previous works done by Lambek (1958, 1961). The most comprehensive form taken by Categorial Grammars is the Lambek Calculus, studied by many authors like Moortgat (1988, 1990), Buszkowski (1986, 1988), Desclés (1990) etc. Since the recent work by J-Y Girard (see for instance Girard 1987), which led to the framework of Linear Logic, it has become apparent that the Lambek Calculus amounts to a non-commutative version of a sub-system of Linear Logic, where a structural rule forbids sequents with an empty antecedent.

Semantic properties of this system have been studied by Buszkowski (1986, 1988) and Wansing (1990). Two models are often given: one consists of residuation semigroups spread over free semigroups, and another one is given by the directional typed lambda-calculus.

Dependency Grammars are originating from earlier works by the French linguist Tesnière (1965). They were theoretically studied by Gaifman, who demonstrated theorems on the Generative Capacity of Dependency Grammars. We will consider here that the formalism of "Word Grammar" (Hudson 1984, 1990) is representative of this trend.

Our purpose in this communication is to show that *building dependency structures* gives an other kind of semantics for the Lambek Calculus and various subsystems. This semantics is useful in that it will allow us to conceive extensions of the Lambek Calculus. Moreover, the correspondance proposed between these two aspects provides us with a method of parsing related to the conception of "parsing as deduction", together with a method for avoiding spurious ambiguities. We will show that it is isomorphic to the method of *proof-nets* (Girard 1987, Danos and Regnier 1989, Roorda 1990, 1991), but that it has the advantage over this last method of being more efficient and of providing more clarity on the result of processing. The devices we obtain are more readable, because they are interpretable in terms of dependency structures. Otherwise, the parsing method can be an incremental one.

2. The Method of Proof-Nets in the Lambek Calculus

The problem of spurious ambiguities in Categorial Grammar is very often discussed (see for instance Hendriks and Roorda (1991)). A proof-net is a device which contains all the equivalent proofs of the same result. As Roorda (1990) says: "A proof-net can be viewed as a parallellized sequent proof [...] It is a concrete structure, not merely an abstract equivalence class of derivations, and surely not a special derivation with certain constraints on the order in which the rules must be applied."

The principles of construction of proof-nets are related to the inference rules of the Lambek Calculus, when it is viewed as a sequent calculus. If we here omit the product, we have the following rules, which belong to two different types:

[1] I am indebted to Dirk Roorda for fruitful discussions during a brief visit I made in Amsterdam in Spring 1991

Binary rules (or **type-2** rules):
(where Θ is a non-empty sequence of categories, and Γ and Λ are arbitrary sequences of categories)[2]

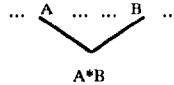
[L/]: $$\frac{\Theta \rightarrow B \quad \Gamma, A, \Lambda \rightarrow C}{\Gamma, A/B, \Theta, \Lambda \rightarrow C}$$

[L\]: $$\frac{\Theta \rightarrow B \quad \Gamma, A, \Lambda \rightarrow C}{\Gamma, \Theta, B\backslash A, \Lambda \rightarrow C}$$

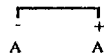
Unary rules (or **type-1** rules): (Γ is non empty)

[R/]: $$\frac{\Theta, B \rightarrow A}{\Theta \rightarrow A/B}$$ [R\]: $$\frac{B, \Theta \rightarrow A}{\Theta \rightarrow B\backslash A}$$

In these rules, contexts (Θ, Γ, Λ) are "static". That means that they can neither be contracted, nor expanded, nor permuted. They play no role in the application of rules. So, it is convenient to "forget" them and to represent the rules according to schemes similar to:

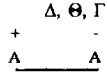


Such a scheme is called *a link*. Other types of links are provided by the *identity axiom*: [ax] $A \rightarrow A$, which becomes:



and by *the cut-rule*:

[cut]: $$\frac{\Theta \rightarrow A \quad \Delta, A, \Gamma \rightarrow C}{\Delta, \Theta, \Gamma \rightarrow C}$$ which becomes:



Links associated to rules belong to two types:

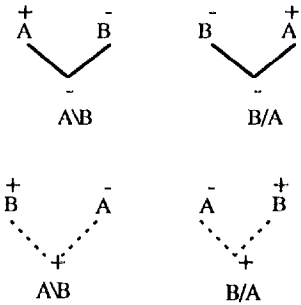
type-2 links corresponding to type-2 rules (depicted by lines)

type-1 links corresponding to type-1 rules (depicted by dashed lines)

D. Roorda (1990) has shown the following theorem:

Theorem: (Soundness and Completeness of Proof-Nets w.r.t. L.)

If the rules [L\], [L/], [R\], [R/] are represented by the following links:



then a sequent $\Gamma \rightarrow \Delta$ is a theorem of the (Product free) Lambek Calculus if and only if we can build, starting with this sequent and applying links recursively, a connected planar graph having the following property: for each application of a type-1 link, every suppression of one of the two dashed lines leaves the graph connected.

Examples:

$a \rightarrow b / (a \backslash b)$ is a theorem: see figure (1) below.

$b / (a \backslash b) \rightarrow a$ is not a theorem: see figure (2) below.

figure (1):

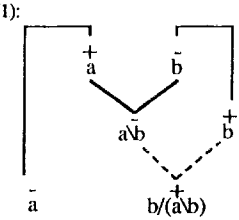
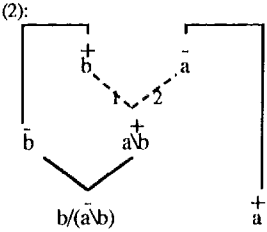


figure (2):



In this last case, we see that the suppression of the edge 2 leads to disconnection.

3.Dependency Structures

A dependency structure associated to a sentence is *a tree* on the words of this sentence. Edges represent dependency links such that the source of an edge is considered as the *head* and the target as a *dependant*. Hudson (1984) gives criteria to distinguish heads and dependants. It is an open question whether a head can be viewed as a functor, the dependant being viewed as an argument. The facts that criteria involve agreement and

[2] We use here the notation introduced by J. Lambek, according to which the argument category is always under the slash, in such a way that A/B means a category which becomes an A if a B is met on the right , and B\A a category which becomes an A if a B is met on the left.

that according to the Keenan's thesis: "functors agree with their argument" seem in favour of identification. But other scholars disagree, like Moortgat and Morrill, who introduce in their recent works, four notions: *head, dependant, functor* and *argument*. Nevertheless, we will accept the first thesis in the following, adopting the conception of Barry and Pickering (1990) on this subject.

Another problem appears in the necessity of accounting for structures with *multiplicity of heads* (in the case of the control of infinitives for instance) because this necessity leads to graphs which are no longer trees, but *dags*.

We assume that a dependency structure is a *graph on words*. In a first step, we will consider only trees. The approach will be that of a semantic interpretation in terms of trees, similar to what we do when we give a semantic interpretation of logic formulae in terms of sets. The usual operators like / and \ will be interpreted as *connection operations* in an algebra of trees. In a second step, we will have to modify this interpretation in order to obtain not only *application* and *composition* but *division* too.

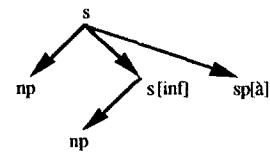
4.Operations on Trees

We start with a set of directed trees associated to lexical entries. (see figure (3) below).

figure (3)

connait:

promet



These trees are called *initial* trees. The initial state of a representation of the structure of a sentence consists in an ordered sequence of these initial trees. Then, at each step, we build a new tree obtained by connection of previous trees. These operations are: (cf Lecomte 1990)

- *left-linkage*
- *right-linkage*

A tree G_1 is *right* (resp. *left*) *linkable* to a tree G_2 iff:

1) G_1 and G_2 are adjacent, G_2 being adjacent to the right (resp. left) of G_1

2) G_1 has a rightmost (resp leftmost) branch the first edge of which is right (resp left) directed and the maximal sub-tree attached to this first edge *entirely covers* a *continuous subtree* of G_2.

The by-product of the right-linkage (resp left-linkage) of G_1 with G_2, when G_1 is right (resp left) linkable to G_2 is the tree G_3 obtained as the *union* of G_1 and G_2, modified in the following way:

The rightmost right-directed (resp. leftmost left-directed) first-level edge of G_1 is connected to the root of G_2, and the subtree of G_2 covered by the maximal subtree attached to this edge is said to be *marked* in G_2. Left (resp. right)-directed edges of G_2 which are not marked remain *free* and take precedence in the left-to-right (resp right-to-left) order of first-level edges over those remaining free in G_1.

We can introduce *restrictions* on these operations:

we will call *restriction*-**AB** the following constraint: the subtree of G_2 covered by the subtree of G_1 must be identical to the whole tree G_2,

restriction-**C**: at most the rightmost (resp leftmost) branch of G_2 may be uncovered.

restriction-**Crec**: a right (resp left) subtree of G_2 may be uncovered

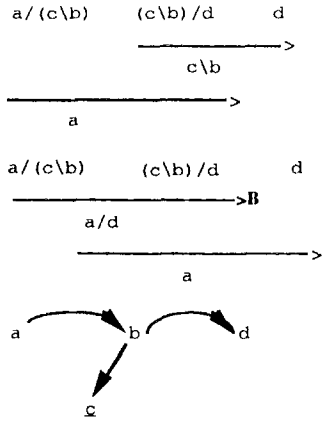
restriction-**Cmix**: at most the rightmost (resp leftmost) *or* the leftmost (resp rightmost) branch of G_2 may be uncovered

Definition: we call *connection tree* every initial tree and every tree obtained by the application of linkage operations on earlier connection trees (according to eventual restrictions).

We claim that such a system gives an interpretation of very simple categorial grammars, depending on the restrictions we select. Like similar constructions (Steedman 1991) where general principles such as *Adjacency*, *Directional Consistency* and *Directional Inheritance* are explained in terms of a more detailed analysis of categories, this system is suited to express such generalities. Because of the structure of linkage operations, these principles are obvious. *Adjacency* and *Directional Consistency* are contained in the definition. *Directional Inheritance* comes from the fact that we never allow to change anything in the *labels* of edges (the fact that they are left or right directed). We only allow to change the *status* of an edge (*free* to *bound* or *marked*). In so doing, we reach, like Steedman does, the conclusion that so-called *Dysharmonic Composition Rules* are consistent with these principles (even if they are not with the Lambek Calculus!).

A connection system eliminates spurious ambiguities because when they are bound, links are undefeasable : there is no way of re-doing something that was primilarly done with success. In this respect, the calculus on trees concurs with the well known method of chart-parsing. (see figure (4): there is only one tree for two reductions by means of Cancellation Schemes).

figure (4):



Moreover, a connection system provides us with a semantics for Dependency-Constituency Grammars, in the tradition of Barry and Pickering (1990) and Mark Hepple (1991).

5.Connection and Identification: an Extension of Connection Systems

5.1.The Need for Division Rules

It is obvious that the previous system does not include any kind of Division Rules or any kind of Type-Raising Rule. So, it cannot provide any analysis for sentences with extraction, as for instance:

le livre dont je connais le titre est sur la table
(the book the title of which I know is on the table)

because in such an analysis, we have to transform a regular n (*titre*) into a functorial category which requires a noun-modifier on its right (n/(n\n)).
We shall define a new connection system which is a conservative extension of the previous one (except for the admissibility of Dysharmonic Rules). We will call it: the *Connection Net System*.
As for the proof-nets, we want to demonstrate *theorems* that have a sequent form like: $\Gamma \rightarrow X$, where Γ is a non empty sequence of categories and X is a category. We distinguish two kinds of connection trees: those which are on the right-hand side of the sequent we want to demonstrate, and those which are on the left-hand side. When we are viewing the problems in a natural-deduction way, we can say that the first are the trees *to build* and the second are those which are *used* in this task. We will call the first*right-trees* and the second *left-trees*. The set of left-trees and right-trees at any stage will be called a *Construction Net*.

Schematically, operations are not merely connections because connections can only expand elementary trees towards more complex ones. And we need operations to reduce the complexity of a tree. For instance, to show the usual rule of Type-Raising: $a \rightarrow b/(a\backslash b)$ we have to show that the right-tree associated to $b/(a\backslash b)$ reduces to something isomorphic to a. The fact that, generally, the converse $(b/(a\backslash b)\rightarrow a)$ is not true results from the fact that the same reduction is not possible when the same tree is put on the left-hand side. This exemplifies the fundamental dissymetry of the calculus.

5.2.Type-1 Edges and Type-2 Edges

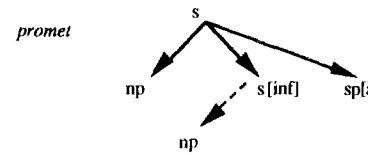
We will then distinguish two sorts of edges and two sorts of nodes in a connection tree: type-1 edges and nodes and type-2 edges and nodes.
Definition: A *type-2 edge* in a connection tree is:
- an odd level edge in a left-tree, or
- an even level edge in a right-tree

A *type-1 edge* in a connection tree is:
- an even level edge in a left-tree, or
- an odd level edge in a right-tree

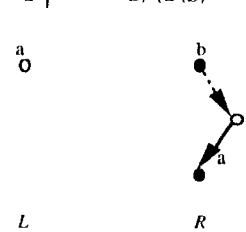
A *type-i* (i =1, 2) *node* is the target of a type-i edge.
Roots are type-1 nodes if in a left-tree, and type-2 nodes if in a right-tree.

Two nodes are said to be *complementary* if they have not the same type.

Examples: figure (5)
a) a new tree assigned to a lexical entry:



b) a pair (L, R) associated to a sequent:



Definition: we call *identification link* either a non-directed edge which links two identical nodes which are complementary, one in a left-tree, the other in a right-tree, or a type-1 directed edge linking two complementary nodes having same label.
We call *connection link* every link we shall be able to establish, according to the following conventions, between a type-1 node, which is the ending point of a

type-2 edge, and a type-2 node which does not belong to the same tree.

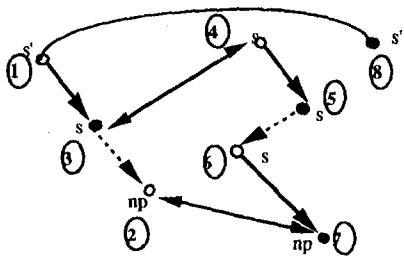
5.3.Nodes-numbering

Rule: each node of the initial construction net receives a number, called its *degree*, according to the following rules:

- *for a type-2 edge*:
  - if it is *right* directed, the degree of its source is less than the degree of all the nodes below it,
  - if it is *left* directed, the degree of its source is greater than the degree of all the nodes below it,
  - for two type-2 edges, children's degrees of the leftmost branch are less than those of the rightmost branch.
- *for a type-1 edge*:
  - if it is *right* directed, the degree of its source is greater than the degree of all the nodes below it,
  - if it is *left* directed, the degree of its source is less than the degree of all the nodes below it.
  - for two type-1 edges, children's degrees of the rightmost branch are less than those of the leftmost branch.

The lowest degree of the right successor of an initial tree is the successor of the greatest degree of this latter tree.

Example of such a numbering: figure (6)



L *R*

intervals:

s'-s' : [1 _ 8]

s-s : [3 _ 4]

np-np : [2 _ 7]

s-s : [5 _ 6]

Each link is now associated to a pair of degrees, called its *interval*.

From now on, L and R will denote respectively: the left hand side and the right hand side of a Construction Net. The Construction Net will be denoted by: <L | R>.

5.4.Linking the Nodes

Nodes will be linked according to the following principles:

COMPLEMENTARITY: two nodes are linked only if they have the same label and they are of *complementary types*.

NON-OVERLAP: the linking of all the nodes in the Construction Net must meet the non-overlap convention, which stipulates that given two arbitrary intervals, *either one contains the other or they are disjoint*.

Theorem: (*Conservativity of Connection Operations*) The Non-Overlap condition is a conservative extension of the conditions on connection (restriction C^{rec}) stipulated in §4. That means: every connection system based on C^{rec}, when translated in the Connection Net System, follows this convention.

5.5.Building a Correct Net

Definition: Given an ordered sequence of left-trees L and a right-tree R, we will say that L and R *yield a correct net* iff there is a linkage of all the nodes in the Construction Net <L | R>, which gives a *connected* graph, respects the *complementarity principle* and the *non-overlap* principle, and is such that: *when all the type-1 edges are removed, the graph remains connected.*

The fundamental result is the following:

Theorem: (*Soundness and Completeness w.r.t.* A[3]) Let $\Gamma \rightarrow A$ be a sequent expressed in the Product-Free Lambek Calculus, where Γ is a non empty sequence of categories and A is a category, let L be the sequence of left connection trees associated to the elements of Γ and R be the right tree associated to A, the sequent is a theorem if and only if L and R yield a correct net. In other terms: the Connection Net System is sound and complete w.r.t. the Product-Free Lambek Calculus.

Examples: figure (6) shows that:

s/(s/np) s/((s/np)\s) |-- s

is a theorem of the Lambek Calculus.

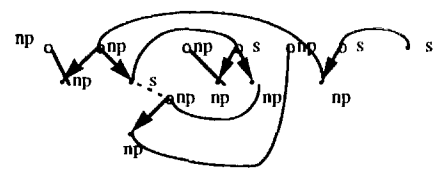
Figure (7) below gives a correct net for the analysis of the sentence:

le livre dont je connais le titre est sur la table

[3] A is the usual designation of the Product-Free Lambek Calculus (see Zielonka 1981).

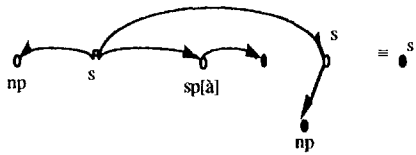
(figure 7):

np np\np/(s/(np\np)) np np\s/np np np\s ⊢ s



Theorem: (*Categorization of Links*) In a correct net, links are either identification links or connection links.
Corollary: A net is correct if and only if all nodes are either identified or connected.
Definition: we call *Tree on words* the tree obtained from a correct net by merging connected nodes and removing identification links. In the case of an identification link consisting in a type-1 edge, the link *and* the nodes linked by it are removed, and the adjacent type-2 edges are merged.

Example: figure (8): *Pierre promet à Marie de venir*

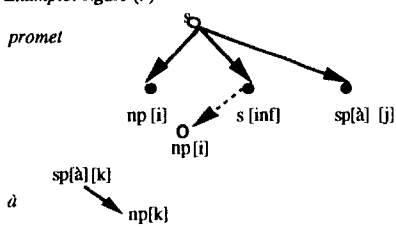


6. Building Dependencies

Obtaining a Dependency Structure from a tree on words amounts to doing little transformations on the correct net obtained by "equating" an ordered sequence of initial trees to a node representing a primitive category. These transformations involve indexing nodes in such a way that:

- indices of two different initial trees constitute two disjoint sets.
- indices inside an initial tree may be identical (if we want to express a coreference)
- linking two nodes results in identifying their indices.

Example: figure (9)



After getting a tree on words, we identify two distant nodes having the same index: we call the new node obtained: a *shared node*.
Finally, we can say that dependencies are obtained in the following stages:

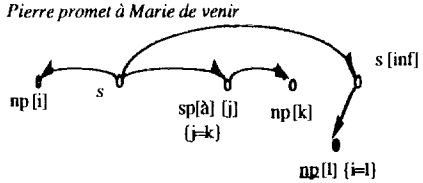
1- *indexing* the nodes having the same label and belonging to different initial trees by different variables, taken in a set {i, j, ...) (the distribution of indices inside an initial tree being set by the lexicon) [INDEXING-step].

2- *building the net* corresponding to the assertion that the sequence is of type s [NET-step].

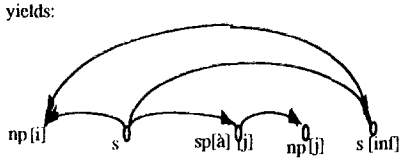
3- *suppressing* the nodes identified by type-1 edges of the left-hand side and all the identification links [COLLAPSING-step].

4- if the same index appears on distinct nodes having the same label: *merging* them [MERGING-step]
Example: figure (10)

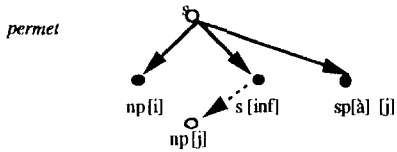
Pierre promet à Marie de venir



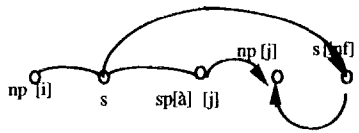
yields:



but, with *permet* instead of *promet*:



Pierre permet à Marie de venir:



7.Remarks and Conclusion

This method of Connection Nets has many advantages over other methods.
Firstly, compared to classical strategies in the sequent calculus, it avoids spurious ambiguities and in so doing, it improves efficiency of searching the solution.
Secondly, compared to the method of Proof-Nets, it gives more clarity to the resulting structures. It is more efficient too, because the stage of checking the connexity when suppressing a branch of a type-1 link is replaced by a stage where the connexity is checked only once: when we have removed all the type-1 edges. The corresponding stage in Proof-Nets is usually named *switching*. In the early method by Girard which used the "long trip condition", there was a switch for each link and that gave an exponential-time algorithm (in the number of links). In the method defined by Roorda, only type-1 links lead to switches. The reason lies in the necessity of checking that a type-1 link is not used to connect two subsets of the net, which would not be connected without it. (Let us recall that a type-1 link refers to a unary rule). In our method, switches are completely avoided.
Thirdly, it can be done incrementally. The reason is that the numbering of nodes is consistent with the order of initial trees. Thus, at each stage of the processing from left to right, we may have a beginning net which represents the present state of the processing. Here, the properties of left-associative grammars (Hausser 1990) are met.
Finally, a very few transformations are needed in order to obtain graphs on words which can be really interpreted as Dependency Structures.

BIBLIOGRAPHY

Ades, A. and Steedman, M.: 1982, 'On the Order of Words', *Linguistics and Philosophy*4, 517-558.

Barry, G. and Morrill, G.: 1990. (Eds) *Studies in Categorial Grammar*. Edinburgh Working Papers in Cognitive Science, Volume 5. Centre for Cognitive Science, Edinburgh.

Barry, G & Pickering, M.: 1990; 'Dependency and Constituency in Categorial Grammar', in Barry, G. and Morrill, G. 1990 and Lecomte, A. 1992.

Buszkowski 1986; 'Completeness Results for Lambek Syntactic Calculus', *Zeitschr. f. math. Logik und Grundlagen d. Math.* **32**, 13-28.

Buszkowski 1988; 'Generative Power of Categorial Grammar',in R.Oehrle, E.Bach et D.Wheeler (eds) *Categorial Grammars and Natural Languages Structures*.

Danos, V. and Regnier, L.: 1989, 'The structure of multiplicatives' *Arch. Math. Logic*, **28**, 181-203.

Desclés, J.P.: 1990, *Langages applicatifs, langues naturelles et cognition*, Hermes, Paris.

Girard 1987; 'Linear Logic', *Thoretical Computer Science* **50**, 1-102.

Hausser, R.: 1990, *Computation of Language*, Springer-Verlag, Berlin, Heidelberg.

Hendriks, H. and Roorda, D.: 1991, 'Spurious Ambiguity in Categorial Grammar', *deliverable of the ESPRIT project BRA 3175 DYANA*.

Hepple, M.:1991, 'Efficient Incremental Processing with Categorial Grammar' *Proceedings of ACL* 1991, Berkeley.

Hudson, R.A.: 1984, *Word Grammar*, Blackwell, Oxford.

Hudson, R. A.: 1990, *English Word Grammar*, Blackwell, Oxford.

Lambek, J.: 1958, 'The Mathematics of Sentence Structure',*AmericanMath.Monthly* 65,154-170.

Lambek, J.: 1961, 'On the Calculus of Syntactic Types' in *Structure of Language and its Mathematical Aspects*, AMS, Providence.

Lecomte, A.: 1990, 'Connection Grammars: a Graph-Oriented Interpretation of Categorial Grammars' in Lecomte, A. (ed), 1992.

Lecomte, A.: 1992. (ed.)*Word Order in Categorial Grammar*, ADOSA, Clermont-Ferrand.

Moortgat, M.:1988; *Categorial Investigations. Logical and Linguistic Aspects of the Lambek Calculus*, Dordrecht, Foris.

Moortgat, M.: 1990, 'Proof nets, partial deduction and resolution - Part 2' in Lecomte, A. 1992.

Oehrle, R.,Bach, E.and Wheeler, D. (eds): 1988, *Categorial Grammars and Natural Languages Structures*, D. Reidel Publishing Company, Dordrecht et Boston.

Roorda, D.: 1990, 'Proof nets, partial deduction and resolution - Part 1' in Lecomte, A. 1992.

Roorda, D.: 1991, *Resource Logics: Proof-theoretical Investigations*, PhD Thesis, Faculteit van Wiskunde en Informatica, Amsterdam.

Steedman, M.: 1991, 'Type-Raising and Directionality in Combinatory Grammar', *Proceedings of ACL* 1991, Berkeley.

Tesnière, L.: 1965, *Eléments de syntaxe structurale*, Klincksieck, Paris.

Wansing, H.: 1990, *Formulaes-as-types for a Hierarchy of Sublogics of Intuitionistic Propositional Logic*, Gruppe für Logik, Wissentheorie und Information an der Freien Universität Berlin.

Zielonka, W.: 1981, 'Axiomatizability of Ajdukiewicz-Lambek Calculus by Means of Cancellation Schemes' *Zeitschr. f. math. Logik und Grundlagen d. Math.* 27, 215-224.