The Non-Associative Lambek Calculus

Logic, Linguistic and Computational Properties

Università degli Studi di Bologna
Dottorato in Filosofia
Tesi di Dottorato
Settore disciplinare: M-FIL/02
LOGICA E FILOSOFIA DELLA SCIENZA

Autore:

Matteo Capelletti

Coordinatore:	Prof. W. Tega
Relatore:	Prof. M. Ferriani
Correlatore:	Prof. G. Sandri

This dissertation is typeset using LaTeX.

Indice

I Categorial Grammars 9

1 Introduction 11
1.1 Categorial linguistics 13
1.1.1 Semantics 15
1.1.2 Logical syntax 16
1.2 Categorial parsing 19
1.2.1 Products 25
1.3 Non-associative Lambek calculus 27
1.4 Overview 30

2 Formal Background 31
2.1 Languages 32
2.2 Grammars 34
2.2.1 Deductive systems 34
2.3 Context-free grammars 37
2.4 Categorial grammars 41
2.5 Lambda terms 45
2.5.1 Typed lambda calculus 47
2.5.2 Lambda calculus in Haskell 53
2.6 The Lambek calculus 56
2.7 Product types 60
2.7.1 Linguistic analysis in *NL* 63
2.8 Multi-modal Type-logical Grammars 68
2.9 Formal properties of categorial grammars 72

3 Automated Reasoning 75
3.1 Problems 75
3.2 Deductive parsers 78
3.3 Bottom-up parsers 78

3.3.1 Basic categorial grammars 80
3.3.2 Product rules 80
3.4 Earley style parsing 85
3.4.1 Earley system for $\mathcal{CF}$ 86
3.5 Implementations 88
3.6 Agenda-driven, chart-based procedure 89
3.7 Tabular parsing 92
3.7.1 Tabular CYK 92
3.8 Parses 98
3.9 Parse forest 98
3.9.1 Parse forests for $AB^{\otimes}$ grammars 101
3.10 From $AB^{\otimes}$ to AB 103
3.10.1 Currying 105
3.10.2 Uncurrying 108
3.11 Parsing approaches for Lambek systems 113
3.12 Conclusion 115

II The Non-associative Lambek Calculus 117

4 A Case Study: Cliticization 119
4.1 Information Structure 121
4.1.1 Hendriks' approach 123
4.1.2 Clitics and pregroups 124
4.2 Italian clitic cluster and core clause 126
4.2.1 Notation for cases 127
4.2.2 Formal treatment 129
4.2.3 Clitic attachment 130
4.3 Clitic left-dislocation 132
4.3.1 Redundant arguments 132
4.3.2 Proposals 133
4.3.3 Non-relevant reasoning 135
4.3.4 A lexical solution 136
4.4 Conclusion 138

5 Normal Derivations in $\mathbf{NL}$ 141
5.1 Alternative formulations of $\mathbf{NL}$ 143
5.2 Normal derivations 146
5.3 Automatic construction of normal derivations 148
5.3.1 Expansion and reduction 149

5.3.2 Remarks on expansions and reductions 158
5.4 Proof terms 160
5.5 Connection to parsing 163
5.6 Conclusion 165

6 Normal Derivations and Ambiguity 167
6.1 Eliminating redundancies 168
6.2 Unique normal derivations 171
6.3 Enumerating readings 173
6.4 Conclusion 178

7 A look at cut elimination 179
7.1 Admissibility of Cut 180
7.1.1 Main cuts 181
7.1.2 Permutations 184
7.1.3 Cycles 187
7.2 Conclusion 188

8 Conclusion 189

Ringraziamenti

Questa tesi conclude cinque anni di ricerca condotti in parte presso l'Università di Bologna ed in parte presso l'Università di Utrecht. Ringraziare le molte persone che mi hanno in vari modi aiutato durante questi anni è oltre che un dovere, un grande piacere.

Vorrei innanzi tutto ringraziare i miei supervisori: il Professor Maurizio Ferriani e il Professor Giorgio Sandri, che hanno seguito con interesse il mio percorso di dottorato. Inoltre, la Professoressa Claudia Casadio, per avermi sostenuto nel mio lavoro fin dall'inizio e per avermi introdotto alla grammatica categoriale prima e in seguito al Professor Moortgat col quale ho svolto gran parte della mia ricerca in questi anni.

In secondo luogo, vorrei ringraziare i miei supervisori all'Università di Utrecht: il Professor Michael Moortgat e il Professor Jan van Eijck, per l'attenzione che hanno sempre dedicato al mio lavoro e le idee che mi hanno aiutato a sviluppare.

Inoltre, devo ammettere che cinque anni non sono la durata standard di un dottorato e, a tal proposito, devo ringraziare il Professor Walter Tega, in qualità di coordinatore del Dottorato in Filosofia, per avermi concesso tutto il tempo necessario a concludere il mio lavoro nel modo migliore.

Inoltre, Raffaella Bernardi, Cristina Paoletti, Willemijn Vermaat, Herman Hendriks, i colleghi di Utrecht e di Bologna, che in vari modi e in vari momenti mi hanno aiutato nell'arco degli anni.

In fine, un ringraziamento speciale va alla mia famiglia e a Saara Jäntti: anche, e forse soprattutto, grazie al loro supporto sono riuscito a portare a termine questo lavoro.

Ovviamente, di ogni errore, la responsabilità è solo mia.

Part I

Categorial Grammars

Chapter 1

Introduction

This book is a comprehensive study of the *non-associative Lambek calculus*, a logical and linguistic system presented by Joachim Lambek in 1961. The non-associative Lambek calculus belongs to the syntactic framework of *categorial grammar*: a formalism for linguistic description grounded on the distinction of syntactic categories between *basic* categories and *functor* categories[1].

Categorial grammar is in turn a branch of the scientific discipline of *computational linguistics*, an approach to linguistics which tries to model natural language with the methods and tools of *mathematics* and *computer science*. During the past fifty years (in fact, from the advent of computers), these two sciences have interacted with linguistics in a valuable way and language technology has always been one of the most active fields of research in computer science. Mathematics provided the linguists with expressive formalisms, appropriate to a precise and concise description of languages. Meanwhile, computers offered the place for large scale evaluation of the mathematical statements about language. The possibility of implementing the mathematical descriptions of specific languages on a computer, and of testing the adequacy of such descriptions, has led to sophisticated methods of linguistic analysis, as so called *tabular parsing* methods and the representation of ambiguous parsing in terms of *shared forests* (see chapter 3).

The main contribution of this study is the application of these methods

[1] To make such distinction more concrete since the beginning, let us give a simple example. If the word *John* belongs to the basic syntactic category n of proper names (or noun phrases) and *John whistles* belongs to the basic category s of well formed sentences, then *whistles* can be seen as a *functor* that takes as argument a syntactic object of category n and returns a syntactic object of category s (we might, for instance, denote such a functor $n \rightarrow s$ to make more explicit the input-output relation that such a category expresses).

to non-associative Lambek grammars with product categories.

Tabular parsing methods were first designed for context-free grammars by [Younger, 1967, Earley, 1970], and later extended to other syntactic frameworks that were based on rewriting rules (we refer to [Shieber et al., 1995] for a survey of these methods from a *deductive parsing* perspective). However, only a few attempts to apply these parsing strategies to Lambek style categorial grammars have been made: while [Morrill, 1996] applies the CYK algorithm to categorial proof nets, [Hepple, 1992] and [König, 1994] develop different strategies for handling hypothetical reasoning in the Lambek calculus (the system of [Lambek, 1958]). Instead, [Finkel and Tellier, 1996] show that the application of dynamic programming techniques to grammars based on the Lambek calculus becomes relatively simple, after the grammar has been reduced to a set of context-free rules according to the method of [Pentus, 1993].

Furthermore, all these algorithms have been designed for *product-free* fragments of (either associative or non-associative) Lambek systems. While this aspect may not be relevant in the case of grammars based on the associative calculus, we believe that it becomes a serious limitation in the case of grammars based on the non-associative calculus. These grammars, in fact, have been proposed in order to generate *trees* representing the *structural descriptions* of grammatical expressions. Product categories are a powerful tool for enforcing specific constituent structures, as we argue in chapter 2. On the other hand, we will see in chapter 3 that their introduction rule is affected by a high degree of indeterminacy, which makes their efficient application difficult. In regard to this, we will show in chapter 3 how it is possible to handle the product rule in a computationally efficient way.

Concerning hypothetical reasoning, which is the most distinguishing feature of Lambek style categorial grammars and, computationally, the hardest to control, our algorithm will rely on a method of *normal derivations construction* inspired by the works [Kandulski, 1988] and [Le Nir, 2004]. This method will be studied in great detail in chapters 5 and 6.

In chapter 4, we will explore the linguistic expressivity of the non-associative Lambek calculus in relation to complex phenomena of Italian syntax as cliticization and clitic left-dislocation.

We conclude this book by presenting the cut-elimination algorithm for our original formulation of the non-associative Lambek calculus. We show that the cut-elimination algorithm for non-associative Lambek calculus integrated with proof terms gives rise to term equations that can be used for proof normalization.

1.1 Categorial linguistics

Generative grammar stems from Chomsky's seminal works [Chomsky, 1957, 1959]. In these works, Chomsky established the theory of grammar as a branch of the mathematical theory of *recursive functions*. A grammar of a language L is a function whose range is L. Furthermore, as natural languages contain an infinite number of expressions, such a function cannot simply consist of a list of pairs of categories and well formed expressions of those categories. In order to make non-trivial claims about the structure of natural language, the grammar should be a *finite* characterization of the language. As such, it should incorporate recursive mechanisms.

Transformational grammar, starting from [Chomsky, 1957, 1965], consisted of a set of context-sensitive rewriting rules. Generalized phrase structure grammar of [Gazdar et al., 1985], and its contemporary offspring head-driven phrase structure grammar, presented in [Pollard and Sag, 1994], adopted a context-free grammar architecture, extended with special mechanisms for handling discontinuous dependencies.

According to the definition of [Chomsky, 1957], *categorial grammars* are *generative grammars*. However, rather than being based on the notion of *rewriting rule* as context-free grammars, categorial grammars are based on an explicit formulation of syntactic categories in terms of functor and argument categories, and on a notion of derivation proper to deductive systems. In this linguistic framework, the internal structure of syntactic categories encodes the combinatorial properties of the expressions to which they are associated. In turn, the combinatorial properties are expressed by the abstract inference schemes of a logical system.

The great majority of generative linguists adopted rewriting systems to design natural language grammars. Indeed, rewriting systems can be seen as deductive systems based on a set of non-logical axioms, namely, the set of rewriting rules or productions, and on the only inference rule of *cut*. When they are in Chomsky normal form the productions of such systems can be divided into *lexical* and *non-lexical*. Lexical productions anchor the vocabulary of the language to the syntactic categories of the grammar, and non-lexical productions define the way complex linguistic expressions are composed out of smaller ones.

One of the advantages of adopting categorial grammar is that the set of non-logical axioms is limited to lexical assignments and that complex expressions are derived according to a small number of *abstract inference schemes*. This makes the lexicon the only idiosyncratic component of the grammar and maintains a universal set of inference schemes for every language specific grammar.

The shift of linguistic information from rule based encoding to lexicon based encoding is often called *lexicalism*. Today, most grammatical frameworks assume some variant of the lexicalist approach, as the main differences among human languages are lexical, for example [Gazdar et al., 1985], [Pollard and Sag, 1994] and [Chomsky, 1995] to mention only a few. In the case of categorial grammar, one can speak of *radical* lexicalism as the *only* information driving the inferential process stems from the lexicon.

While empirically adequate, the division of the grammar architecture between a language specific component, the lexicon, and a language universal component, the deductive system, has proved to be extremely fruitful for the theoretical foundation of categorial logic. The deductive system of a Lambek style categorial grammar is a fragment of *linear logic*, the logic system introduced in [Girard, 1987]. The logic in [Lambek, 1958] has been recognized as the non-commutative intuitionistic multiplicative fragment of linear logic and in fact, also as the first fragment of linear logic in history. In the past twenty years, linear logic has been one of the most intensively investigated fields of research and the categorial grammar framework has profited from this research both methodologically and theoretically, see [Moortgat, 1997b, Moot, 2002].

The acknowledgment of the distinction between *logical* rules and *structural* rules is one of the cornerstones of linear logic. One of its main assumptions, which is expressed in [Girard, 1995] in a very clear way, is that the meaning of the logical constants of a given deductive system depends on the possibilities of structural manipulation available in the deductive system itself.

Contemporary research in categorial grammar is based on the so called *multi-modal* setting of [Moortgat, 1997b]. Such system admits a variety of category forming operators and partition the deductive module of the grammar into a core module to which every logical connective obeys (the so called *pure logic of residuation*) and a structural module, whose postulates can interact only with specific configurations of the connectives. This has the great advantage of assuming a highly restrictive base logical system and of introducing restructuring operations only when and where it is required.

Among the formal tools introduced by linear logic, proof nets are a *redundancy free* representation of logical proofs. Deductive systems may be affected by so called *spurious ambiguity*: different syntactic derivations may correspond to the same semantic object. Instead, different proofs nets are always different semantic objects. A pleasant property of categorial logic is that it is included in a fragment of linear logic, the multiplicative fragment, for which proof nets have a clear computational interpretation.

Proof nets have been applied to linguistics by several authors; we refer to [Moot, 2002] as the most representative work on this approach to linguistics and to the works he draws on. On the other hand, we will not adopt the proof net formalism for our proposes, but more traditional axiomatic formulations of categorial logic. These kinds of formulations, used, among others, by [Lambek, 1958, 1961, Zielonka, 1981, Kandulski, 1988], offer the possibility of detailed investigations of the logical and computational properties of different sets of axioms and inference rules. As we will see in chapter 6, the problem of spurious ambiguity will not affect our system as a consequence of our method of constructing derivations.

1.1.1 Semantics

One of the most attractive features of categorial grammar is its direct link with semantics. Model theoretic semantics was first applied to natural language by R. Montague in a series of papers, many of which have been collected in [Montague, 1974], see also [Dowty et al., 1981, Partee et al., 1990]. Such approach assumes a close correspondence between the construction of the syntactic derivation of an expression and the construction of its semantic representation. This is indeed what the *compositionality principle* states: the meaning of a complex expression depends on the meaning of its parts and *on the way in which the parts are put together*. By "way in which the parts are put together" we shall mean *syntax*. We will see, in chapter 2, that the meaning representation language adopted by Montague, and by the great majority of the linguists working in the model-theoretic tradition, is based on typed *lambda calculus*: a formal language for speaking about *functions*. In generative grammar, the compositionality principle amounts to the assignment of lambda terms to the basic elements of the vocabulary, the assignment of a semantic term to complex expressions being completely determined by the syntactic derivation.

In the context of phrase structure grammar, the implementation of the principle of compositionality assumes the form of a *rule-to-rule approach*, see for instance [Montague, 1970a,b, Gazdar et al., 1985]. This means that to every syntactic rule of the grammar a specific semantic rule is associated.

Categorial grammar improves notably on this situation. In first place, the *syntactic* notation based on the function-argument distinction, which is in fact grounded on a semantic distinction, makes the syntax-semantics parallelism completely transparent. Besides, as we discussed before, and we will see in detail in chapter 2, only a small number of *logical* rules is required for building syntactic derivations since all the combinatorial properties of the expressions are encoded in the lexicon. The correspondence between proofs and typed lambda terms, known as Curry-Howard correspondence (we refer to [Girard et al., 1989] and to [Hindley, 1997] for clear expositions of the topic) or formulae-as-types correspondence, allows to see contemporarily the process of constructing a syntactic deduction as the process of building a lambda term or a semantic representation. Thus, if the lexical items of a categorial grammar are assigned lambda terms encoding their meaning, the syntactic process constructing a derivation for a complex expression should be contemporarily seen as a semantic process building the semantic representation of that expression.

In this book, we will examine two ways of assigning terms to syntactic derivations. In particular, while we will adopt the lambda calculus for natural language semantics and present a new method for assigning lambda terms to axiomatic proofs, we will also adopt a more refined term language, in the style of [Lambek, 1988, 1993, Moortgat and Oehrle, 1997] for the semantics of categorial proofs. This term language is isomorphic to Lambek logic and will enable us to define reduction of proofs to normal form and equivalence of proofs as syntactic congruence of terms in normal form.

1.1.2 Logical syntax

Several kinds of categorial grammar have been presented in the past century, and further refinements and extensions to the system are continuously being developed. The first formalization of syntactic categories as functors and arguments appeared long before the advent of generative grammar. [Ajdukiewicz, 1935] was one of the firsts to apply the distinction between complete and incomplete expressions to the analysis of natural language. He was formalizing and applying to natural language concepts from Leśniewski's mereology and Husserl's notion of semantic category, with mathematical tools derived from Russel's theory of types and from Frege's functional notation. We refer to [Casadio, 1988] for a discussion of the historical background of categorial grammars. A *functor category*, in Ajdukiewicz notation, was an object of the form $\frac{a}{b}$. A linguistic resource w of category $\frac{a}{b}$ combines with another linguistic resource w' of category b to give a complex resource ww' of category a.

In [Bar-Hillel, 1953], the functorial notation for syntactic categories was further refined in order to distinguish left-functor categories and right-functor categories. The rules governing the combination of such functor categories to their arguments can be expressed as follows.

(1.1) A linguistic resource w of category a/b combines with another linguistic resource w' of category b to its right to give a linguistic resource ww' of category a.

(1.2) A linguistic resource w of category $b\backslash a$ combines with another linguistic resource w' of category b to its left to give a linguistic resource $w'w$ of category a.

Actually, more general patterns of function-argument composition are adopted in so called Ajdukiewicz Bar-Hillel systems, of which the previous ones are instances. We will see later in more detail the Ajdukiewicz Bar-Hillel system and discuss also this second kind of composition rule.

In this introductory chapter, we wish to emphasize the close relationship between categorial grammar and deductive systems or typed logic. Observe that the rules 1.1 and 1.2 express order sensitive variants of the inference rule of *modus ponens*, which can be expressed as follows.

(1.3)
$$\frac{p \quad p \Rightarrow q}{q}$$

This rule can be spelled out as asserting that 'if p and if p, then q are given, then q follows'. One can see the left and right slashes of categorial grammar as left and right implications of a deductive system in which the rule of modus ponens has been split in two rules. The reader will recognize in the following inference schemes a compact formulation of the rules 1.1 and 1.2. We simply replaced the expression 'linguistic resource z of category x' with the notation $z :: x$ and we put the given syntactic material in the top part of the tree and the resulting material in the lower part.

(1.4)
$$\frac{w :: a/b \quad w' :: b}{ww' :: a}$$

(1.5)
$$\frac{w' :: b \quad w :: b\backslash a}{w'w :: a}$$

To give a concrete example consider the following deduction of the sentence *John likes music*:

(1.6)

$$\dfrac{John :: n \qquad \dfrac{likes :: (n\backslash s)/n \quad music :: n}{likes\ music :: n\backslash s}}{John\ likes\ music :: s}$$

Here, *likes* is taken to be a functor looking for a noun phrase to the right to return another functor looking for a noun phrase to the left to give a sentence. Shortly *likes* :: $(n\backslash s)/n$.

At this stage of the presentation of categorial grammars, the correspondence with deductive systems is only partial. In fact, deductive systems have also rules for inferring implicational formulas which categorial grammars, in the sense of Ajdukiewicz and Bar-Hillel, lack.

(1.7)

$$\dfrac{\begin{matrix}[p]\\ \vdots \\ q\end{matrix}}{p \Rightarrow q}$$

Such a rule states that 'if q follows from the assumption of p, then $p \Rightarrow q$ holds'. The notation $[p]$ indicates that the assumption p is 'discharged' as a result of the inference rule, in other words, p is no longer among the assumptions from which the conclusion $p \Rightarrow q$ is obtained.

A great enrichment to the theory of categorial grammar was the *syntactic calculus* of [Lambek, 1958]. Lambek added to the inference rules of the Ajdukiewicz Bar-Hillel calculus the following rules.

(1.8) If, with the aid of a *rightmost* assumption b, we proved that the linguistic resource $w\,b$ is of category a, then we can conclude that w is of category a/b.

$$\dfrac{w\ b :: a}{w :: a/b}$$

(1.9) If, with the aid of a *leftmost* assumption b, we proved that the linguistic resource $b\,w$ is of category a, then we can conclude that w is of category $b\backslash a$.

$$\dfrac{b\ w :: a}{w :: b\backslash a}$$

The rules 1.8 and 1.9 can be seen as the oriented variants of rule 1.7. Consider now, as an example, a categorial lexicon containing the assignments *John* :: n and $left :: (s/(n\backslash s))\backslash s$. We want to prove that $John\ left :: s$. Observe that, if no other lexical assignments are given for *John* and *left*, the Ajdukiewicz Bar-Hillel system does not derive such conclusion. On the other hand, such a sentence is derivable in a Lambek system.

(1.10)

$$\dfrac{\dfrac{\dfrac{\mathit{John} :: n \quad n\backslash s :: n\backslash s}{\mathit{John}\ n\backslash s :: s}}{\mathit{John} :: s/(n\backslash s)} \qquad \mathit{left} :: (s/(n\backslash s))\backslash s}{\mathit{John}\ \mathit{left} :: s}$$

In this derivation, the assumption $n\backslash s :: n\backslash s$ can be called a non-lexically anchored *hypothesis* (or non-lexical hypothesis, for short), while the other hypotheses are all lexically anchored. Being an identity, such hypothesis is universally valid, thus it is possible to assume it in the derivation process. The use of such hypotheses is often called *hypothetical reasoning* and is undoubtedly the most distinguishing feature of Lambek style grammars with respect to all other generative frameworks.

We will see that among the benefits of hypothetical reasoning there is the reduction of the size of the lexicon, and especially an elegant account of syntactic and semantic dependencies in local and non-local domains.

To find a solution to the difficulties that hypothetical reasoning involves from the computational perspective is one of the central concerns of this thesis and chapter 5 will be dedicated to this topic. In example 1.10, we can see that while the hypothesis is triggered by the higher order verb, it interacts in first place with the n resource. This raises problems because several linguistic resources may occur in between the trigger and the point where the hypothesis is used. The kernel of the procedure that we develop in definition 82 of chapter 5 is in fact a method for handling hypotheses in an efficient way. In the next section, we discuss in more detail the problems we face in building a parsing algorithm for Lambek grammars and we give an informal overview of our solution.

1.2 Categorial parsing

Together with the assumption that the grammar generates the language, the theory of generative grammar requires that each generated expression is assigned a structural description. The structural description, usually shown as a *tree*, indicates

- the hierarchical grouping of the parts of the expression into constituents,

- the grammatical category of each constituent and

- the left-to-right order of the constituents.

Technically, *parsing* is the process of assigning a structure to the expressions generated by the grammar. As such, it presupposes that the expression to be parsed has already been *recognized* as belonging to the language generated by the grammar. Furthermore, as an expression can be assigned several structures by a grammar, thus parsing can assume existential or universal import, depending on whether it addresses the task of returning one or all the structures of the expression, respectively. Both tasks can be accomplished by means of the method of so called *parse forest*: a compact representation of all possible derivations that an expression may have in a given grammar. Chapter 3 will be dedicated to these issues.

In the second half of the last century, several ways of parsing with context-free grammars have been developed. These techniques have been later extended to other grammar formalisms based on rewriting rules, as feature grammars, tree-adjoining grammars, by [Vijay-Shanker, 1987], indexed grammars by [van Eijck, 2005] or combinatory categorial grammars by [Vijay-Shanker and Weir, 1990]. The most interesting are the so called *tabular* parsing methods for context-free grammars, see [Aho and Ullman, 1972]. These algorithms are called tabular because their methods consists in the construction of a table whose cells contain non-terminal symbols of the grammar. Each cell is identified by a pair of integers indicating the portion of the input string covered by the non-terminals contained in the cell, if any. These algorithms are examples of *dynamic programming*, a programming technique that relies on storing the results of partial computations (memoization) for their subsequent reuse in case of need.

The most famous parsing algorithms for context-free grammars are the Cocke-Younger-Kasami algorithm (CYK) and the Earley parsing algorithm, which can be found in [Younger, 1967, Earley, 1970, Aho and Ullman, 1972]. They are tabular parsing methods and work in time cubic on the length of the input string.

These algorithms can be presented as deductive systems adopting the *deductive parsing* formalism of [Shieber et al., 1995]. This is a simple and clean formalization of dynamic parsing based on the deductive approach to syntax proper to logical grammars. For instance, in the case of Ajdukiewicz Bar-Hillel grammars we can define the following rules (we write $w_i \dots w_j :: c$ for the sequence of words from w_i to w_j is of category c):

(1.11) If $w_{i+1} \dots w_k :: a/b$ and $w_{k+1} \dots w_j :: b$, then $w_{i+1} \dots w_j :: a$. Formally,

$$\frac{(i, a/b, k) \quad (k, b, j)}{(i, a, j)}$$

(1.12) If $w_{i+1} \dots w_k :: b$ and $w_{k+1} \dots w_j :: b \backslash a$, then $w_{i+1} \dots w_j :: a$. Formally,

$$\frac{(i, b, k) \quad (k, b\backslash a, j)}{(i, a, j)}$$

In this formalism, parsing is seen as a deductive process regimented by the linear order of the grammatical expressions occurring in the input string. Deductive parsers consist of a specification three components:

1. a set of axioms, for instance:

   $$(0, a_1, 1) \dots (n-1, a_n, n)$$

   where $w_i :: a_i$ is in the input grammar for all $1 \leqslant i \leqslant n$ and n is the length of the input string,

2. a set of inference rules, for instance those in 1.11 and 1.12, and

3. the goal of the computation, for example $(0, s, n)$.

The axioms link the syntactic categories assigned by the grammar to each word in the input string to the portion of the input occupied by that word. Inference rules specify how to assign a syntactic category to a larger portion of the input string on the basis of the grammar rules and of the syntactic categories assigned to adjacent smaller portions of the input. The goal defines which category should be assigned to the entire input.

Section 4.4 of [Shieber et al., 1995] is dedicated to the problems arising in the application of the deductive parsing methodologies to Lambek calculi. As our work represents a solution to these problems for the calculus of [Lambek, 1961], we discuss here in detail these problems, quoting from [Shieber et al., 1995]. Lambek calculi

> are better presented in a sequent-calculus format. The main reason for this is that those systems use nonatomic formulas that represent concurrent or hypothetical analyses. [...] The main difficulty with applying our techniques to sequent systems

> is that computationally they are designed to be used in a *top-down direction*. For instance, the rule used for the hypothetical analysis [...] has the form
>
> $$\frac{\Gamma B \vdash A}{\Gamma \vdash A/B}$$
>
> *(cfr. our rule 1.8)*. It is reasonable to use this rule in a goal-directed fashion (consequent to antecedent) to show $\Gamma \vdash A/B$, but using it in a forward direction is impractical, because B must be *arbitrarily* assumed before knowing whether the rule is applicable. [...] such undirected introduction of assumptions just in case they may yield consequences that will be needed later is computationally very costly. Systems that make full use of the sequent formulation therefore seem to require top-down proof search[2].

The rules of the Lambek calculus that we presented before in 1.4, 1.5, 1.8 and 1.9, are, indeed, sequent-style natural deduction rules, see [Moortgat, 1997b]. Consider now the rules 1.11 and 1.12, the deductive parsing rules of a (non-associative) Ajdukiewicz Bar-Hillel calculus. Observe that a *forward application* of these rules is the only natural one. In fact, the formula b which disappears in the conclusion is *given* in the premises. Instead, a top-down approach would have to *guess* which formula b has disappeared from the conclusion, and this would be a quite complex operation[3].

On the other hand, the *hypothetical* rules 1.8 and 1.9 deserve a special treatment. Firstly, we wish to remark that as in the case of the *application* rules 1.4 and 1.5, the hypothetical rules express a conditional whose antecedent is given by the premise(s), and whose succedent is given by the conclusion. This means that while an antecedent-to-succedent (or forward, or bottom-up, or from premises to conclusion) application of such rules guarantees soundness of the approach, a goal-directed (or succedent-to-antecedent, or top-down, or from conclusion to premise(s)) application

[2] Cursive mine.

[3] [Shieber et al., 1995] do not specify whether they refer to a *Gentzen style* sequent system or a sequent-style natural deduction system as the one we gave. In the first case, rules 1.4 and 1.5 would be replaced by the *left introduction rules* of the slashes. For such rules, forward application has not such a clear interpretation as for the natural deduction rules we are examining. On the other hand, the hypothetical rule on which the argument in [Shieber et al., 1995] is based, is common to both formalisms. In fact, section 4.2 of [Shieber et al., 1995] presents a deductive parsing formulation of combinatory categorial grammar based on the CYK algorithm, whose *forward* and *backward* application rules are very much alike the 1.11 and 1.12 rules that we are discussing.

does not. This, in turn, implies that even in the search of a proof for a valid input sequent of a Lambek system, a top-down approach will often encounter non-valid sequents and our computers will spend a great amount of resources to prove that in fact they are not provable. Instead, a bottom-up approach guarantees that only valid sequents are encountered in the search of a proof for a given Lambek sequent.

Secondly, the *arbitrary* status of the hypothesis B in the previous quatation is simply not true. In example 1.10, we have seen that the occurrence of the non-lexical hypothesis was linked to the third order verb assignment. Indeed, it would be rather strange if such hypothesis B could be *any* formula, because sequent systems (either Gentzen style or natural deduction style) enjoy the *subformula property*. Hypotheses can, in fact, be identified as the *negative premises of a par link* in the proof net formalism. Thus, a trivial unfolding of the lexical categories may determine which non-lexical hypotheses will be needed in the derivation.

However, it is true that the hypothetical rules of a Lambek system, as they have been formulated before, introduce a high degree of indeterminacy in proof search. This is serious problem for automated theorem proving for Lambek systems.

As we said, a first constraint that can limit such indeterminacy in the choice of hypotheses is represented by the *subformula property*.

(1.13) In a cut free Gentzen style deduction, only subformulas of the conclusion sequent appear in the premise sequent(s).

As discussed in [Lambek, 1958], this is the key property that guarantees decidability of the syntactic calculus as it implies that the search space in the proof of a Lambek sequent can be bound to sequents made of subformulas of the input sequent. What we want to emphasize is that in proving a sequent we may rely only on the subformulas of this sequent. Thus also hypotheses can be chosen from such range of formulas.

As stated in 1.13, the subformula property seems to suggest a top-down approach. However, this is not the only option as we can see: an example of bottom-up algorithm taking advantage of the subformula property is discussed in [Buszkowski, 2005].

A second way of constraining the hypothetical rules consists in defining more specialized variants of them. We will, in fact, adopt this is the approach in chapter 5. The hypothetical rules allow to derive what we will call, adopting the terminology of [Buszkowski, 1986, Kandulski, 1988], *expanding patterns*. These are characteristic theorems of the Lambek systems as the famous *type lifting*, $a \to b/(a\backslash b)$[4], which we saw implicitly already in example 1.10. Lifting can be derived, with the rules given before, as follows.

(1.14)

$$\frac{\dfrac{a \to a \quad a\backslash b \to a\backslash b}{a\ a\backslash b \to b}}{a \to b/(a\backslash b)}$$

This can be seen as a proof that everything that is assigned a category a can infer the category $b/(a\backslash b)$, for any category b.

If we want to avoid the hypothetical rule in this derivation, one option is to add such theorem as an axiom of the deductive system, as [Kandulski, 1988] does or, alternatively, to introduce a special rule for it, as [de Groote, 1999] does. For instance, we could have derived $a \to b/(a\backslash b)$ with the following rule.

(1.15)

$$\frac{b \to a\backslash c' \quad c' \to c}{a \to c/b}$$

The lifting scheme, $a \to b/(a\backslash b)$, follows immediately from this rule without need of the hypothetical rules.

(1.16)

$$\frac{a\backslash b \to a\backslash b \quad b \to b}{a \to b/(a\backslash b)}$$

The advantages of using a specialized rule as 1.15 for capturing *part of* the hypothetical reasoning of Lambek grammars are several and we will discuss them in detail in the following chapters. Here it is worth emphasizing that the rule has a clear bottom-up computational interpretation as it appeals to specific patterns occurring in the premises from which a subformula (namely c') is canceled in the conclusion.

The expanding patterns may look problematic also for another reason. Let us consider lifting once more. This law states that for any formula a and for any formula b, if a is derivable, then also $b/(a\backslash b)$ is. Now, if we implement lifting as a rule that for any input formula a returns a formula $b/(a\backslash b)$, for some formula b, we enter an infinite loop, since the output of this rule can be directly taken as an input for its iterated application.
A better approach is to assume $b/(a\backslash b)$ as the given material, and to see a as

[4] The arrow → can be read as 'derives'.

the result of a 'simplification' of this formula. For instance, we may capture the meaning of lifting through a transition from $b/(a\backslash b) \to x$ to $a \to x$ as the following.

(1.17)

$$\frac{b/(a\backslash b) \to x}{a \to x}$$

This rule has, again, a clear bottom-up interpretation. In fact, $b/(a\backslash b)$ is among the given material. The rule performs a well determined reduction and the conclusion is shorter than the premise. As for every formula x, $x \to x$ holds, we derive lifting by simply replacing $b/(a\backslash b)$ for x in 1.17.

Of course, rule 1.15 or 1.17, does not exhaust the contexts in which the hypothetical rules can apply. On the other hand, we can aim at identifying all the patterns that the hypothetical rules capture in a given fragment of the Lambek calculus. As we just saw for the case of 1.15 and 1.17, the resulting specialized rules convey important informations that can help in the context of parsing. As proved in [Buszkowski, 1986, Kandulski, 1988, de Groote, 1999] and [Le Nir, 2004], in the case of the non-associative Lambek calculus, such an enterprise indeed is possible, and also relatively simple in the case of this fragment, in the sense that a small number of specialized rules can replace the hypothetical rules altogether.

Our point of departure will be [Kandulski, 1988]. In this work, Kandulski proves the equivalence of non-associative Lambek grammar *with product formulas* and context-free grammars. This result relies on the possibility of reducing the derivations of non-associative Lambek grammars with product formulas to those of Ajdukiewicz Bar-Hillel grammars with product formulas by assuming that all hypothetical reasoning takes place 'at the lexical level'.

1.2.1 Products

In our presentation of Lambek categorial grammar and of its parsing properties, we never encountered product formulas. Their role in linguistic description will be discussed in the next section. Now we examine the problems that product formulas (actually, their rules) raise from a computational perspective. The rule for forming complex phrases of product category can be given as follows.

(1.18) If the linguistic resource w is of category a and the linguistic resource w' is of category b, then the resource $w\,w'$ is of category $a \otimes b$.

$$\frac{w :: a \quad w' :: b}{w\,w' :: a \otimes b}$$

One may observe that this rule can be correctly regimented along the linear order constraints of deductive parsing, as we did for the application rules in 1.11 and 1.12.

(1.19) If $w_{i+1} \dots w_k :: a$ and $w_{k+1} \dots w_j :: b$, then $w_{i+1} \dots w_j :: a \otimes b$.

$$\frac{(i, a, k) \quad (k, b, j)}{(i, a \otimes b, j)}$$

On the other hand, such a constraint does not prevent an exponential explosion of the search space in automatic proof construction: given an input string of length n, there are $C(n)$ ways of applying this rule, where $C(n)$ is the Catalan number of n (a huge number!). Thus it is important to find a way to constrain the direct application of rule 1.19.

In chapter 3, we will see two methods to achieve this result. One based on the subformula property, and one based on the complete elimination of products. The first one restricts the application of rule 1.19 to generate only products $a \otimes b$ that are required by other functor categories. The second consists in a lexical transformation which has the effect of eliminating the product rule from the grammar. Roughly, the elimination of product can be seen as an application of the following inference schemes, attributed to Schönfinkel in [Lambek, 1958]. The symbol ↔ indicates that the inference is in both directions.

(1.20)

$$a/(b \otimes c) \leftrightarrow (a/c)/b$$

$$(c \otimes b)\backslash a \leftrightarrow b\backslash(c\backslash a)$$

One may observe that these inferences *do not* hold in the non-associative Lambek calculus, as they essentially rely on the structural rule of associativity. However, we will use them in a highly constrained way which will prevent the system from overgenerating. In fact, we will also be able to recover the deduction in the original grammar, through a simple term encoding and normalization.

1.3 Non-associative Lambek calculus

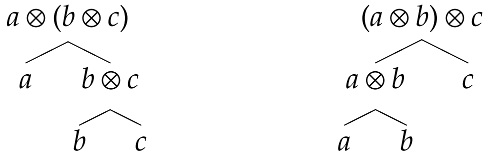
The non associative Lambek calculus, *NL* hereafter, was introduced in [Lambek, 1961]. The system resulted from the logic presented in [Lambek, 1958], called *syntactic calculus* by eliminating the structural rules of *associativity*.

(1.21) Associativity:

$$a \otimes (b \otimes c) \rightarrow (a \otimes b) \otimes c \qquad (a \otimes b) \otimes c \rightarrow a \otimes (b \otimes c)$$

Roughly, the associativity rules state that the branching structure assigned in a derivation is not relevant. For example, one can see them as asserting the interderivability of left-branching structures and right-branching structures.

(1.22)



More in general, a property of grammars based on the syntactic calculus is *structural completeness*: for any string in the language generated by such grammars all possible tree structures living on this string are derivable, see [Buszkowski, 1997]. Thus, one can simply change the type of the antecedent of a sequent from a tree of formulas to a *list* of formulas, *a b c*, as in [Hendriks, 1993], as lists are flat structures, insensitive to the hierarchical grouping of constituents.

Lambek introduced the non-associative logic in the categorial grammar framework to deal with tree structures representing the constituent analysis of the expressions: while in the syntactic calculus the antecedent of a sequent is a list of formulas, in *NL* it is a tree structure.

Although several categorial linguists adopt an associative regime for linguistic analysis, and indeed, associativity simplifies partly the job of the linguist, we believe that *NL* is a better starting point for linguistic description than the syntactic calculus.

Our first remark concerns the *weak generative capacity* of Lambek syntactic calculus and *NL*. Both logics generate the same class of languages, namely the *context-free languages*. The equivalence of non-associative Lambek grammars and context-free grammars was proved in [Kandulski, 1988], and we will examine in detail his method in chapter 5 and in chapter 6. Instead, the equivalence of Lambek grammars based on the syntactic calculus and context-free grammars, conjectured in [Chomsky, 1963], was proved in [Pentus, 1993]. As the syntactic calculus results from *NL* by adding the associativity rules, Pentus' proof shows that such extension does not increase the generative power of *NL*. Hence, our first reason to chose *NL* as a syntactic framework for natural language is of methodological nature. *NL* is a *simpler theory* of natural language than the syntactic calculus, in the sense that it appeals to a smaller number of axioms.

A second remark concerns the *computational* properties of the two systems. We have discussed how important an efficient implementation of these linguistic models is. On the other hand, [Pentus, 2003] proves that Lambek syntactic calculus is NP-complete. Hence, it is most unlikely that any efficient parsing algorithm will ever be discovered for this system. Instead, *NL* has much better parsing properties than the syntactic calculus. In fact, [de Groote, 1999] and [Buszkowski, 2005] prove that *NL* can be parsed in polynomial time.

Our last observation has to do with the *strong generative capacity*. The assignment of structural descriptions to the expressions of a language is a primary concern of generative linguistics, [Chomsky, 1957]. In this respect, *NL*, equipped with product categories, has all the tools for generating appropriate syntactic descriptions for grammatical expressions. We wish to underline here that the presence of products categories in the *NL* is not ornamental[5]. For example, one wants to distinguish the syntactic structures projected by heads of the form in A from those of the form in B, below.

(1.23)

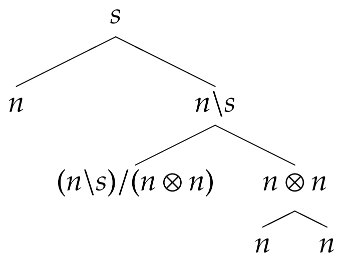
A	B
$a/(b \otimes c)$	$(a/c)/b$
$(c \otimes b)\backslash a$	$b\backslash(c\backslash a)$
$(a\backslash b)/c$	$a\backslash(b/c)$

For example, one may prefer to assign the ditransitive verb *gives* the category $(n\backslash s)/(n \otimes n)$, rather than the category $((n\backslash s)/n)/n$ which would be the only option available in the product free variant of *NL*. The differ-

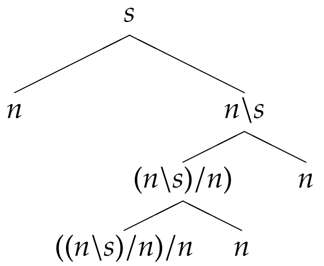
[5] As it is in the syntactic calculus due to the laws in 1.20.

ent structures projected by these two categories are given in the following examples.

(1.24) Structure projected by *gives* :: $(n\backslash s)/(n \otimes n)$.



(1.25) Structure projected by *gives* :: $((n\backslash s)/n)/n$.



Due to structural completeness, no such distinction is possible within the syntactic calculus. In fact, any of the previous two lexical assignments would generate both syntactic structures as well as all the others living on the same lexical categories.

A final issue that deserves some discussion is that of the generative power of *NL* with respect to other frameworks for natural language analysis. Context-free is generally considered a too restrictive class for natural language, as human languages exhibit types of dependencies which are beyond the scope of context-free grammars. Linguists believe that natural language lies somewhere in between context-free and context-sensitive, within the range of so called *mildly context-sensitive languages*. In the past decades, several formalisms for mildly context-sensitive grammars have been designed. Among others indexed grammars, [Aho, 1967, Gazdar, 1988], generalized phrase structure grammars, [Gazdar et al., 1985], head-driven phrase structure grammars, [Pollard and Sag, 1994], tree adjoining grammars, [Joshi et al., 1975, Joshi, 1985], combinatory categorial grammar, [Steedman, 2000b], minimalist grammars, [Stabler, 1997], and multi-modal type-logical grammars, [Moortgat, 1997b, Morrill, 1994].

Our work should be located within the multi-modal framework, [Moortgat, 1997b]. As we discussed before, *NL* is the basic inferential engine of this logical system in which more expressive logics are obtained from *NL* by addition of structural postulates, [Kurtonina and Moortgat, 1997]. Thus, our work addresses the core of the multi-modal setting and lay the ground for future works oriented to more refined options of structure management.

1.4 Overview

This book is organized as follows.

Chapter 2 introduces the framework of categorial grammar in a formal way. It defines the concepts and the notation that will be used throughout the book with some emphasis to functional implementation using the Haskell programming language.

In chapter 3, we formulate CYK style parsing systems for basic categorial grammars.

Chapter 4 gives some concrete examples of type-logical analysis of specific linguistic phenomena taken from Italian syntax.

Chapters 5 and 6 contain our main contribution. In definition 82, we present an effective procedure for proving two-formula sequents of *NL*. Such procedure is simple, elegant and efficient for most applications. It can be used to transform an *NL* grammar into a basic categorial grammar with product by finite lexical extension. In chapter 6 we show that such procedure is not affected by *redundancies* and we prove a beautiful result linking the number of semantic reading of *NL* sequents to the Pascal triangle.

Chapter 7 presents the so called cut-elimination algorithm with proof term labeling.

Chapter 2

Formal Background

In this chapter, we look in more detail at the notions informally introduced in the first chapter. We start with the definition of the basic notions of formal language theory as *string*, *language*, *grammar*, *derivation*, *generation*. We introduce context-free grammars and categorial grammars as *deductive systems*. The term *categorial grammar* includes various linguistic formalisms which share the assumption that expressions may belong to either 'complete' or 'incomplete' syntactic categories. However, the tools and the methods of these formalisms may be quite different. On one side, there are the *combinatory combinatorial grammars* of [Steedman, 2000b], originating from the combinatorial grammars of [Ajdukiewicz, 1935] and [Bar-Hillel, 1953]. On the other, the *logical grammars* of [Morrill, 1994] and [Moortgat, 1997b] among others, stemming from [Lambek, 1958]. The two systems differ both for theoretical and computational properties. In particular, the Ajdukiewicz Bar-Hillel systems that we will consider contain only rules for composing smaller structures into larger structures, while Lambek style categorial grammars contain rules for composing structures as well as rules for *decomposing* complex structures into simpler ones. This difference makes combinatorial grammars easier to handle under many respects, for instance in automatic proof search. On the other hand, the logical approach of Lambek, even in its extended formulation as *multi-modal* system, can be proved to be complete with respect to the appropriate models as shown in [Kurtonina and Moortgat, 1997][1].

In this chapter, we will see associative and non-associative Ajdukiewicz and Bar-Hillel grammars, the associative and non-associative Lambek cal-

[1] Which means that we have a syntactic method for proving all the valid statements of the system.

culus and the framework of multi-modal type-logical grammars.

As a methodological remark, we underline the fact that in developing the content of this book the Haskell programming language has played a great role. We tried to render our notation independent from the Haskell syntax, as specified in [Jones, 2003]. However, within the functional programming paradigm, Haskell represents an excellent environment to reason about mathematical statements, as shown in [Doets and van Eijck, 2004]. Therefore, our definitions may sometimes have more than the flavor of Haskell definition. This should allow the uninitiated reader to get acquainted with some basic notion of functional programming and eventually to implement the algorithms by straightforwardly translating our sugared notation into a Haskell code.

2.1 Languages

Languages are defined from their terminal elements, often called *words* or more in general *symbols*. In turn, symbols belong to a given vocabulary.

Definition 1. A *vocabulary* (or *alphabet*) is a finite non-empty set of *symbols*.

If the symbols in the vocabulary are of type a, we write the type of the vocabulary as $\{a\}$.

Example 1. Vocabularies:

- $V_0 = \{0, 1\}$, where $0, 1$ are of type *Int*(eger).
- $V_1 = \{a, b, c\}$, where a, b, c are of type *Char*.
- $V_2 = \{John, Mary, every, a, man, woman, loves, ismissing\}$,
  where all the elements of V_2 are of type *String*.

An object of type *String* consists in a list of objects of type *Char* in a given order. More precisely, we introduce the notion of list, below.

Definition 2. A *list* of objects of type a, denoted $List\,a$ is defined as

$$List\,a := \epsilon \mid H\,a\,(List\,a)$$

This definition states that a list of objects of type a is either ϵ (the empty list) or the result of the application of the type constructor H to an object of type a and to a list of objects of type a. Following the Haskell syntax, [Jones, 2003], instead of $List\,a$, we use the notation $[a]$ and instead of the constructor

H we use the colon : infix notation. The elements of a list are separated by commas. Again, according to the Haskell conventions, a prefix function f can be made infix by writing \`$f$\`. While an infix function g is made prefix by writing (g). Thus, $H = (:)$ and $: =$ \`$H$\`. The *type* of this function is written $(:) :: a \rightarrow [a] \rightarrow [a]$, which means that (:) is a function that takes in input an argument x of type a, and returns a function that takes in input a list $[x_1, \ldots, x_n]$ of objects of type a and returns as output a list $[x, x_1, \ldots, x_n]$ of objects of type a. Thus, for example, we have the following equivalences.

$$\begin{array}{lllll} [\,] & = & \epsilon & & \\ a\!:\!as & = & H\,a\,as & & \\ [a,b,c] & = & a : b : c : [] & = & H\,a\,(H\,b\,(H\,c\,\epsilon)) \end{array}$$

With this notation, we write the type *String* as [*Char*]. As usual, we use _ for the *wildcard*. We define the length of a list xs, denoted $|xs|$:

$$\begin{array}{ll} |[\,]| & = 0 \\ |(\_\!:\!as)| & = 1 + |as| \end{array}$$

Definition 3. The *concatenation*, $(+\!\!+) :: [a] \rightarrow [a] \rightarrow [a]$, of two lists xs and ys is defined as follows.

$$\begin{array}{lll} [\,] +\!\!+ ys & = & ys \\ (x\!:\!xs) +\!\!+ ys & = & x\!:\!(xs +\!\!+ ys) \end{array}$$

List concatenation has the following properties.

- $[\,] +\!\!+ ys = ys = ys +\!\!+ [\,]$
- $(xs +\!\!+ ys) +\!\!+ zs = xs +\!\!+ (ys +\!\!+ zs)$

Definition 4.

The *Kleene closure* of a set A of type $\{a\}$, denoted A^* and of type of type $\{[a]\}$, is the set of all lists over A.

The *positive Kleene closure* of a set A of type $\{a\}$, denoted A^+ and of type of type $\{[a]\}$, is the set of all non-empty lists over A.

A *language* over a vocabulary V is a subset of V^*

Example 2. Languages:

- $V_0^* = \{[], [0], [1], [0,1], [0,0], [1,1], [0,0,0], \ldots\}$,
- $V_0^+ = \{[0], [1], [0,1], [0,0], [1,1], [0,0,0], \ldots\}$,

- $L_1 = \{[0], [1], [0, 1, 1, 1]\}$ is a (finite) language over V_0.

By *string*, we mean a list symbols of some type. In order to simplify the notation, we may write a list of *Char* type objects, $[a, b, c]$, as *abc* and the concatenation of two lists *abc* and *de* simply as *abcde*. This convention may be extended to all the lists of unstructured symbols, like *Char*. When the objects are more structured, we can use white-spaces to separate the various tokens. For example, the list of integers $[1, 100, 3]$ is written 1 100 3 and a list of *String* type symbols $[John, walks]$ as *John walks*. We will use this simplified notation as far as it does not result ambiguous.
We extend the notion of concatenation to languages.

Definition 5. If L_1 and L_2 are two languages of type $\{[a]\}$, we write L_1L_2 the language consisting of all the strings xy such that $x \in L1$ and $y \in L_2$.
$(\overline{+\!\!+}) :: \{[a]\} \to \{[a]\} \to \{[a]\}$

$$L_1 \overline{+\!\!+} L_2 = \{x +\!\!+ y \mid x \in L_1,\ y \in L_2\}$$

2.2 Grammars

A grammar is a formal device that *characterizes* a language. This means that given a string in input, the grammar determines, in a *finite* number of steps, whether the string is in the language or not, [Chomsky, 1957, 1959]. A grammar can be seen as a kind of deductive system, subject to specific constraints. We will define in the next section deductive systems. Then, we will examine context-free grammars and categorial grammars as instances of deductive systems.

2.2.1 Deductive systems

We define a set syntactic categories. These are the basic objects that our deductive systems will manipulate.

Definition 6. *Syntactic categories.*

Let a set of *atoms*, A be defined as

$$A := S \mid NP \mid VP \mid N \mid \ \ldots \ \mid p_1 \mid p_2 \mid \ \ldots$$

Furthermore, let

$$\mathcal{F} := A$$

Later we will extend the data type $\mathcal{F}$ of formulas with other type constructors.

Definition 7. A **deductive system** D is a triple $\langle \mathcal{F}, AX, R\rangle$, where

- $AX = \{\langle \Gamma, \Delta\rangle \mid \Gamma \in F^*,\ \Delta \in F^+\}$ is the set of axioms. We show each $\langle \Gamma, \Delta\rangle \in AX$ as $\Gamma \to \Delta$.
- R is the set of inference rules proper to the system. Such rules are of the form

  $$\begin{array}{ll} \text{if} & \Gamma_0 \to \Delta_0 \text{ and} \ldots \text{and } \Gamma_n \to \Delta_n \\ \text{then} & \Gamma \to \Delta \end{array}$$

  We write rules as
  $$\frac{\Gamma_0 \to \Delta_0 \quad \ldots \quad \Gamma_n \to \Delta_n}{\Gamma \to \Delta}$$

The objects of the form $\Gamma \to \Delta$ are called *sequents*. In a sequent $\Gamma \to \Delta$, Γ is called the *antecedent* and Δ the *succedent*. The $\Gamma_0 \to \Delta_0 \ldots \Gamma_n \to \Delta_n$ in the "if" part of the rules are called *premises*, and $\Gamma \to \Delta$ in the "then" part is called *conclusion*. In what follows, we will present both context-free grammars and categorial grammars as deductive systems, adopting the perspective of *parsing as deduction* of [Pereira and Warren, 1983]. We denote $\Gamma[\Delta]$ an element of $\mathcal{F}^+$ with a distinguished occurrence of $\Delta \in \mathcal{F}^+$. The result of replacing $\Delta \in \mathcal{F}^*$ for a in $\Gamma[a]$ is denoted $\Gamma[\Delta]$.

Definition 8. A *tree* of objects of type a, denoted *Tree a*, is defined as follows

$$Tree\ a := Branch\ a\ [Tree\ a]$$

This definition states that a tree of objects of type a consists of a *Brach* constructor followed by an a type object, the *root*, and by a list of trees of type a, the brances of the tree. For simplicity, we write $Leaf\ a$ for $Branch\ a$ []. We show a tree of the form $Leaf\ x$ as x and a tree of the form $Branch\ r\ [t_0,\ t_1,\ \ldots,\ t_n]$ as

$$\frac{t_0 \quad t_1 \quad \ldots \quad t_n}{r}$$

However, we will usually work with binary trees.

Definition 9. Let a deductive system $D = \langle F, AX, R\rangle$ be given. We recursively define a *deduction* in D as

1. $Leaf\ a$ is a deduction, if $a \in AX$.

2. *Branch* $r\ [t_0, t_1, \ldots, t_n]$ is a deduction, if $t_0, t_1, \ldots, t_n$ are deductions, and r is the conclusion of an inference rule in R with the conclusions of $t_0, t_1, \ldots, t_n$ (in the order) as premises.

We introduce the notion of generation and of language generated.

Definition 10.

A deductive system D *generates* a sequent $\Gamma \rightarrow \Delta$, denoted $\vdash_D \Gamma \rightarrow \Delta$, if $\Gamma \rightarrow \Delta$ is the conclusion of a deduction in D.

The language generated by a deductive system D, denoted $L(D)$ is the set of sequents generated by D.

In the context of rewriting systems, which are the formal systems commonly adopted to describe context-free grammars, the notion of *derivation* is used more often than the one of deduction. We will make use both of deductions and of drivations. Thus, we introduce also the following definitions.

Definition 11.

A *rewriting system* $\mathcal{R}$ is a pair $\langle \mathcal{F}, AX \rangle$ such that AX is as in definition 7.

The *one step derivation* $\Rightarrow$ is defined as follows:

$$\Gamma[\Lambda] \Rightarrow \Gamma[\Delta] \text{ if } \Lambda \rightarrow \Delta \in AX.$$

A *derivation* is the reflexive transitive closure of $\Rightarrow$, denoted $\Rightarrow^*$ and defined as follows:

$$\Gamma_n \Rightarrow^* \Gamma_0 \text{ if and only if either } \Gamma_0 \equiv \Gamma_n \text{ or } \Gamma_n \Rightarrow^* \Gamma_1 \Rightarrow \Gamma_0.$$

$\mathcal{R}$ *generates* the pair $\langle \Gamma, \Delta \rangle$, notation $\vdash_{\mathcal{R}} \Gamma \rightarrow \Delta$ if and only if $\Gamma \Rightarrow^* \Delta$.

An immediate consequence of definition 11 is the following.

Proposition 1. Let $\mathcal{R}$ be a rewriting system. Then

$$\text{if } \Lambda \Rightarrow^* \Delta \text{ and } \Gamma[\Delta] \Rightarrow^* \Sigma, \text{ then } \Gamma[\Lambda] \Rightarrow^* \Sigma$$

Proof.

If $\Lambda \equiv \Delta$, then it holds trivially.

If $\Lambda \Rightarrow \Delta \equiv \Lambda'[\Omega'] \Rightarrow \Delta'[\Omega]$ and $\Omega' \rightarrow \Omega \in AX$, then $\Gamma[\Delta] \Rightarrow^* \Sigma \equiv \Gamma[\Delta'[\Omega]] \Rightarrow^* \Sigma$. Hence $\Gamma[\Delta'[\Omega']] \Rightarrow^* \Sigma \equiv \Gamma[\Lambda] \Rightarrow^* \Sigma$.

If $\Lambda_n \Rightarrow^* \Lambda_1 \Rightarrow \Delta$, and $\Gamma[\Delta] \Rightarrow^* \Sigma$, then $\Gamma[\Lambda_1] \Rightarrow \Gamma[\Delta] \Rightarrow^* \Sigma$. □

We make now clear the link between a rewriting system and a deductive system.

Proposition 2. Let $\mathcal{R} = \langle \mathcal{F}, AX \rangle$ be a rewriting system and $D = \langle \mathcal{F}, AX, \{Cut\} \rangle$ a deductive system and *Cut* is the following rule

$$\frac{\Lambda \rightarrow \Delta \quad \Gamma[\Delta] \rightarrow \Sigma}{\Gamma[\Lambda] \rightarrow \Sigma}$$

Then

$$\vdash_{\mathcal{R}} \Gamma \rightarrow \Delta \quad \text{iff} \quad \vdash_D \Gamma \rightarrow \Delta$$

Proof. Clearly, *Cut* is the same inference rule as the one in proposition 1. □

The rule used in proposition 2, is an unrestricted version of the deductive rule of *cut*. In what follows, we will assume that the succedent of every sequent will be a single formula.

2.3 Context-free grammars

The context-free grammar formalism is important under several respects. In first place, these grammars are simple and easy to use for designing grammars. They are theoretically well understood and have pleasant computational properties. Furthermore, every natural language is to a large extent (though not entirely) context-free ($\mathcal{CF}$), in the sense that it can be analyzed with the formalism of $\mathcal{CF}$ grammars. These aspects made these systems the first candidate for natural language analysis and a standard for evaluating the properties of other frameworks.

Definition 12. A *context-free grammar G* is a triple $\langle V_t, S, D \rangle$ where

- $D = \langle \mathcal{F}, AX, R \rangle$ is a deductive system[2],
- $AX = \{ \Gamma \rightarrow A \mid \Gamma \in (\mathcal{F} \cup V_t)^*,\ A \in \mathcal{F} \}$
- R consists of the *Cut rule*:

$$\frac{\Delta \rightarrow B \quad \Gamma[B] \rightarrow C}{\Gamma[\Delta] \rightarrow C}$$

[2] Equivalently, one may adopt a rewriting system in place of a deductive system.

- S is a distinguished formula, the start symbol,

- V_t is the terminal vocabulary. We assume that $V_t \cap \mathcal{F} = \emptyset$.

In the $\mathcal{CF}$ framework, the $\Gamma \rightarrow A \in AX$ are often called *productions* or also rules. In the following examples of $\mathcal{CF}$ grammars, we write only the set of non-logical axioms AX, as the sets V_t and $\mathcal{F}$ is easily inferable, while the set R and the start symbol S are constant. Moreover, whenever several rewriting options $\Gamma_0 \rightarrow A, \ldots, \Gamma_n \rightarrow A$ appear in AX, we write $\Gamma_0 \mid \ldots \mid \Gamma_n \rightarrow A$.

Example 3. $\mathcal{CF}$ grammars:

- $G_0 = \{\ 0S1 \mid \epsilon \rightarrow S\ \}$.

- $G_1 = \{\ (S)S \mid \epsilon \rightarrow S\ \}$.

- $G_2 = \{\ 1S \mid 0 \rightarrow S\ \}$.

- $G_3 =$

$$\begin{array}{lll} \{ & NP\ VP & \rightarrow & S, \\ & Np \mid Det\ N & \rightarrow & NP, \\ & IV \mid TV\ NP & \rightarrow & VP, \\ & whistles & \rightarrow & IV, \\ & loves & \rightarrow & TV, \\ & Renzo \mid Lucia & \rightarrow & Np, \\ & every \mid a & \rightarrow & Det, \\ & man \mid woman & \rightarrow & N\ \}. \end{array}$$

We write $\vdash_G \Gamma \rightarrow A$, if the grammar G generates the sequent $\Gamma \rightarrow A$.

Definition 13. The *terminal language* generated by a context-free grammar G, denoted $L_t(G)$, is the set of the $\Gamma \in V_t^*$ such that $\vdash_G \Gamma \rightarrow S$.

Definition 14. A grammar G_1 is equivalent to a grammar G_2 if and only if $L_t(G_1) = L_t(G_2)$.

From the derivational perspective, we have the following notions.

Definition 15.

A context-free grammar G based on a rewriting system $\mathcal{R} = \langle \mathcal{F}, AX \rangle$ is a triple $\langle V_t, S, R \rangle$.

The terminal language generated is the set

$$\{\ \Gamma \Rightarrow^* S \mid \Gamma \in V_t^*\ \}$$

Example 4. Terminal languages:

- $L_t(G_0) = \{0^n 1^n \mid 0 \leqslant n\}$.
- $L_t(G_1)$, is the language of strings of balanced brackets.
- $L_t(G_2) = \{1^n 0 \mid 0 \leqslant n\}$.
- $L_t(G_3)$ is the language of well-formed English sentences over the terminal vocabulary of G_3 which contains only the words appearing in the productions.

Example 5. A deduction of the terminal string (())() in G_1.

$$\dfrac{\dfrac{\epsilon \to S \quad \dfrac{\epsilon \to S \quad (S)S \to S}{()S \to S}}{() \to S} \qquad \dfrac{\dfrac{\epsilon \to S \quad \dfrac{\epsilon \to S \quad (S)S \to S}{()S \to S}}{() \to S} \qquad (S)S \to S}{(())S \to S}}{(())() \to S}$$

Observe that several other deductions are available for the same terminal string and all of them are in some sense *equivalent*. This may be seen, for example, from the fact that all these deduction could be mapped to the same *structural description*. A structural description is a tree whose nodes are labeled by the non-terminals of the grammar and whose leaves by terminals. We refer to section 1.2 of chapter 1 for the information encoded by structural descriptions and define here a recursive procedure for mapping a deduction into a structural description.

Definition 16. Let a $\mathcal{CF}$ grammar $G = \langle V_t, S, \langle \mathcal{F}, AX, Cut \rangle\rangle$ be given. Let $\vdash_G \Gamma \to C$. We build a structural description T for $\Gamma \to C$ in G as follows. Assume that the γ's and δ's are elements of $V_t \cup \mathcal{F}$.

If $\Gamma \to C \equiv \delta_1 \dots \delta_m \to C \in AX$, then T is the tree

$$\dfrac{\delta_1 \quad \dots \quad \delta_m}{C}$$

Otherwise, $\Gamma \rightarrow C \equiv \gamma_1 \dots \gamma_i \delta_1 \dots \delta_m \gamma_{i+1} \dots \gamma_n \rightarrow C$ and the last step of a deduction of $\Gamma \rightarrow C$ in G is

$$\frac{\delta_1 \dots \delta_m \rightarrow B \quad \gamma_1 \dots \gamma_i B \gamma_{i+1} \dots \gamma_n \rightarrow C}{\gamma_1 \dots \gamma_i \delta_1 \dots \delta_m \gamma_{i+1} \dots \gamma_n \rightarrow C}$$

Assume that the tree

$$\frac{\overline{\delta_1} \quad \dots \quad \overline{\delta_m}}{B}$$

is assigned to the deduction of $\delta_1 \dots \delta_m \rightarrow B$ and that the tree

$$\frac{\overline{\gamma_1} \quad \dots \quad \overline{\gamma_i} \quad B \quad \overline{\gamma_{i+1}} \quad \dots \quad \overline{\gamma_n}}{C}$$

is assigned to the deduction of $\gamma_1 \dots \gamma_i B \gamma_{i+1} \dots \gamma_n \rightarrow C$. Then, T is the tree

$$\frac{\overline{\gamma_1} \quad \dots \quad \overline{\gamma_i} \quad \dfrac{\overline{\delta_1} \quad \dots \quad \overline{\delta_m}}{B} \quad \overline{\gamma_{i+1}} \quad \dots \quad \overline{\gamma_n}}{C}$$

The notion of deduction (as well as that of derivation) is affected by *spurious ambiguity*. In the context of CF grammars, this means that several deductions may correspond to the same structural description. In a computation, we want to avoid such proliferation of equivalent deductions. However, there may also be cases of genuine ambiguity, that is, different deductions of the same sequent that correspond to different structural descriptions, and we want to maintain these, for example because they express different semantic interpretations of the root sequent. In the next chapter, we will see that there are elegant and powerful methods to solve these problems for CF grammars. We will also see that the notion of spurious ambiguity has a more subtle character in the case of Lambek style categorial grammars and that its elimination requires more ingenuity.

A class of context-free grammars has particularly nice computational properties.

Definition 17. A grammar G is in Chomsky normal form (CNF), if it contains only productions of the form:

$A\,B \rightarrow C, A \neq S, B \neq S,$

$w \rightarrow A, w \in V_t$ or

$\epsilon \rightarrow S.$

[Chomsky, 1959] proves that for every context-free grammars G there is a CNF grammar G' such that $L_t(G) = L_t(G')$. Observe that, for grammars in CNF, it is possible to partition the set AX into two distinct sets: one contains only *lexical axioms*, namely productions of the form $w \to A$, the other only *non-lexical axioms*, namely productions of the form $A\,B \to C$. All productions of CNF grammars are binary, and we will see that this allows to parse them with a very simple and elegant parsing algorithm known as the CYK algorithm. The fact that all CF grammars can be put in CNF also makes the CYK algorithm a general parsing algorithm for CF languages. However, it should be remarked that the CNF variant of a CF grammar may generate a different *structural* language from the one of the original grammar.

2.4 Categorial grammars

The first form of categorial notation was introduced by the Polish logician Kazimier Ajdukiewicz in [Ajdukiewicz, 1935]. A category was presented as an object of the following form,

$$\frac{a}{b}$$

where a and b are also categories. The intuition behind such a notation is that of a *function* from an object of type b, the input (or argument or denominator), to an object of type a, the output (or value or numerator). In other words, an expression of category $\frac{a}{b}$ is an *incomplete* expression, looking for an expression of category b to give an expression of category a.

Later, Ajdukiewicz's notation for categories was refined by Joshua Bar-Hillel, who distinguished categories of the form a/b and categories of the form $b\backslash a$ in [Bar-Hillel, 1953]. The meaning of such a notation was the following.

- An expression of category a/b combines with an expression of category b to its right to give an expression of category a.

- An expression of category $b\backslash a$ combines with an expression of category b to its left to give an expression of category a.

The calculus resulting by adopting these rules, which we call *cancellation schemes*, is nowadays called AB calculus, or also basic categorial logic. More formally, we extend the formula type constructor as follows.

Definition 18. *Formulas*, or *categories*, are defined from the set of atoms A as

$$\mathcal{F} := A \mid \mathcal{F}/\mathcal{F} \mid \mathcal{F}\backslash\mathcal{F} \mid \mathcal{F} \otimes \mathcal{F}$$

In the context of categorial grammar, *CG* hereafter, formulas as also called categories. Formulas of $C\!F$ grammars are also formulas of categorial grammar. However, we will distinguish the two systems by writing atoms of *CG* with lowercase letters, while the atoms of $C\!F$ grammar will always start with capital letters. In showing complex formulas, we omit the most external brackets. Furthermore, we assume the slashes have higher precedence over the product and juxtaposition. Thus for example we write $a{\otimes}b\backslash c$ for $a \otimes (b\backslash c)$ and $a\ b\backslash c$ for $a\ (b\backslash c)$. Finally, we assume that / is left associative and \ is right associative. So $(a/b)/c$ is written as $a/b/c$, and $c\backslash(b\backslash a)$ as $c\backslash b\backslash a$ (we will sometimes drop this convention on the basis of readability).

Product formulas, that is formulas of the form $a \otimes b$, appear for the first time in [Lambek, 1958]. Still nowadays, many categorial linguists work within *product-free* systems. In some case, this may be legitimate. However, if we aim at the structural adequacy of the syntactic description, product categories are a valuable tool. We will discuss categorial systems with and without product, although one of our contributions is the application of parsing systems as the CYK or the Earley parser to categorial grammars *with product*.

Categorial grammars consist of a lexicon and of a deductive system. The lexicon assigns words in the terminal vocabulary to syntactic categories and the deductive system specifies the way complex expressions are derived, according to the inference rules.

Definition 19. A *categorial grammar* based on a deductive system D is a quadruple $\langle V_t, s, Lex, \langle \mathcal{F}, AX, R\rangle\rangle$ where

- V_t is the terminal vocabulary of the grammar.
- s is the distinguished start symbol
- *Lex*, the *lexicon*, is a set of pairs $\langle w, c\rangle$, *the lexical assignments*, with $w \in V_t$ and $c \in \mathcal{F}$ which we write $w \to c$.

Thus, we can specify different kinds of categorial grammars by just specifying a deductive system D. For instance, the Ajdukiewicz Bar-Hillel calculus is defined as follows.

Definition 20. The *AB* deductive system is a triple $\langle \mathcal{F}, AX, R\rangle$ such that

- $AX = \{\, a \rightarrow a \mid a \in \mathcal{F} \,\}$,
- R consists of the following rules which we call *basic cancellation rules*.

$$\frac{\Gamma \rightarrow a/b \quad \Delta \rightarrow b}{\Gamma\,\Delta \rightarrow a} \qquad\qquad \frac{\Gamma \rightarrow b \quad \Delta \rightarrow b\backslash a}{\Gamma\,\Delta \rightarrow a}$$

We call *AB* grammar a categorial grammar based on the deductive system *AB*. Consider, for example, the following *AB* grammar.

Example 6.

$A_0 = \langle \{a, b\},\ s,\ \{a \rightarrow s/c/s,\ \epsilon \rightarrow s,\ b \rightarrow c\},\ AB \rangle$

We can easily see that grammar A_0 generates the language $a^n b^n$. However, we still have no link between the terminal language generated by the grammar and the language generated by the categorial deductive system. The following definition provides us with this link, observe that the lexicon may contain assignments for the empty string, this is why the input string may be shorter than the list of categories in the root sequent.

Definition 21. A categorial grammar $G = \langle V_t, s, Lex, D \rangle$ *generates* a string $w_0 \ldots w_m \in V_t^*$ if and only if $\vdash_D a_0 \ldots a_n \rightarrow s$, $0 \leqslant m \leqslant n$ and $w_0 \ldots w_m \Rightarrow^* a_0 \ldots a_n$ on the basis of the axioms in *Lex*.

Example 7.

Grammar A_0 generates *aabb*, since *AB* generates $s/c/s,\ s/c/s,\ s,\ c,\ c \ \rightarrow\ s$ according to the following deduction

$$\dfrac{\dfrac{s/c/s \rightarrow s/c/s \qquad \dfrac{\dfrac{\dfrac{s/c/s \rightarrow s/c/s \quad s \rightarrow s}{s/c/s,\ s \rightarrow s/c} \qquad c \rightarrow c}{s/c/s,\ s,\ c \rightarrow s}}{}}{s/c/s,\ s/c/s,\ s,\ c \rightarrow s/c} \qquad c \rightarrow c}{s/c/s,\ s/c/s,\ s,\ c,\ c \rightarrow s}$$

and

$$aabb \quad \Rightarrow^* \quad s/c/s,\ s/c/s,\ s,\ c,\ c$$

on the basis of the axioms in *Lex*.

In the original formulations of *AB* grammars, a second kind of cancellation rule was present together with the basic cancellation schemes, see also [Lambek, 1958]. This second kind of rules can be seen as a generalization of the basic cancellation rules. On the other hand, we preferred to keep distinct the two systems and discuss them separately.

Definition 22. We call AAB (associative AB calculus) the deductive system $\langle \mathcal{F}, AX, R\rangle$ where $\mathcal{F}$ and AX are as in definition 20, and R consists of the basic cancellation rules of the AB calculus and of the following inference rules, which we call *associative cancellation rules*[3].

$$\frac{\Gamma \to a/b \quad \Delta \to b/c}{\Gamma\,\Delta \to a/c} \qquad\qquad \frac{\Gamma \to c\backslash b \quad \Delta \to b\backslash a}{\Gamma\,\Delta \to c\backslash a}$$

An AAB grammar is a categorial grammar that has AAB as deductive engine. Consider the following associative Ajdukiewicz Bar-Hillel categorial grammar.

Example 8.

Let A_1 be an AAB grammar, with the following lexicon:

$$\begin{array}{llll} \{ & \mathit{John} & \to & n, \\ & \mathit{Mary} & \to & n, \\ & \mathit{someone} & \to & s/(n\backslash s), \\ & \mathit{everyone} & \to & (s/n)\backslash s, \\ & \mathit{everyone} & \to & ((n\backslash s)/n)\backslash(n\backslash s), \\ & \mathit{loves} & \to & (n\backslash s)/n, \\ & \mathit{ismissing} & \to & n\backslash s, \\ & \mathit{ismissing} & \to & (s/(n\backslash s))\backslash s\ \} \end{array}$$

As formulas may become soon very long, we may introduce the following macros.

Definition 23. *Macros.*

$$iv := n\backslash s$$
$$tv := iv/n$$

Example 9. Deductions of *Someone loves everyone* in A_1.

1.
$$\dfrac{\dfrac{s/iv \to s/iv \quad tv \to iv/n}{s/iv,\ tv \to s/n} \qquad (s/n)\backslash s \to (s/n)\backslash s}{s/iv,\ tv,\ (s/n)\backslash s \to s}$$

[3] In fact, one can generalize the cancellation schemes in the following way, with $1 \leqslant j$ (if $j \equiv 1$, then $a/_1 \dots /_j b \equiv a/b$) and $0 \leqslant k$ (if $k \equiv 0$, then $b/_0 \dots /_k c \equiv b$):

$$\frac{\Gamma \to a/_1 \dots /_j b \quad \Delta \to b/_0 \dots /_k c}{\Gamma\,\Delta \to a/_1 \dots /_{j+k-1} c}$$

2.

$$\dfrac{s/iv \to s/iv \qquad \dfrac{tv \to tv \qquad tv\backslash iv \to tv\backslash iv}{tv,\ tv\backslash iv \to iv}}{s/iv,\ tv,\ tv\backslash iv \to s}$$

Deduction 1 relies essentially on the rules of the system *AAB*. We can observe that these two deductions are not a case of spurious ambiguity. Indeed they are different (one could easily map them to two different structural descriptions) and there are several reasons to be interested in both of them. In the next section, we will introduce one of the most interesting aspects of categorial grammars: the correspondence between syntax and semantics. We will see that the two deductions in example 9 express different scope relations between the subject and the object noun phrase[4].

2.5 Lambda terms

The lambda calculus is the term language used in model theoretic semantics to build semantic representations of linguistic expressions in accordance with the compositionality principle. For our proposes, we can define the lambda term language as follows.

Definition 24.

$$\begin{array}{lcl}
\text{Var} & := & x_1 \mid x_2 \mid \ldots \\
\text{Con} & := & c_1 \mid c_2 \mid \ldots \mid c_n \\
\text{Lam} & := & \text{Var} \mid \text{Con} \mid \lambda\text{Var. Lam} \mid (\text{Lam Lam}) \\
 & & \mid \langle \text{Lam},\ \text{Lam} \rangle \mid \pi(\text{Lam}) \mid \pi'(\text{Lam})
\end{array}$$

We show variables in Var as x, y or z, in which case different letters denote different variables. A term $\lambda x.t$ is called an *abstract*, $(t_1\ t_2)$ is called *application*, $\langle t_1, t_2 \rangle$ *pair* and $\pi(t)$ and $\pi'(t)$ are the first and second *projections*, respectively. In order to simplify the notation, we assume that application is left associative, thus we write the term $((t_1\ t_2)\ t_3)$ as $(t_1\ t_2\ t_3)$. Furthermore, when it creates no ambiguity, we write $\pi(x)$ as πx.

[4] At the syntactic and prosodic levels, instead, the two deductions in example 9 encode different structural descriptions that can be assigned to the input string. The connection between categorial deductions and prosodic phrasing, expressed in terms of the branching of structural descriptions, is being intensively studied in recent years. The works on *combinatory categorial grammar* of M. Steedman and his school, see for example [Steedman, 2000b,a], emphasize the connection between the structures arising from (combinatory) categorial deductions and prosodic phrasing.

Definition 25.

The *scope* of λx in $\lambda x.t$ is t.

A variable x is said to be *free* in a lambda term t, if it is not in the scope of a λx. Otherwise is said to be *bound*.

As stated in [Blackburn and Bos, 2003], "the lambda calculus is a tool for controlling the process of making substitutions". The definition of *substitution* given below is partially taken from [Hindley, 1997] and extended to the case of constant, pair and projection terms. We denote $FV(t)$ the set of free variables of the term t.

Definition 26. *Substitution*: $t[x := t']$ is the result of substituting t' for x in t. Formally,

$$\begin{array}{lcll}
x[x := v] & = & v & \\
x[y := v] & = & x & \\
c[y := v] & = & c & \text{if } c \in \mathrm{Con} \\
\langle t_1, t_2\rangle[y := v] & = & \langle t_1,[y := v],\ t_2[y := v]\rangle & \\
\pi(t)[y := v] & = & \pi(t[y := v]) & \\
\pi'(t)[y := v] & = & \pi'(t[y := v]) & \\
(t_1\ t_2)[y := v] & = & (t_1[y := v]\ t_2[y := v]) & \\
(\lambda\, x.t)[x := v] & = & \lambda\, x.t & \\
(\lambda\, x.t)[y := v] & = & \lambda\, x.t & \text{if } y \notin FV(t) \\
(\lambda\, x.t)[y := v] & = & \lambda\, x.(t[y := v]) & \text{if } y \in FV(t) \text{ and } x \notin FV(v) \\
(\lambda\, x.t)[y := v] & = & \lambda\, z.(t[x := z][y := v]) & \text{if } y \in FV(t) \text{ and } x \in FV(v)
\end{array}$$

Observe that we are adopting the untyped variant of the lambda calculus. This means that expressions as $(x\ x)$ are well formed. We will dedicate the next section to *typed* lambda calculus and discuss its connection with categorial grammar.

The following equalities define the basic steps of term reduction. In fact, these equalities should be seen as rewriting rules. In each case, the term on the left of the equality symbol, called *redex*, is rewritten to the equivalent, though shorter, term on the right, called *contractum*. We put on the left column β-contraction and on the right η-contraction.

Definition 27.

Contraction: term equations for lambda terms

β-contraction	η-contraction
$\pi\langle t_1, t_2\rangle = t_1$ $\pi'\langle t_1, t_2\rangle = t_2$	$\langle \pi t, \pi' t\rangle = t$
$((\lambda x.t)\ v) = t[x := v]$	$\lambda x.(t\ x) = t$, if $x \notin FV(t)$

A *reduction* is a series of contractions.

A term is *normal*, if no contraction can be applied to it.

2.5.1 Typed lambda calculus

In the type free lambda calculus, each functional term is from lambda terms to lambda terms, without restrictions. The *typed* lambda calculus is a proper *subset* of the full lambda calculus (for example, self-application terms, like $(x\ x)$, are not in the typed calculus). In this system, each lambda term has a single type associated with it. Thus, types can be seen as equivalence classes of lambda terms. If a typed term t is functional, its type determines the domain of its possible argument terms as well as the type of the result of applying t to its argument. Observe the similarity with the categorial formalism: an expression of functional category, say $\frac{a}{b}$ combines with an expression of category b to give a compound expression of category a. We will see that there is indeed a close correspondence between formulas of categorial grammar, types and typed terms.

For our proposes, we can define the type language as follows.

$$Type := e \mid t \mid Type \rightarrow Type \mid Type \times Type$$

The types e and t are the primitive types of *individuals* and *truth values*, respectively. Any other type is either a function from a type to a type or a product of types. Intuitively, we can think of the types e as the *set* of individuals and t as the set of truth values, $\{True, False\}$[5]. A type $p \rightarrow q$ is a function from p to q, while a type $p \times q$ is the Cartesian products of p and q.

We define the mapping from syntactic categories to types.

Definition 28. Let ty be a function from syntactic categories to types, $ty :: \mathcal{F} \rightarrow Type$. Let $ty(x)$ be given for all the $x \in A$, for instance $ty(n) = e$ and

[5] In Montague grammar, one finds also the primitive type s for possible worlds, which is beyond the scope of this book.

$ty(s) = t$. Compound formulas are mapped to compound types as follows:

$$\begin{array}{lcl} ty(a/b) & = & ty(b) \rightarrow ty(a) \\ ty(b\backslash a) & = & ty(b) \rightarrow ty(a) \\ ty(b \otimes a) & = & ty(b) \times ty(a) \end{array}$$

In [Moortgat, 1997b], one can find a so called Church style definition of typed lambda terms. Instead, we adopt an approach which is more in the style of [Curry and Feys, 1958], see also [Hindley, 1997]. This means that while our definition of the lambda term language is type-free, as it admits terms as $(x\ x)$, we define *typed terms* as those lambda terms to which a type can be assigned.

The following typing rules are deduction rules working on *type assignments*: objects of the form $t\!:\!p$, where t is a term and p is a type, meaning 't is a term of type p'. Typing rules are expressed in the form of *typing judgements* of the form

$$\{x_1\!:\!p_1, \ldots x_n\!:\!p_n\} \vdash t\!:\!p$$

with the meaning that 'if $x_i : p_i,\ 0 \leqslant i \leqslant n$, then t is a well formed term of type p'. In such judgemets, all free variables of t must be contained in $\{x_1, \ldots, x_n\}$. We use $\Gamma, \Gamma_i,\ 0 \leqslant i$ as variables over sets of type assigments.

Definition 29. Typing rules for typed lambda calculus:

$$x\!:\!p \vdash x\!:\!p$$

$$\frac{\Gamma_1 \vdash t\!:\!q \rightarrow p \quad \Gamma_2 \vdash t'\!:\!q}{\Gamma_1 \cup \Gamma_2 \vdash (t\ t')\!:\!p} \qquad \frac{\Gamma \cup \{x\!:\!q\} \vdash t\!:\!p}{\Gamma \vdash \lambda x.t\!:\!q \rightarrow p}$$

$$\frac{\Gamma \vdash t\!:\!p \times q}{\Gamma \vdash \pi t\!:\!p} \qquad \frac{\Gamma \vdash t\!:\!p \times q}{\Gamma \vdash \pi' t\!:\!q}$$

$$\frac{\Gamma_1 \vdash t\!:\!p \quad \Gamma_2 \vdash t'\!:\!q}{\Gamma_1 \cup \Gamma_2 \vdash \langle t, t' \rangle\!:\!p \times q}$$

To be precise, the typing rules operate on *type variables* $p, q \ldots$. Thus the result of a type deduction is a *type schema* for the input lambda term. In the following example, we directly unify the typing of the term with the type resulting for the syntactic category. A more precise description of the typing algorithm for lambda calculus, including polymorphism and unification, can be found in [Hankin, 2004] which in turn is based on [Damas and Milner, 1982].

Example 10. The word *himself* is assigned the category $((n\backslash s)/n)\backslash n\backslash s$ and the term $\lambda x\lambda y.(x\ y\ y)$. We have:
$ty(((n\backslash s)/n)\backslash n\backslash s) = (e \to (e \to t)) \to (e \to t)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\{x:e \to (e \to t)\} \vdash x:e \to (e \to t) \quad \{y:e\} \vdash y:e}{\{x:e \to (e \to t), y:e\} \vdash (x\ y):e \to t} \qquad \{y:e\} \vdash y:e}{\{x:e \to (e \to t), y:e\} \vdash (x\ y\ y):t}}{\{x:e \to (e \to t)\} \vdash \lambda y.(x\ y\ y):e \to t}}{\emptyset \vdash \lambda x\lambda y.(x\ y\ y):(e \to (e \to t)) \to (e \to t)}$$

By *extended deductive systems* we mean a deductive system extended with proof-encoding term of some sort. If we are interested in compositional semantics, the term language will be the lambda term language. However, depending on the proposes, different term languages may result to be useful. For example, [Lambek, 1993] uses a term language isomorphic to deductions in the non-associtive Lambek calculus and defines proof normalization via term equations, on the basis of the cut elimination algorithm. Presently, we will largely use the lambda calculus as a term language also because the notation for syntactic terms in the style of Lambek, see also [Lambek, 1988, Lambek and Scott, 1987] and [Moortgat and Oehrle, 1997], becomes soon quite heavy to read and is not relevant to our proposes.

Each sequent of an extended deductive system is paired with a term. We call *arrow* an object of the form $f : \Gamma \to c$, where f is a term and $\Gamma \to c$ is a sequent.

Definition 30. An *extended deductive system* is a quadruple $\langle \mathcal{F}, \mathcal{T}, AX, R\rangle$ such that

- $\mathcal{F}$ is a set of formulas.
- $\mathcal{T}$ is a set of terms of a given term language (for example the lambda calculus).
- $AX = \{\ t : \Gamma \to \Delta \mid t \in \mathcal{T},\ \Gamma \in \mathcal{F}^*,\ \Delta \in \mathcal{F}^+\ \}$.
- R is a set of inference rules of the form

$$\dfrac{f_1 : \Gamma_1 \to \Delta_1 \quad \dots \quad f_n : \Gamma_n \to \Delta_n}{\rho(f_1 \dots f_n) : \Gamma \to \Delta}$$

  where ρ is an operation on terms such that $\rho(f_1 \dots f_n) \in \mathcal{T}$.

We define now the semantic variant of the *AAB* calculus.

In categorial logic, one works with *linear* lambda terms which are a subset of typed lambda terms obeying further constraints. As our lambda calculus uses projections, we shall slightly modify the standard definition of linear lambda terms. For $x \in \mathrm{Var}$, let us call a *projection* of x, denoted $\pi_n x$, a term of the form $\pi_n \ldots \pi_1 x$, where each π_i, $1 \leqslant i \leqslant n$ is either a first or a second projection. We assume that $\pi_0 x \equiv x$ and we say that a projection $\pi_i x$ is free in t, if it is not in the scope of λx.

Definition 31. A *linear* lambda term t is a lambda term such that

- for each subterm $\lambda x.t'$ of t, each projection of x occurs free in t' exactly once,

- no variable variable occurs free in t.

In fact, these constraints apply to what, in the next section, we will call *derivational* semantics. Instead, at the *lexical* level one is, to some extent, free to violate them. An example of the lexical violation of the constraints in definition 31 is the assignment of the reflexive pronoun, *himself*, whose category is $((n\backslash s)/n)\backslash(n\backslash s)$ and whose term assignment is $\lambda x \lambda y.(x\ y\ y)$. In section 2.7.1, we will see other examples of lexical assignments that violate the constraints on linear lambda terms. However, we will always keep distinct the derivational and the lexical components of semantic interpretation.

The functorial structure of lambda calculus and categorial logic allows to interpret each inference rule as an operation on the lambda term. We present a new formulation of the *AAB* calculus. Here we use uppercase greek letters for *bracketed strings* of categories. The role of brackets will be clarified in definition 43. Let us extend the function ty as to apply to such bracketed strings of categories by adding the clause

$$ty((\Gamma, \Delta)) = ty(\Gamma) \times ty(\Delta)$$

An arrow with semantic label is always an object of the form

$$\lambda x.v : \Gamma \to c$$

In such an arrow, the type of the term is (unifiable with) the type of the sequent: both are functions from a Γ-type object to a c-type object.

Definition 32. Semantic *AAB* calculus:

$$\lambda\, x.x \colon a \to a$$

$$\frac{v \colon \Gamma \to a/b \quad u \colon \Delta \to b}{\lambda x.(v\ \pi x\ (u\ \pi' x)) \colon (\Gamma,\ \Delta) \to a} \qquad \frac{v \colon \Gamma \to a/b \quad u \colon \Delta \to b/c}{\lambda x \lambda y.(v\ \pi x\ (u\ \pi' x\ y)) \colon (\Gamma,\ \Delta) \to a/c}$$

$$\frac{u \colon \Gamma \to b \quad v \colon \Delta \to b\backslash a}{\lambda x.(v\ \pi' x\ (u\ \pi x)) \colon (\Gamma,\ \Delta) \to a} \qquad \frac{u \colon \Gamma \to c\backslash b \quad v \colon \Delta \to b\backslash a}{\lambda x \lambda y.(v\ \pi' x\ (u\ \pi x\ y)) \colon (\Gamma,\ \Delta) \to c\backslash a}$$

We formulate labeled categorial grammar.

Definition 33. A labeled categorial grammar is a categorial grammar based on a labeled deductive system D and whose lexicon *Lex* contains triples $\langle t, w, c\rangle$, which we show as $t : w \to c$, where $w \in V_t$, $c \in \mathcal{F}$ and $t \in \mathcal{T}$, the term language of D.

The following is the variant of lexicon A_1 resulting by adding semantic terms. Well-typing requires the type of the lambda term to be the same as that of the category to which it is associated. Thus, j and m are constants of type e, $\textsc{loves}$ is of type $e \to (e \to t)$, $\textsc{ismissing}$ is of type $e \to t$ and $\textsc{ismissing}^*$ is of type $((e \to t) \to t) \to t$. $\exists$ and $\forall$ are constants of type $(e \to t) \to t$. An expression $\exists x\, v$ is shorthand for $(\exists\ \lambda x.\, v)$, which is the term resulting from application of the constant $\exists$ to the term $\lambda x.\, v$ of type $e \to t$[6]. The same holds for $\forall x\, v$. The type of the other terms can be easily recovered from the category. We will see immediately that the well typing constraints are enforced by the categories in the derivation. Hence, we do not need to explicitly distinguish variables of different types.

Example 11.

$A_1 =$

$$\begin{array}{rlll}
\{\ j : & \textit{John} & \to & n, \\
m : & \textit{Mary} & \to & n, \\
\lambda x.\exists y\,(x\ y) : & \textit{someone} & \to & s/(n\backslash s), \\
\lambda x.\forall y\,(x\ y) : & \textit{everyone} & \to & (s/n)\backslash s, \\
\lambda x \lambda y.\forall z\,((x\,z)\ y) : & \textit{everyone} & \to & ((n\backslash s)/n)\backslash(n\backslash s),
\end{array}$$

[6] Observe that $\lambda x.\exists y\,(x\ y)$ assigned to *someone* in lexicon A_1 η-reduces to $\exists$. Indeed, we could use this reduced form in the lexicon. However, we preferred to keep the more traditional notation for quantifiers.

$$\begin{aligned}\textsc{loves}: &\ \mathit{loves} && \to (n\backslash s)/n,\\ \textsc{ismissing}: &\ \mathit{ismissing} && \to n\backslash s,\\ \textsc{ismissing}^*: &\ \mathit{ismissing} && \to (s/(n\backslash s))\backslash s\ \}\end{aligned}$$

The notion of generation is extended to labeled systems in order to assign a term to the generated string.

Definition 34. A categorial grammar with semantic labels $G = \langle V_t, s, Lex, D\rangle$ generates a string $w_0 \ldots w_m \in V_t^*$ and assigns it the semantic term t if and only if D generates $f : \Gamma \to s$ and $t = (f\ t')$, where

1. Γ is a structure living on $a_0 \ldots a_n$, $m \leqslant n$,
2. $w_0 \ldots w_m \Rightarrow^* a_0 \ldots a_n$ by means of axioms

$$\begin{aligned} t_i : w_i \to a_i \in Lex, &\quad 0 \leqslant i \leqslant m\\ t_i : \epsilon \to a_i \in Lex, &\quad 0 \leqslant i \leqslant n\end{aligned}$$

3. t' is obtained by replacing in Γ each a_i with t_i, each (with $\langle$ and each) with $\rangle$.

This definition deserves some comments. For a structure Γ to *live on* a list of formulas $a_0 \ldots a_n$, means that $a_0 \ldots a_n$ is the result of eliminating all brackets from Γ. In 2, we check whether we can rewrite $w_0 \ldots w_m$ to $a_0 \ldots a_n$ by means of transitions in the lexicon. The structural information encoded by Γ is exploited in 3 to build a pair lambda term t', which is itself a structure living on the terms $t_0 \ldots t_n$ assigned, respectively, to $a_0 \ldots a_n$ in the lexicon. The term t' encodes the *lexical semantics* of $w_0 \ldots w_m$ according to G. The *derivational semantics* instead is encoded by the term f resulting from the derivation. The semantic representation t of $w_0 \ldots w_m$ in G is given by the application of the derivational term f the the lexical term t', that is $t = (f\ t')$.

The deductions given in example 9 are now proposed once more with lambda term decorations. For reasons of space we extend the macros in definition 35 with the following new macros.

Definition 35. *Macros.*

$$\begin{aligned} qs &:= s/iv\\ qo &:= (s/n)\backslash s\end{aligned}$$

In the deductions below, we immediately reduce terms to normal form. The notation $t \rightsquigarrow t'$ indicates that the term t' is obtained from the term t through a number of steps of contraction.

Example 12. Semantic deductions:

1. Derivational semantics of deduction 1 of example 9.

$$\dfrac{\dfrac{\lambda x.x\colon qs \to s/iv \quad \lambda y.y\colon tv \to iv/n}{\lambda x\lambda y.(\pi x\,(\pi' x\ y))\colon qs,\ tv \to s/n} \qquad \lambda z.z\colon qo \to qo}{\lambda x.(\pi' x\ \lambda k.(\pi\pi x\,(\pi'\pi x\ k)))\colon (qs,\ tv),\ qo \to s}$$

   Lexical semantics: from the bracketed string of categories $(qs,\ tv)$, qo and A_1, we obtain the term $\langle\langle\lambda\, x.\exists y\,(x\ y), \textsc{loves}\rangle, \lambda\, x.\forall y\,(x\ y)\rangle$.
   Semantic representation of *someone loves everyone* according to the deduction 1:

$$\begin{array}{l}
(\lambda x.(\pi' x\ \lambda k.(\pi\pi x\,(\pi'\pi x\ k)))\ \langle\langle\lambda\, x.\exists y\,(x\ y), \textsc{loves}\rangle, \lambda\, x.\forall y\,(x\ y)\rangle) \rightsquigarrow \\
(\lambda\, x.\forall y\,(x\ y)\ \lambda k.(\lambda\, x.\exists y\,(x\ y)\ (\textsc{loves}\ k))) \rightsquigarrow \\
(\lambda\, x.\forall y\,(x\ y)\ \lambda k.\exists y\,(\textsc{loves}\ k\ y)) \rightsquigarrow \\
\forall z\,(\lambda k.\exists y\,(\textsc{loves}\ k\ y)\ z) \rightsquigarrow \\
\forall z\,\exists y\,(\textsc{loves}\ z\ y)
\end{array}$$

2. Deduction 2 of example 9, instead, assigns the lambda term $\exists y\,\forall z\,(\textsc{loves}\ z\ y)$ to the input string, as it can be easily checked.

2.5.2 Lambda calculus in Haskell

The Haskell programming language is a *functional* language. In such a programming language, a program is expressed as a function and the execution of the program amounts to the evaluation of a function. The syntax and semantics of Haskell are largely based on lambda calculus and its notion of reduction, respectively. The Haskell notation for lambda terms is closely reminiscent of the lambda notation. For instance, $\backslash x \to v$ in Haskell corresponds to what we wrote in the previous sections as $\lambda x.v$.

In implementing the semantic component of a categorial grammar, it would be convenient to rely on the lambda term notation and reduction which is hardwired in the Haskell syntax and semantics, instead of developing our own object lambda term syntax and then a term reduction routine, as it is done in [van Eijck, 2003]. For example, this would simplify notably the mechanism of assigning fresh variables in the construction of the derivation. And of course, a reduction routine defined in Haskell cannot be faster than the reduction mechanism proper of the Haskell compiler which interprets such routine.

On the other hand, Haskell lambda terms, as *typed terms*, inherit their type from the type of their arguments and of their value and well typing constraints prevent an object of type b from occuring where an object of type a should occur. Thus, different Haskell lambda terms may belong to different types. This raises some difficulties if we want to implement an arrow $t\colon w \to c$, where t is lambda term, by writing t as the corresponding *Haskell* lambda term.

We will proceed in a way similar to the one proposed in [Wadler, 1992], although we will not enter the details on monads. In that paper, a lambda term language is defined, largely long the same lines in which we defined the language Lam. An interpretation function maps terms in Lam into the evaluation. The evaluation, `eval` :: Lam $\to$ *Lam*, in turn relies on the reduction mechanism of the Haskell interpreter. Thus $\mathtt{eval}(\lambda x.v)$ is interpreted as $Abs(\backslash x \to (\mathtt{eval} v))$[7]. Observe that while $\lambda x.v$ is an object lambda term of type Lam, $Abs(\backslash x \to v)$ is a value in the interpretation language which includes a subterm $\backslash x \to v$ built by means of Haskell abstraction. Consider the following Haskell definition of object lambda terms.

$$\begin{aligned} Lam \quad &:= \quad X\ Int \mid C\ String \\ &\quad\ \ \mid Abs\ (Lam \to Lam) \mid App\ Lam\ Lam \\ &\quad\ \ \mid Pair\ Lam\ Lam \mid Pi\ Lam \mid Pi'\ Lam \end{aligned}$$

The most remarkable difference with respect to the definition of Lam, in the previous section, is that in the interpretation we state that an object of type *Lam* can be of the form *Abs* (*Lam* $\to$ *Lam*), that is, it may consist of a constructor *Abs*(traction) followed by a function from a *Lam* type object to a *Lam* type object. This operation transforms a function $f :: Lam \to Lam$ into an object $Abs\ f :: Lam$.

Working on the *interpreted* language *Lam*, we can define the contractions relying on Haskell reduction system.

Definition 36.

$(\wr) :: Lam \to Lam \to Lam$

$$\begin{aligned} (Abs\ f) \wr g &= (f\ g) \\ f \wr g &= f\ \grave{}App\acute{}\ g \end{aligned}$$

$fst, snd :: Lam \to Lam$

$$\begin{aligned} fst\ (Pair\ f\ \_) &= f \\ fst\ f &= Pi\ f \\ snd\ (Pair\ \_\ f) &= f \\ snd\ f &= Pi'\ f \end{aligned}$$

[7] We are omitting the environment variable here, for simplicity.

The operator $(\wr)$ is interprets application. If its left argument is an abstract of the form $Abs\ f$, then we know that $f :: Lam \rightarrow Lam$. As the right argument g is of type Lam, $(Abs\ f) \wr g$ returns $(f\ g)$, that is the *result* of the application of f to g. Instead, if the left argument is not an abstract, it returns the application of the constructor *App*, object level application, to the arguments. The functions fst and *snd*, for first and second projection, have a similar behavior. Thus the constructors *App*, *Pi* and *Pi'* 'fake' their lambda calculus counterparts exactly in those cases in which reduction is not possible.

One may wish to define also the composition of lambda, which interprets the cut rule, as follows

Definition 37.

$(\circ) :: Lam \rightarrow Lam \rightarrow Lam$

$$(Abs\ f) \circ (Abs\ g) = Abs\ f.g$$

Here, $(.) :: (a \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow c \rightarrow b$ is Haskell function composition, defined as $g.f = \backslash x \rightarrow (g\ (f\ x))$.

Another suitable operation is *type raising*, see for example [Hendriks, 1993].

Definition 38.

$lift :: Lam \rightarrow Lam$

$$lift\ x = Abs\ (\backslash y \rightarrow y \wr x)$$

With this notation, the lexical assignment for *someone* $\rightarrow s/(n\backslash s)$ becomes

$$Abs\ (\backslash y \rightarrow \exists \wr (Abs\ (\backslash x \rightarrow y \wr x)))$$

Observe that no object variable is used in such a lexical assignment. Indeed, y and y are Haskell variables. The object variables $X\ i$ play a role only in two cases. The first case has to do with *lexical assignments* containing free variables, as pronouns of category $s/(n\backslash s)$, like *she*, which would be assigned the term $lift\ (X\ i)$ for some integer i. The second case has to do simply with *showing* a term $Abs\ f$. If we show a variable $X\ i$ as x_i, we can extend the show function as to apply to a term $Abs\ f$ in such a way that *show* $(Abs\ f)$ is $\lambda x_i.\ show\ (f\ (X\ i))$ for a suitable index i. Thus $show\ (Abs\ (\backslash y \rightarrow \exists \wr (Abs\ (\backslash x \rightarrow y \wr x)))) = \lambda x_0.\exists x_1\ (x_0\ x_1)$, which is what we want.

2.6 The Lambek calculus

The rule component of an $(A)AB$ deductive systems consists of rules for building larger structures from simpler ones. So, for example, if we have a structure Γ of category a/b and a structure Δ of category b, we can build a structure $\Gamma\,\Delta$ of category a. Under this perspective the AB and AAB grammars are not substantially different from phrase-structure grammars where one builds a structure $\Gamma\,\Delta$ of category C from a structure Γ of category A and a structure Δ of category B, if the production $A\,B \rightarrow C$ is among the axioms of the grammar.

A great enrichment to the field of categorial grammar was given by the work of Joachim Lambek. In [Lambek, 1958], new rules were added to the composition rules of the system AAB. These rules *decompose* a structured sequent in a way that can be spelled out as follows.

- if a structure $a\,\Gamma$ (that is, a structure with a as the leftmost category) is of category c, then the structure Γ is of category $a\backslash c$.
- if a structure $\Gamma\,a$ is of category c, then the structure Γ is of category c/a.

These rules are called *introduction* rules of the slashes (in opposition to the *cancellation* or elimination rules of AB) as they introduce a slash connective in the conclusion, or also rules of *proof* (in opposition to the rules of *use* of AB), as they state how to prove formulas with a main slash connective.

We present the product-free Lambek calculus. In [Lambek, 1958] the calculus was associative, that is the following principles were assumed as axioms (where $y \leftrightarrow x$ is shorthand for $y \rightarrow x$ and $x \rightarrow y$).

$$a,\ (b,\ c) \leftrightarrow (a,\ b),\ c$$

The unbracketed notation we use for the antecedents implicitly expresses these laws. An actual simplification, with respect to the logic in [Lambek, 1958] is the absence of *product* rules, that is of rules which deal with formulas of the form $a \otimes b$. This is because we will dedicate a very special attention to the product in later chapters. We call the following logic the associative Lambek calculus without product, and we refer to it as L.

Definition 39. A product-free Lambek deductive system, L, is a triple $\langle \mathcal{F}, AX, R\rangle$ where

$\mathcal{F}$ is a set of formulas,

$AX = \{a \rightarrow a \mid a \in \mathcal{F}\}$

R consists of the following rules:

- Elimination rules:

$$\frac{\Gamma \to c/b \quad \Delta \to b}{\Gamma\,\Delta \to c} \qquad\qquad \frac{\Gamma \to a \quad \Delta \to a\backslash c}{\Gamma\,\Delta \to c}$$

- Introduction rules, $\Gamma \in \mathcal{F}^+$:

$$\frac{\Gamma\, b \to c}{\Gamma \to c/b} \qquad\qquad \frac{a\,\Gamma \to c}{\Gamma \to a\backslash c}$$

A product-free Lambek grammar based on L is a categorial grammar based on L.

We show now the system L at work with some examples.

Example 13. Lambek grammar A_2:

$A_2 =$

$$\begin{array}{llll}
\{ & \textit{John} & \to & n, \\
 & \textit{Mary} & \to & n, \\
 & \textit{someone} & \to & s/(n\backslash s), \\
 & \textit{everyone} & \to & (s/n)\backslash s, \\
 & \textit{loves} & \to & (n\backslash s)/n, \\
 & \textit{ismissing} & \to & (s/(n\backslash s))\backslash s\ \}
\end{array}$$

Consider the sentence *John ismissing*. This sentence is deduced as follows.

(2.1)

$$\frac{\dfrac{\dfrac{n \to n \quad n\backslash s \to n\backslash s}{n,\ n\backslash s \to s}}{n \to s/(n\backslash s)} \qquad (s/(n\backslash s))\backslash s \to (s/(n\backslash s))\backslash s}{n,\ (s/(n\backslash s))\backslash s \to s}$$

Observe that in the *AAB* grammar A_1 we had to assign the category $n\backslash s$ to *ismissing* to accept this as a sentence. However, this is not the case for the Lambek grammar A_2. In fact, one may verify that $\vdash_L (s/(n\backslash s))\backslash s \to n\backslash s$, that is to say that a third order verb as $(s/(n\backslash s))\backslash s$ behaves also as a first order one as $n\backslash s$.

Consider now the example *someone loves everyone* of the previous section. In this example, we wrote the lambda terms corresponding to each derivation, although we have not presented the term construction procedure for L yet. The first deduction corresponds to the object wide scope reading, while the second to the subject wide scope reading. Thus the Lambek grammar A_2 generates both readings even though its lexicon lacks the assignments that were required in A_1 to achieve this result. For reasons of space we use again the macros previously introduced and expand them into the deduction on a readability basis.

Example 14. Deductions of *someone loves everyone* in A_2

1. $\forall x\, \exists y\, ((\textsc{loves}\, x)\, y)$

$$\cfrac{\cfrac{\cfrac{qs \to s/iv \quad \cfrac{tv \to iv/n \quad n \to n}{tv,\ n \to iv}}{qs,\ tv,\ n \to s}}{qs,\ tv \to s/n} \quad qo \to (s/n)\backslash s}{qs,\ tv,\ qo \to s}$$

2. $\exists y\, \forall z\, ((\textsc{loves}\, z)\, y)$

$$\cfrac{qs \to s/iv \quad \cfrac{\cfrac{\cfrac{\cfrac{n \to n \quad \cfrac{tv \to iv/n \quad n \to n}{tv,\ n \to n\backslash s}}{n,\ tv,\ n \to s}}{n,\ tv \to s/n} \quad qo \to (s/n)\backslash s}{n,\ tv,\ qo \to s}}{tv,\ qo \to iv}}{qs,\ tv,\ qo \to s}$$

In the object wide scope reading (deduction 1) the *hypothesis* n is 'consumed' by the verb phrase tv and discharged after the subject second order noun phrase, qs, has been composed. The result s/n is taken in input by the object second order noun phrase, qo. In the subject wide scope reading (deduction 2) even a more drastic *restructuring* intervenes: both arguments of the verb phrase are assumed at the hypothetical level. These hypotheses are used in a first-in first-out way allowing the composition of the object noun phrase before that of the subject noun phrase.

Deduction 2 in the previous example shows that in L the verb assignment $(n\backslash s)/n$ derives $n\backslash(s/n)$. Indeed this is an instance of the more general scheme

$$(a\backslash b)/c \leftrightarrow a\backslash(b/c)$$

which justified the notation $n\backslash s/n$ in [Lambek, 1958].

In L, the following principles can be proved.

Example 15. Characteristic theorems of *L*:

$$\begin{array}{cc}
a \to c/(a\backslash c) & a \to (c/a)\backslash c \\
(a\backslash b)/c \to a\backslash(b/c) & a\backslash(b/c) \to (a\backslash b)/c \\
a\backslash b \to (c\backslash a)\backslash(c\backslash b) & b/a \to (b/c)/(a/c) \\
a\backslash b \to (a\backslash c)/(b\backslash c) & b/a \to (c/b)\backslash(c/a)
\end{array}$$

Moreover, different kinds of *derived inference rule* can be adopted, and could replace those we used in defining *L*, as it is done in [Lambek, 1958] or [Zielonka, 1981] among others.

With regard to semantics, we can see that every inference rule of *L* has its counterpart in the lambda term language, as in the case of (*A*)*AB* calculus. Furthermore, we can see that each lambda calculus operation, apart from pairing and projecting, has a counterpart in a rule of *L*[8]. The correspondence between Lambek calculus, as a typed logic, and lambda terms is called Curry-Howard *correspondence*[9]. Such correspondence does not hold for (*A*)*AB*, as in (*A*)*AB* there are no general rules for introducing abstract terms. The addition of the slash introduction rules completes the correspondence between categorial logic and typed term systems. We propose once more *L* with lambda term annotation. Observe that, as in the case of the labeled *AAB*, we make explicit use of brackets.

Definition 40. Semantic labeling for *L*.

$$\lambda\, x.x \colon a \to a$$

$$\frac{v \colon (\Gamma, a) \to c}{\lambda x \lambda y.(v\ \langle x, y\rangle) \colon \Gamma \to c/a} \qquad \frac{v \colon (a, \Gamma) \to c}{\lambda x \lambda y.(v\ \langle y, x\rangle) \colon \Gamma \to a\backslash c}$$

$$\frac{v \colon \Gamma \to c/b \quad u \colon \Delta \to b}{\lambda x.(v\ \pi x\ (u\ \pi' x)) \colon (\Gamma,\ \Delta) \to c} \qquad \frac{u \colon \Gamma \to b \quad v \colon \Delta \to b\backslash c}{\lambda x.(v\ \pi' x\ (u\ \pi x)) \colon (\Gamma,\ \Delta) \to c}$$

Moreover, as the labeled calculus is bracketed we should assign a term also to the associativity laws which we express in rule format and omitting the

[8] In fact, at this point of the discussion, neither *L* nor (*A*)*AB* have rules introducing pair or projection terms. This is due to the absence of product rules in the systems we are considering now. We will soon extend (*A*)*AB* and *L* to handle product formulas and we will see the full correspondence with the lambda calculus.

[9] While Curry-Howard *isomorphism* is the correspondence between intuitionistic logic and lambda calculus. Observe that there is not an isomorphism between proofs in *L* and lambda terms, as the same term may encode more than one *L* proof.

most external brackets.

$$\frac{v\colon (\Gamma,\ \Delta),\ \Lambda \to c}{\lambda x.(v\ \langle\langle \pi x, \pi\pi' x\rangle, \pi'\pi' x\rangle)\colon \Gamma,\ (\Delta,\ \Lambda) \to c}$$

$$\frac{v\colon \Gamma,\ (\Delta,\ \Lambda) \to c}{\lambda x.(v\ \langle \pi\pi x, \langle \pi'\pi x, \pi' x\rangle\rangle)\colon (\Gamma,\ \Delta),\ \Lambda \to c}$$

Let us derive of the labeled variant of the associative cancellation rules in the labeled *L*. Again, we omit the most external brackets and we reduce the term to normal form at each step of the derivation.

Example 16. Deduction of one of the associative cancellation rules in *L*.

$$\frac{\dfrac{\dfrac{v\colon \Gamma \to c/b \quad \dfrac{u\colon \Delta \to b/a \quad \lambda x.x\colon a \to a}{\lambda x.(u\ \pi x\ \pi' x)\colon \Delta,\ a \to b}}{\lambda x.(v\ \pi x\ (u\ \pi\pi' x\ \pi'\pi' x))\colon \Gamma,\ (\Delta,\ a) \to c}}{\lambda x.(v\ \pi\pi x\ (u\ \pi'\pi x\ \pi' x))\colon (\Gamma,\ \Delta),\ a \to c}}{\lambda x \lambda y.(v\ \pi x\ (u\ \pi' x\ y))\colon \Gamma,\ \Delta \to c/a}$$

2.7 Product types

In the previous sections, we presented the (*A*)*AB* calculus and *associative product-free* Lambek calculus. Although they may have product formulas, these systems have no *rules* to handle them. However, in the original formulations of [Lambek, 1958, 1961], Lambek calculi also include product formulas and rules to handle them. (*A*)*AB* calculi, instead, are customarily presented product-free. The only formulation we know of an *AB* calculus with product is given in [Kandulski, 1988].

Definition 41. Inference rules of the $AB^{\otimes}$ deductive system:

$$\frac{\Gamma \to a \quad \Delta \to b}{\Gamma\,\Delta \to a \otimes b}$$

$$\frac{\Gamma \to a/b \quad \Delta \to b}{\Gamma\,\Delta \to a} \qquad\qquad \frac{\Gamma \to b \quad \Delta \to b\backslash a}{\Gamma\,\Delta \to a}$$

The system $AB^{\otimes}$ consists of the basic cancellation rules of the system *AB* and of the *product rule*. The semantic labeling of the product rule is as follows:

$$\frac{u\colon \Gamma \to a \quad v\colon \Delta \to b}{\lambda x.\langle (u\ \pi x), (v\ \pi' x)\rangle\colon (\Gamma,\ \Delta) \to a \otimes b}$$

We will largely use the system $AB^{\otimes}$ in the next chapters. Indeed, we can see $AB^{\otimes}$ as the link between the non-associative Lambek calculus presented below and context-free grammars.

Let us look now at the calculus of [Lambek, 1961]: the non-associative Lambek calculus with product, *NL* for short. The calculus was introduced by Lambek in order to handle *bracketed strings of formulas*, that is to say to generate *trees* of formulas rather than *lists* of formulas[10]. As we discussed in the first chapter, product formulas play a more important role here than in the associative system of [Lambek, 1958] where structural completeness holds. We present directly the labeled version of *NL*. The original calculus of [Lambek, 1961] can be obtained from the following formulation by mapping each arrow to a (two-formula) sequent. The logic is also called *pure logic of residuation* in the multi-modal framework which we will discuss in the next section.

Definition 42. *Pure logic of residuation*, *NL*.

- Identities:

Axioms

$$\lambda x.x\colon a \to a$$

Cut

$$\frac{v\colon a \to b \quad u\colon b \to c}{\lambda x.(u\ (v\ x))\colon a \to c}$$

- Residuation rules:

$$\frac{v\colon a \otimes b \to c}{\lambda x \lambda y.(v\ \langle x, y\rangle)\colon a \to c/b} \qquad \frac{v\colon a \to c/b}{\lambda x.(v\ \pi x\ \pi' x)\colon a \otimes b \to c}$$

$$\frac{v\colon a \otimes b \to c}{\lambda x \lambda y.(v\ \langle y, x\rangle)\colon b \to a\backslash c} \qquad \frac{v\colon b \to a\backslash c}{\lambda x.(v\ \pi' x\ \pi x)\colon a \otimes b \to c}$$

Since there is no structural connective in *NL*, the definition of *generation* is slightly more complex.

Definition 43. A *NL* grammar with semantic labels G generates a string $w_0 \ldots w_m \in V_t^*$ and assigns it the semantic term t if and only if *NL* generates $f\colon c \to s$ and $t = (f\ t')$, where

[10] Following [Buszkowski, 1997], we are assuming that the tree generated is the antecedent of the conclusion of a deduction. However, [Tiede, 1998] claims that the tree structures generated by deductions in the (non-associative) Lambek calculus are the deductions themselves.

1. $w_0 \dots w_m \Rightarrow^* a_0 \dots a_n$ by application of axioms

$$\begin{array}{ll} t_i\colon w_i \to a_i \in Lex, & 0 \leqslant i \leqslant m \\ t_i\colon \epsilon \to a_i \in Lex, & 0 \leqslant i \leqslant n \end{array}$$

2. $a_0 \dots a_n \Rightarrow^* c$ by application of the transition axiom

$$a\,b \to a \otimes b$$

3. t' is obtained by replacing in c each a_i with t_i, each (with $\langle$ and each) with $\rangle$ and each $\otimes$ with a comma.

Semantic interpretation

Formulas of *NL* can be interpreted in *modal frames*. A modal frame F, is a pair $(W, \{R^3\})$, where W is a set and R^3 is a ternary relation, see [Moortgat, 1997b, Buszkowski, 1997]. A *model* is a pair (F, v), of a modal frame F and an *interpretation* function v which maps formulas into subsets of W. Assuming that the interpretation of every atom is given, compound formulas are interpreted as follows (here we use the symbol $\Rightarrow$ here as shorthand for *implies*).

Definition 44. Interpretation of formulas:

$$\begin{array}{lcl} v(a \otimes b) & = & \{\, x \mid \exists y\, \exists z\, (R^3(x,y,z)\ \&\ y \in v(a)\ \&\ z \in v(b))\, \} \\ v(a/b) & = & \{\, y \mid \forall x\, \forall z\, ((R^3(x,y,z)\ \&\ z \in v(b)) \ \Rightarrow\ x \in v(a))\, \} \\ v(b\backslash a) & = & \{\, z \mid \forall x\, \forall y\, ((R^3(x,y,z)\ \&\ y \in v(b)) \ \Rightarrow\ x \in v(a))\, \} \end{array}$$

We refer to [Moortgat, 1997b] and to [Kurtonina, 1995] and to the references therein for the proofs of soundness and completeness of *NL* with respect to modal frames. The interpretation of Lambek formulas with respect to other models is given in [Buszkowski, 1997]. Here we state the following result.

Proposition 3. [Došen, 1992]
$\vdash_{NL} a \to c$ iff $v(a) \subseteq v(c)$ for every evaluation v on every ternary frame.

2.7.1 Linguistic analysis in $\mathcal{NL}$

Since this book is largely about $\mathcal{NL}$, we present here some examples of syntactic analysis of natural language constructs. The main propose of the following analyses is to familiarize with $\mathcal{NL}$ and with some features of categorial grammar. The examples are meant to enlighten the *strong generative capacity* and the interface between syntax and semantics. We will work on the following assignments.

Example 17. Lexicon $\mathcal{PS}$.

$$\begin{array}{rll}
give: & gave & \rightarrow (n\backslash s)/(n\otimes n)\\
seem: & seems & \rightarrow (n\backslash s)/s\\
seem: & seems & \rightarrow (n\backslash s)/i\\
\lambda x.(persuade\ \pi x\ (\pi' x\ \pi x)): & persuaded & \rightarrow (n\backslash s)/(n\otimes i)\\
\lambda x\lambda y.(promise\ \pi x\ (\pi' x\ y)\ y): & promised & \rightarrow (n\backslash s)/(n\otimes i)\\
\lambda x\lambda y.(want\ (x\ y)\ y): & wants & \rightarrow (n\backslash s)/i\\
praise: & praises & \rightarrow (n\backslash s)/n\\
disappear: & disappear & \rightarrow b\\
a: & a & \rightarrow n/c\\
advice: & advice & \rightarrow c\\
\lambda x.x: & to & \rightarrow i/b\\
\lambda x\lambda y.\langle y,x\rangle: & to & \rightarrow (n\backslash(n\otimes n))/n\\
\lambda x\lambda y.(x\ y\ y): & himself & \rightarrow ((n\backslash s)/n)\backslash(n\backslash s)
\end{array}$$

As we said before, the semantic terms in a categorial lexicon may violate the constraints, stated in definition 31, on linear lambda terms, which in turn apply to derivational semantics. It is instructive to see how the lexical and derivational processes naturally interact to produce appropriate, or at least partially appropriate, semantic representations of the input sentences. As before, we we apply macros on the basis of a compromise between space and readability.

Example 18. The first example is concerned the *reflexive pronoun*. In the lexicon we have $\lambda x\lambda y.(x\ y\ y): himself \rightarrow ((n\backslash s)/n)\backslash n\backslash s$. Thus $himself$ needs a transitive verb *tv* and a noun phrase *n* to its left to produce a sentence. At the semantic level, the first argument of *himself* is applied to the second and the result is applied again to the second argument.

Don Rodrigo praises himself.

1. Derivation:

$$\dfrac{\dfrac{\lambda x.x\colon tv\backslash iv \to tv\backslash iv}{\lambda x.(\pi' x\ \pi x)\colon tv \otimes tv\backslash iv \to n\backslash s}}{\lambda x.(\pi'\pi' x\ \pi\pi' x\ \pi x)\colon n \otimes (tv \otimes tv\backslash iv) \to s}$$

2. Lexical semantics:

$$\langle DR, \langle praise, \lambda x\lambda y.(x\ y\ y)\rangle\rangle$$

3. Meaning representation:

$$(\lambda x.(\pi'\pi' x\ \pi\pi' x\ \pi x)\ \langle DR, \langle praise, \lambda x\lambda y.(x\ y\ y)\rangle\rangle) = (praise\ DR\ DR)$$

In example 18, we saw that an abstractor can bind multiple variables in a lexical term. A similar situation occurs with verbs like *wants*, *tries*, *promises* and *persuades*. An assignment for *each other*, as in *Renzo and Lucia love each other*, should express reciprocity of the verb relation, that is its *symmetry*. If we want that such a sentence is translated as

$$((loves\ L)\ R) \wedge ((loves\ R)\ L)$$

we need the same bound variable to occur in different places.

Below we give an example with *want*.

Example 19.

Don Abbondio wants to disappear.

1. Derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\lambda x.x\colon i/b \to i/b}{\lambda x.(\pi x\ \pi' x))\colon i/b \otimes b \to i} \qquad \dfrac{\dfrac{\lambda x.x\colon iv/i \to iv/i}{\lambda x.(\pi x\ \pi' x))\colon iv/i \otimes i \to iv}}{\lambda x\lambda y.(y\ x)\colon i \to (iv/i)\backslash iv}}{\lambda x\lambda y.(y\ (\pi x\ \pi' x)))\colon i/b \otimes b \to (iv/i)\backslash iv}}{\lambda x.(\pi x\ (\pi\pi' x\ \pi'\pi' x))\colon iv/i \otimes (i/b \otimes b) \to iv}}{\lambda x.(\pi\pi' x\ (\pi\pi'\pi' x\ \pi'\pi'\pi' x)\ \pi x)\colon n \otimes (iv/i \otimes (i/b \otimes b)) \to s}$$

2. Lexical semantics:

$$\langle DA, \langle \lambda x\lambda y.(want\ (x\ y)\ y), \langle \lambda x.x, disappear\rangle\rangle\rangle$$

3. Meaning representation:

$$(\mathit{want}\ (\mathit{disappear}\ \mathit{DA})\ \mathit{DA})$$

The multiple binding in $\lambda x \lambda y.(\mathit{want}\ (x\ y)\ y)$ guarantees that the subject of the main clause is the subject of the embedded clause. The verb *promised* has a similar behavior. Instead *persuaded* enforces object control of the embedded clause. An appropriate assignment for *persuaded*, as in *Padre Cristoforo persuaded Renzo to leave* is

$$\lambda x.(\mathit{persuade}\ \pi x\ (\pi' x\ \pi x)) : \mathit{persuaded} \rightarrow (n \backslash s)/(n \otimes i)$$

In its simpler occurrences, a verb like *seem* exhibits the following two behaviors: in one case, its subject is the subject of the infinitive embedded clause, in the second, it takes a 'dummy' subject and a finite embedded clause. However, the two constructs may be seen as truth-conditionally equivalent. In [Gazdar et al., 1985], *vacuous abstraction* is exploited at the lexical level in order to account for the occurrence of dummy arguments. We show how the same device can be applied in the logical setting.

Example 20.

Azzeccagarbugli seems to understand.

1. Derivation, see 19:

$$\lambda x.(\pi\pi' x\ (\pi\pi'\pi' x\ \pi'\pi'\pi' x)\ \pi x) : n \otimes (iv/i \otimes (i/b \otimes b)) \rightarrow s$$

2. Lexical semantics:

$$\langle AG, \langle \lambda x \lambda y.(\mathit{seem}\ (y\ x)), \langle \lambda x.x, \mathit{understand} \rangle\rangle\rangle$$

3. Meaning representation:

$$(\mathit{seem}\ (\mathit{understand}\ AG))$$

It seems that Azzeccagarbugli understands.

1. Derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{iv \to n\backslash s}{n \otimes iv \to s} \quad \begin{matrix} s/s \to s/s \\ \vdots \\ s \to (s/s)\backslash s \end{matrix}}{n \otimes iv \to (s/s)\backslash s}}{s/s \otimes (n \otimes iv) \to s} \quad \begin{matrix} iv/s \to iv/s \\ \vdots \\ s \to (iv/s)\backslash iv \end{matrix}}{s/s \otimes (n \otimes iv) \to (iv/s)\backslash iv}}{iv/s \otimes (s/s \otimes (n \otimes iv)) \to n\backslash s} \quad \begin{matrix} qs \to qs \\ \vdots \\ iv \to (s/iv)\backslash s \end{matrix}}{iv/s \otimes (s/s \otimes (n \otimes iv)) \to (s/iv)\backslash s}}{qs \otimes (iv/s \otimes (s/s \otimes (n \otimes iv))) \to s}$$

2. Derivational semantics:

$$\lambda x.(\pi x\ (\pi\pi' x\ (\pi\pi'\pi' x\ (\pi'\pi'\pi'\pi' x\ \pi\pi'\pi'\pi' x))))$$

3. Lexical semantics:

$$\langle \lambda x.(x\ y), \langle \lambda x \lambda y.(seem\ x), \langle \lambda x.x, \langle AG, understand \rangle\rangle\rangle\rangle$$

4. Meaning representation:

$$(seem\ (understand\ AG))$$

The semantic assignment $\lambda x \lambda y.(seem\ x)$ for *seem* in the 'dummy' subject construction, contains a vacuous abstraction, which is used to discard the meaning contribution of the 'dummy' subject *it* from the final meaning representation. Thus the two constructs are rendered truth-conditionally equivalent[11]. In general, vacuous abstraction can be exploited to discard from the final meaning representation the contribution of arguments which are not relevant.

Although the previous sentences were quite simple from the syntactic (and semantic) point of view, we will see that the product operator improves notably on the structural and semantic description in case of multi-argument verbs. Consider the following two examples, from which we omit the derivational component, as easily recoverable from the given sequents.

[11] Observe that *vacuous abstraction*, that is an abstraction that binds no variable in its scope, is not as rare as it may seem. For instance, the parentheses of a formal system, whose only role is to disambiguate, translate to terms containing such vacuous abstractions.

Example 21.

Agnese gave Renzo an advice.

1. $n \otimes (iv/(n \otimes n) \otimes (n \otimes (n/c \otimes c))) \rightarrow s$
2. Derivational semantics:

$$\lambda x.(\pi\pi' x \, \langle \pi\pi'\pi' x, (\pi\pi'\pi'\pi' x \; \pi'\pi'\pi'\pi' x)\rangle \; \pi x)$$

3. Lexical semantics:

$$\langle A, \langle give, \langle R, \langle a, advice\rangle\rangle\rangle\rangle$$

4. Meaning representation:

$$((give \; \langle R, (a \; advice)\rangle) \; A)$$

Agnese gave an advice to Renzo.

1. $n \otimes (iv/(n \otimes n) \otimes ((n/c \otimes c) \otimes ((n\backslash(n \otimes n))/n \otimes n))) \rightarrow s$
2. Derivational semantics:

$$\lambda \, x.(\pi\pi' x \; (\pi\pi'\pi'\pi' x \; \pi'\pi'\pi'\pi' x \; (\pi\pi\pi'\pi' x \; \pi'\pi\pi'\pi' x)) \; \pi x)$$

3. Lexical semantics:

$$\langle A, \langle give, \langle\langle a, advice\rangle, \langle \lambda x \lambda y.\langle y, x\rangle, R\rangle\rangle\rangle\rangle$$

4. Meaning representation:

$$((give \; \langle R, (a \; advice)\rangle) \; A)$$

Observe that in both deductions the head *gave* is assigned the same category $(n\backslash s)/(n \otimes n)$. Nonetheless we obtain both syntactic and semantic constructions. Both syntactic structures are right branching, in accordance with intuition. Furthermore, both constructions express the same truth-conditional interpretation. While other frameworks would appeal to *meaning postulates* to express equivalence of the two constructs (see for instance [Gazdar et al., 1985]) we obtain this result by assigning the preposition *to* the term $\lambda x \lambda y.\langle y, x\rangle$, encoding a commutation in the order of its arguments. Observe that while syntactically inadequate, the assignment $(n\backslash s)/n/n$ for the ditransitive verb should be associated to two different semantic translations to achieve the same effect.

2.8 Multi-modal Type-logical Grammars

Multi-modal type-logical grammars are a generalization of categorial grammars. One may observe that the rules of *associativity* of the *syntactic calculus* of [Lambek, 1958], have a somewhat special status: they are *structural rule*. In the product-free system L, these rules were assumed implicitly, by treating the antecedent of every sequent as a *list*. However, in the labeled variant of L, where brackets appeared, we explicitly expressed associativity by means of two rules. From this perspective, the NL can be seen as the result eliminating the rules of associativity from the syntactic calculus.

The type-logical approach adopts the opposite perspective. In this setting, NL is the basic deductive system and other logics are defined by adding to NL some packages of structural rules. Thus, a type-logical system is a pair $\langle Q, NL\rangle$, where Q is a set of structural postulates. As we mentioned, in the type-logical framework NL is also called the *pure logic of residuation*, see [Moortgat, 1997b], since everything that is derivable in NL is either an axiom or is derived through *residuation principles* and *cut*.

The type-logical analysis of a language always starts within NL as the most restrictive deductive system. Postulates may be added mainly for two reasons. Either to *increase the generative power* of the system or to capture *generalizations* on classes of lexical items. For example, given a grammar G based on $D = \langle Q, NL\rangle$, for some structural package Q, the extension of G to G' based on $D' = \langle Q' \cup Q, NL\rangle$ for some set of postulates Q', may allow G' to derive sentences that were not derivable in G, or it may allow G' to derive a class of lexical categories in G from a single category, in such a way that the lexicon of G' results more compact. In this second case, the structural package is used to capture a generalization on linguistic structure.

More concretely, the addition of a subset of the following postulates to NL defines a hierarchy of logics, see [Kurtonina and Moortgat, 1997].

(2.2) Associativity, A:

$$A_1 \quad \lambda x.\langle\langle \pi x, \pi\pi' x\rangle, \pi'\pi' x\rangle : a \otimes (b \otimes c) \rightarrow (a \otimes b) \otimes c$$

$$A_2 \quad \lambda x.\langle \pi\pi x, \langle \pi'\pi x, \pi' x\rangle\rangle : (a \otimes b) \otimes c \rightarrow a \otimes (b \otimes c)$$

(2.3) Permutation, P:

$$\lambda x.\langle \pi' x, \pi x\rangle : b \otimes a \rightarrow a \otimes b$$

(2.4) Contraction, C:

$$\lambda x.\langle x, x\rangle : a \rightarrow a \otimes a$$

(2.5) Weakening, W:

$$\begin{array}{cc} W_1 & W_2 \\ \lambda x.\pi x\colon a \otimes b \rightarrow a & \lambda x.\pi' x\colon b \otimes a \rightarrow a \end{array}$$

The addition to NL of some subset of these postulates identifies a logic. For instance:

Example 22.

Intuitionistic logic: $\mathbb{L} = \langle \{A, P, W, C\}, NL \rangle$.

Commutative Lambek calculus: $LP = \langle \{A, P\}, NL \rangle$, [van Benthem, 1991, Hendriks, 1993].

Lambek *syntactic calculus* $L^* = \langle \{A\}, NL \rangle$, [Lambek, 1958].

However, none of these systems is suited for natural language analysis. Linguistic reasoning is highly restrictive on the multiplicity of the resources, thus postulates C and W should not be assumed in a natural language grammar[12]. The natural candidates for natural language grammar seem to lay in between the Lambek-van Benthem calculus and the syntactic calculus. However, while the Lambek-van Benthem calculus is too inclusive, as it would admit any permutation of the input string, the syntactic calculus is too restrictive. For example, although this system can easily handle peripheral extraction in unbounded contexts (as in *Who did Don Abbondio say ... he met?*), it cannot as easily cope with unbounded embedded dependencies (as in *What does Don Abbondio think ... he will tell to Renzo?*).

In second place, the introduction of structural postulates into the grammar system, in the naive way in which it has been presented before, obscures more properties of the language structure than it reveals. For instance, the syntactic calculus loses track of all the structural information that a derivation in NL encodes. The price to pay for such a *global* introduction of structural reasoning may be too high with respect to its benefits.

The *multi-modal* setting, [Moortgat, 1996, 1997b] generalizes the notion of the syntactic category in the following way.

Definition 45. Multi-modal formulas:

$$\mathcal{F} = A \mid /^n_i(\mathcal{F}^1, \ldots, \mathcal{F}^n) \mid \backslash^n_i(\mathcal{F}^1, \ldots, \mathcal{F}^n) \mid \otimes^n_i (\mathcal{F}^1, \ldots, \mathcal{F}^n)$$

[12] In fact, some syntactic phenomena seem to involve multiple binding at the derivational level. Constructions like *file x without reading x* are dealt with in the *combinatory categorial* setting of [Steedman, 2000b] the by means combinator $(a\backslash b)/c \rightarrow (a/c)\backslash(b/c)$, which would be derived in the type logical setting by means of contraction C.

In this definition, n is a positive integer expressing the arity of the connective and i another integer, the so called *composition mode* distinguishing for example $\otimes^n_j$ from $\otimes^n_k$. In previous systems, $n = 2$ and there was only one composition mode. In [Buszkowski, 2005], one can find a Gentzen style sequent calculus formulation for generalized Lambek calculus with n-ary type forming operators. In linguistic applications, one usually works with $n \leqslant 2$. Thus the multi-modal formulation of *NL* results as follows.

Definition 46. *Multi-modal pure logic of residuation*, for every i:

- Identities:

Axioms Cut

$$\lambda x.x\colon a \to a \qquad \frac{v\colon a \to b \quad u\colon b \to c}{\lambda x.(u\ (v\ x))\colon a \to c}$$

- Residuation rules:

$$\frac{v\colon a \otimes_i b \to c}{\lambda x \lambda y.(v\ \langle x, y\rangle)\colon a \to c/_i b} \qquad \frac{v\colon a \to c/_i b}{\lambda x.(v\ \pi x\ \pi' x)\colon a \otimes_i b \to c}$$

$$\frac{v\colon a \otimes_i b \to c}{\lambda x \lambda y.(v\ \langle y, x\rangle)\colon b \to a\backslash_i c} \qquad \frac{v\colon b \to a\backslash_i c}{\lambda x.(v\ \pi' x\ \pi x)\colon a \otimes_i b \to c}$$

Observe that every connective obeys the laws of the pure logic of residuation. However, by specifying the *modes* appearing in the structural rules, one specifies also which syntactic resources are subject to these rules. Hence, which configurations of categories are subject to what kind of restructuring.

In recent years, the research on type-logical has tried to identify the packages of structural rules required for natural language analysis. [Moortgat, 1996] proposes the following forms of *mixed commutativity*, *MP*, and *mixed associativity*, *MA* to deal with discontinuous dependencies.

Example 23. Mixed postulates:

$$\begin{array}{ll} MP & a \otimes_i (b \otimes_j c) \leftrightarrow b \otimes_j (a \otimes_i c) \\ MA & a \otimes_i (b \otimes_j c) \leftrightarrow (a \otimes_i b) \otimes_j c \end{array}$$

We shall remark that the addition of the unlabeled variants of *MP* and *MA* to *NL* would collapse the system into *LP*, as proved in [Moortgat, 1988]. However, the introduction of such modalized postulates guarantees that only specific configurations will access their restructuring power.

More recently, [Vermaat, 2005] analyzes the realization of long distance dependencies in several languages with the tools of type-logical grammar. The shape of Vermaat's postulates closely resembles that of Moortgat's postulates, the main difference consists in the use of the unary modalities to mark the substructure subject to displacement.

The unary operators have a special role in the multi-modal setting. They are the diamond $\Diamond_i = \otimes^1_i$ and the box $\Box_i$, and are subject to the following pure residuation logic.

Definition 47. Unary residuation rules:

$$\frac{v\colon \Diamond_i a \to c}{\lambda x \lambda y.(v\ \langle x, y\rangle)\colon a \to \Box_i c} \qquad \frac{v\colon a \to \Box_i c}{\lambda x.(v\ \pi x\ \pi' x)\colon \Diamond_i a \to c}$$

The interpretation of unary formulas is given by extending the modal frame F which we saw before to a pair $(W, \{R^2, R^3\})$, where R^2 a binary relation. The model-theoretic interpretation v is extended to unary formulas in the straightforward way:

$$\begin{aligned} v(\Diamond a) &= \{\, x \mid \exists y\, (R^2(x,y)\ \&\ y \in v(a))\,\} \\ v(\Box a) &= \{\, y \mid \forall x\, (R^2(x,y) \ \Rightarrow\ x \in v(a))\,\} \end{aligned}$$

The unary operators have found a wide range of application in categorial linguistics. One option is to use them for a regimented interaction with the structural module of the grammar. Vermaat adopts the following postulates for long distance dependencies:

Example 24. Vermaat's postulates:

$$\begin{gathered} \Diamond a \otimes (b \otimes c) \to (\Diamond a \otimes b) \otimes c \\ (a \otimes b) \otimes \Diamond c \to a \otimes (b \otimes \Diamond c) \\ \Diamond a \otimes (b \otimes c) \to b \otimes (\Diamond a \otimes c) \\ (a \otimes b) \otimes \Diamond c \to (a \otimes \Diamond c) \otimes b \end{gathered}$$

One may observe that only the specific ternary configurations in which the diamond marker appears are subject to these forms of restructuring. The diamond decoration, in turn, is ultimately triggered by some lexical item. Hence *restructuring* is always *local*, lexically driven and controlled. We shall also remark that Vermaat claims that such package of structural postulates is *language universal*, that means that it is capable of accounting for all forms of variation across natural languages.

Recent literature on type-logical grammar exemplifies a wide range of syntactic phenomena which can be elegantly analyzed through the multimodal devices. [Heylen, 1999] develops a feature theoretic syntax relying of the unary logic. He also explores the expressivity of various forms of unary *distribution postulates* expressing different forms feature percolation. In [Bernardi, 2002], the unary operators are used to express *co-occurrence restrictions* on polarity sensitive items. [Kraak, 1995] analyzes the phenomenon of so called *clitic climbing* in Fench. In her treatment, the box marks the verbal head and the clitic pronouns: unary distribution postulates then enforce clitic climbing and 'attachment' of the clitic to the verb. Finally in [Hendriks, 1999], the diamond is used as a prosodic selector.

We mention the fact that also in the framework of combinatory categorial grammar, as for instance in [Kruijff and Baldridge, 2003], forms of multimodal control have been succesfully used.

2.9 Formal properties of categorial grammars

We conclude this chapter by recalling some results on the generative power of categorial grammars.

The weak equivalence of *CF* grammars and *AB* grammars was proved in [Bar-Hillel et al., 1964a]. [Buszkowski, 1988] establishes the equivalence in strong generative power of the two systems, see also [Buszkowski, 1997]. The weak equivalence of $AB^{\otimes}$ and *CF* grammars is proved in [Kandulski, 1988].

Concerning Lambek style categorial grammars, [Buszkowski, 1986] and [Kandulski, 1988] prove the equivalence of *NL* grammars, respectively without and with product, and *CF* grammars. The proof of context-freeness of the associative Lambek calculus, which has been an open problem for more than thirty years, was given by [Pentus, 1993].

The generative power of multi-modal type-logical grammar depends on the package of structural postulates which is assumed by the deductive engine and on the way the lexical resources are allowed to interact with the structural module. [Carpenter, 1996] shows that even without rules which duplicate (e.g. *contraction*) or erase (e.g. *weakening*) material, multi-modal type-logical grammars are Turing-complete, see also [Moot, 2002]. Moot proves that if all postulates are *non-expanding* the multi-modal grammar is equivalent to some context-sensitive grammar.

One may ask what is the generative power of a grammar adopting a specific set of structural postulates as the mixed postulates of [Moortgat, 1996] and [Vermaat, 2005]. Although there is no proof of this, it is largely believed that these systems lay within the *mildly context-sensitive* formalisms.

Chapter 3

Automated Reasoning

In the previous chapter, we have presented linguistic frameworks of increasing complexity. We discussed context-free grammars and categorial grammars. As examples of categorial grammars we saw (non-)associative Ajdukiewicz Bar-Hillel grammars, the product-free Lambek calculus, the non-associative Lambek calculus and the general framework of multi-modal type-logical grammar.

In this chapter, we will reformulate the $\mathcal{CF}$ and $(A)AB$ system as *parsing systems*, that is to say as deductive systems that take advantage of the linear order of the syntactic categories involved to build a deduction.

Our contribution consists in the application of the CYK parsing systems to $AB^{\otimes}$ grammars. A formulation CYK deductive parser for AAB grammars can be found also in [Shieber et al., 1995]. We present a similar parser for $AB^{\otimes}$.

CYK for $AB^{\otimes}$ has a natural way of encoding the cancellation rules, while it is rather inelegant in respect to the product rule. For this reason we will also formulate a procedure for converting an $AB^{\otimes}$ grammar into an equivalent product-free AB grammar and conversely. The parsing system presented here will be extended to $\mathcal{NL}$ in the following chapters.

3.1 Problems

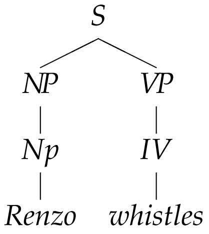
In the chapter 2, we discussed *what* it means for a grammar to generate a sequent and a terminal string. However, we did no specify any method for the process of generation. For example, we observed about example 5 that several other deductions were be available. Let us consider another example.

According to the purely declarative notion of derivation given in definition 11, there are six possible ways of obtaining the sentence *Renzo whistles* in grammar G_3 from example 3 in chapter 2.

Example 25. Derivations of *Renzo whistles* in grammar G_3

1. [*Renzo whistles, Np whistles, NP whistles, NP IV, NP VP, S*]
2. [*Renzo whistles, Np whistles, Np IV, NP IV, NP VP, S*]
3. [*Renzo whistles, Renzo IV, Np IV, NP IV, NP VP, S*]
4. [*Renzo whistles, Np whistles, Np IV, Np VP, NP VP, S*]
5. [*Renzo whistles, Renzo IV, Np IV, Np VP, NP VP, S*]
6. [*Renzo whistles, Renzo IV, Renzo VP, Np VP, NP VP, S*]

All these derivations differ only in the order in which the productions of the grammar are applied. We can see that all encode the following structural description:



We called spurious ambiguity the presence of multiple derivations encoding the same structural description. Clearly, if six different derivations are available for such a simple sentence, spurious ambiguity is a serious problem that may affect drastically the search of a proof for a given input string.

There are several (suboptimal) ways of reducing the degree of spurious ambiguity in automatic proof search. We present some options in this introductory section, and examine the more sophisticated methods in the rest of the chapter.

One may observe that such a redundancy depends also on the shape of the productions of the grammar. Indeed, grammar G_3 could be rephrased as follows.

Example 26.

$G'_3 =$

$$\begin{array}{llll}
\{ & \mathit{NP}\ \mathit{VP} & \rightarrow & S, \\
 & \mathit{Renzo} \mid \mathit{Lucia} \mid \mathit{Det}\ N & \rightarrow & \mathit{NP}, \\
 & \mathit{whistles} \mid \mathit{TV}\ \mathit{NP} & \rightarrow & \mathit{VP}, \\
 & \mathit{loves} & \rightarrow & \mathit{TV}, \\
 & \mathit{every} \mid a & \rightarrow & \mathit{Det}, \\
 & \mathit{man} \mid \mathit{woman} & \rightarrow & N\ \}.
\end{array}$$

However, grammar G'_3 still allows two equivalent derivations for the sentence *Renzo whistles*.

1. [*Renzo whistles*, *NP whistles*, *NP VP*, *S*]

2. [*Renzo whistles*, *Renzo VP*, *NP VP*, *S*]

A second observation is that in building a derivation for a sentence we shall chose a *strategy* for applying the productions of the grammar. For example, for each rule $X_1 \dots X_n \rightarrow X$ we may chose to always expand first either the leftmost symbol X_1 or rightmost symbol X_n. This would reduce the derivations in the following way.

Example 27.

Leftmost derivation:

[*Renzo whistles*, *NP whistles*, *NP VP*, *S*]

Rightmost derivation:

[*Renzo whistles*, *Renzo VP*, *NP VP*, *S*]

Although the choice of one of these recognition strategies may seem to solve the problem of multiple redundant derivations, we immediately see that another problem may arise. A leftmost (resp. rightmost) reduction strategy may enter an infinite loop if the input grammar contains left (resp. right) recursive productions, of the form $Y\, X_1 \dots X_n \rightarrow Y$ (resp. $X_1 \dots X_n\, Y \rightarrow Y$). An example of left recursive rule in English is the following:

$$A\, N \rightarrow N$$

where A is the category of *adjectives*. Such a rule generates, for instance, $happy_1 \dots happy_n$ *man*.

We will see in the next sections that these problems have been solved in extremely elegant and simple ways and that parsing context-free formalisms like the *CF* or $(A)AB^{\otimes}$ systems can be efficiently carried on automatically.

3.2 Deductive parsers

In chapter 2, we presented context-free and categorial grammars as deductive systems. We are going to see that *parsers*, or more precisely *recognizers*, can also be seen as deductive systems. This perspective on parsing is proper to the *deductive parsing* formalism of [Shieber et al., 1995]: parsing a sentence amounts to the construction of a deduction.

Parsers are deductive systems whose *items* (replacing the sequents of the deductive systems previously seen) encode, at least, the portion of the input string that is analyzed and its syntactic category. A deduction succeeds if the entire input is analyzed as being a sentence.

The deductive perspective on parsing plays an important role also in *parsing schemata* theory, see [Sikkel, 1993, 1998]. In fact, the formulation of a parser as a deductive system offers a high level of abstraction over implementational details which allows to easily prove formal properties of the parser, such as its correctness.

Definition 48.

A *parsing system* $\mathcal{D}$ is a triple $\langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{I}$ is a set of *items*, $\mathcal{A}$ is the set of axioms of $\mathcal{D}$ and $\mathcal{R}$ a set of inference rules on items.

A *deduction* in a parsing system $\mathcal{D}$ is defined in the usual way.

We say that a parsing system $\mathcal{D}$ *generates* an item η, denoted $\eta \in \mathcal{D}$, if there is a deduction in $\mathcal{D}$ of η.

In the following sections we will see the most famous and efficient parsing systems for natural language and apply them to $(A)AB^{\otimes}$ grammars. As we said before, these are the CYK parser and the Eraley parser.

3.3 Bottom-up parsers

The CYK algorithm owes its name to the names of its inventors. The algorithm was developed in the early 1960s by John Cocke for parsing $\mathcal{CF}$

grammars. Later [Younger, 1967] showed that this algorithm takes $O(n^3)$ operations to recognize a string of length n. A similar algorithm had been proposed by [Kasami, 1965].

Let us consider the deductive formulation of CYK parser for CF grammars. From now on, for simplicity, we write a CF grammar G as $\langle V_t, S, \mathcal{F}, AX\rangle$ instead of $\langle V_t, S, \langle \mathcal{F}, AX, \{Cut\}\rangle\rangle$ and a Chomsky normal form grammar $G = \langle V_t, S, Lex, \langle \mathcal{F}, AX, \{Cut\}\rangle\rangle$ as $G = \langle V_t, S, \mathcal{F}, Lex, AX\rangle$. The system $\mathcal{CF}_{CYK}$, in its simplest form, works with Chomsky normal form CF grammars without ϵ-productions. However, it can easily be generalized for grammars which are not in CNF.

Definition 49. Let a CNF grammar CF grammar $G = \langle V_t, S, \mathcal{F}, Lex, AX\rangle$ and a string $w_1 \dots w_n$ be given. The parsing system $\mathcal{CF}_{CYK} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R}\rangle$ is defined as follows:

$$\begin{aligned}
\mathcal{I} &= \{\ (i, A, j) \mid A \in \mathcal{F},\ 0 \leqslant i < j < n\ \}\\
\mathcal{A} &= \{\ (i-1, A, i) \mid w_i \to A \in Lex\ \}\\
\mathcal{R} &= \frac{(i, A, k) \quad (k, B, j)}{(i, C, j)}\ \text{if } A\, B \to C \in AX
\end{aligned}$$

As we said, the CYK system builds deductions *bottom-up*, that is from premises to conclusion. Observe that two instances of the cut rule are implicitly encoded by the only rule in $\mathcal{R}$. In this way, the following two distinct deductions are identified.

Example 28. Decoding if the inference rule of $\mathcal{CF}_{CYK}$.

$$\cfrac{\Gamma \to A \qquad \cfrac{\Delta \to B \quad A\,B \to C}{A\,\Delta \to C}}{\Gamma\Delta \to C} \qquad \cfrac{\Delta \to B \qquad \cfrac{\Gamma \to A \quad A\,B \to C}{\Gamma\, B \to C}}{\Gamma\Delta \to C}$$

Consider now the following example.

Example 29. Deduction in $\mathcal{CF}_{CYK}$.

Inputs: grammar G'_3 and string *Renzo loves a woman*

$$\cfrac{\cfrac{\mathit{Renzo}}{(0, NP, 1)} \quad \cfrac{\cfrac{\mathit{loves}}{(1, TV, 2)} \quad \cfrac{\cfrac{\mathit{a}}{(2, Det, 3)} \quad \cfrac{\mathit{woman}}{(3, N, 4)}}{(2, NP, 4)}}{(1, VP, 4)}}{(0, S, 4)}$$

It can be proved that the system $\mathcal{CF}_{CYK}$ for a CF grammar G recognizes items (i, C, j) such that $\vdash_G w_{i+1} \dots w_j \to C$.

3.3.1 Basic categorial grammars

From $\mathcal{CF}_{CYK}$ to the CYK parsing system for *AB* grammars the step is short. As before, we simplify the notation and we write an (*A*)$AB^{\otimes}$ grammar $G = \langle V_t, s, Lex, \langle \mathcal{F}, AX, R\rangle\rangle$ as $G = \langle V_t, s, \mathcal{F}, Lex, R\rangle$.

Definition 50. Let a *AB* grammar $G = \langle V_t, S, \mathcal{F}, Lex, AB\rangle$ and a string $w_1 \ldots w_n$ be given. The parsing system $\mathcal{AB}_{CYK} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R}\rangle$ is defined as follows:

$$\begin{array}{rcl} \mathcal{I} & = & \{ (i,a,j) \mid a \in \mathcal{F},\ 0 \leqslant i < j < n \} \\ \mathcal{A} & = & \{ (i-1,a,i) \mid w_i \to a \in Lex \} \\ \mathcal{R} & = & \left\{ \begin{array}{l} \dfrac{(i,c/a,k) \quad (k,a,j)}{(i,c,j)} \\ \\ \dfrac{(i,a,k) \quad (k,a\backslash c,j)}{(i,c,j)} \end{array} \right. \end{array}$$

Observe that the addition to the rule package of $\mathcal{AB}_{CYK}$ of the following rules gives the parsing system for associative *AB* grammars, which we call $\mathcal{AAB}_{CYK}$.

$$\frac{(i,c/a,k) \quad (k,a/b,j)}{(i,c/b,j)} \quad \frac{(i,b\backslash a,k) \quad (k,a\backslash c,j)}{(i,b\backslash c,j)}$$

A parsing system for *CCG*, which is an extension of $\mathcal{AAB}_{CYK}$, is presented in [Shieber et al., 1995].

As in the case of $\mathcal{CF}_{CYK}$, one can easily prove that the item (i,c,j) is generated by $(\mathcal{A})\mathcal{AB}_{CYK}$ if and only if $\vdash_G w_{i+1} \ldots w_j \to c$, where G is the (*A*)*AB* grammar of reference. In the next section we will prove this statement for the system $\mathcal{AB}^{\otimes}_{CYK}$.

3.3.2 Product rules

In the previous section, we considered *product free* basic categorial system. However, it is possible to extend the CYK deductive systems to the *AB* calculus with product, $AB^{\otimes}$. The product rule,

$$\frac{\Gamma \to a \quad \Delta \to b}{(\Gamma, \Delta) \to a \otimes b}$$

can be straightforwardly transformed in a correct parsing rule. For example, the following rule could be added to $\mathcal{AB}_{CYK}$ and produce a *correct* parsing system $\mathcal{AB}^{\otimes}_{CYK}$ for for grammars based on $AB^{\otimes}$.

(3.1)

$$\frac{(i,a,k) \quad (k,b,j)}{(i,a\otimes b,j)}$$

However, as it is, this rule is applicable to every two adjacent items, generating thus all possible product formulas having as immediate subformulas the formulas occurring in the premises. For example, given item (i,a,j), (j,b,k) and (k,c,l), we can have both the following deductions

$$\frac{\dfrac{(i,a,j) \quad (j,b,k)}{(i,a\otimes b,k)} \quad (k,c,l)}{(i,(a\otimes b)\otimes c,l)} \qquad \frac{(i,a,j) \quad \dfrac{(j,b,k) \quad (k,c,l)}{(j,b\otimes c,l)}}{(i,a\otimes(b\otimes c),l)}$$

Thus if we adopt rule 3.1, the parsing system for $AB^{\otimes}$ grammars will generate an exponential number of items. More precisely, the product rule in 3.1 gives rise to a so called Catalan explosion of the number of items generated.

Clearly, only a small subset of all the product items that can be generated by rule 3.1 are in fact needed in the derivation. Thus we should constrain rule 3.1 in such a way that an item of the form $(i,a\otimes b,k)$ is generated by rule 3.1 only if the formula $a\otimes b$ is needed in the deduction process. As $AB^{\otimes}$ grammars enjoy the *subformula property*[1], we can restrict the application of rule 3.1 to generate only items whose formulas belong to the set of subformulas of the axioms. Observe also that the problem of limiting the search space to subformulas of the input sequent does not arise for $\mathcal{AB}_{CYK}$. In this case the conclusion of each inference rule is a subformula of the premises.

In fact, not all subformulas of the axiom item are needed: we are interested only in the product formulas that can be generated by rule 3.1. We define the following set of formulas.

Definition 51. We define two functions, $\delta^+,\ \delta^- :: \mathcal{F} \to \{\mathcal{F}\}$, returning the set of product subformulas needed for the subformula test for the product parsing rule (we omit the symmetric cases).

$$\begin{array}{lcl}
\delta^+(a\otimes b) & = & \{a\otimes b\}\cup\delta^+(a)\cup\delta^+(b) \\
\delta^+(\_) & = & \emptyset \\
\delta^-(c/x) & = & \begin{cases} \delta^+(x)\cup\delta^-(c) & \text{if } x\equiv a\otimes b \\ \delta^-(c) & \text{otherwise.} \end{cases} \\
\delta^-(\_) & = & \emptyset
\end{array}$$

[1] The subformula property states that all formulas occurring in a deduction are subformulas of the formulas occurring in the conclusion sequent.

In order to clarify the role of the functions δ^+ and δ^- we generalize $AB^{\otimes}$ grammars to tuples $G = \langle V_t, E, \mathcal{F}, Lex, AB^{\otimes}\rangle$, where E, occurring in place of s, is a set of formulas: the output categories.

Definition 52. Let an $AB^{\otimes}$ grammar $G = \langle V_t, E, \mathcal{F}, Lex, AB^{\otimes}\rangle$, a string $w_1 \dots w_n$ and a formula $c \in E$ be given.

A set of formulas Σ is generated as follows:

$$\Sigma = \delta^+(c) \cup \{\, b \mid w_i \to a \in Lex,\ 1 \leqslant i \leqslant n,\ b \in \delta^-(a) \,\}$$

We define the parsing system $\mathcal{AB}^{\otimes}_{CYK} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R}\rangle$ as follows:

$$\begin{array}{rcl}
\mathcal{I} & = & \{\, (i, a, j) \mid a \in \mathcal{F},\ 0 \leqslant i < j \leqslant n \,\} \\[1ex]
\mathcal{A} & = & \{\, (i-1, a, i) \mid w_i \to a \in Lex \,\} \\[1ex]
\mathcal{R} & = & \left\{ \begin{array}{l}
\dfrac{(i, c/a, k) \quad (k, a, j)}{(i, c, j)} \\[2ex]
\dfrac{(i, a, k) \quad (k, a\backslash c, j)}{(i, c, j)} \\[2ex]
\dfrac{(i, a, k) \quad (k, b, j)}{(i, a \otimes b, j)} \ \text{if } a \otimes b \in \Sigma
\end{array} \right.
\end{array}$$

Observe that the set Σ contains all and only the formulas of the form $a \otimes b$ whose generation may require an instance of the product parsing rule. As we discussed, without this restriction the $\mathcal{AB}^{\otimes}_{CYK}$ parsing system would generate a number of items greater than the Catalan number of the length of the input string.

Example 30. Deduction in $\mathcal{AB}^{\otimes}_{CYK}$:

We consider the $AB^{\otimes}$ grammar for propositional logic, $\mathcal{PL}$, whose lexicon consists of the following entries:

$$\begin{array}{lcll}
p_i & \to & s & \quad 0 \leqslant i \\
\wedge & \to & (s\backslash s)/s & \\
\vee & \to & (s\backslash s)/s & \\
\neg & \to & s/s & \\
[& \to & s/(s \otimes c) & \\
] & \to & c &
\end{array}$$

Input string: $\neg[p_1 \vee p_2]$. Output category s. $\Sigma = \{s \otimes c\}$.

$$\dfrac{\dfrac{\neg}{(0,s/s,1)} \quad \dfrac{\dfrac{[}{(1,s/(s\otimes c),2)} \quad \dfrac{\dfrac{\dfrac{p_1}{(2,s,3)} \quad \dfrac{\dfrac{\vee}{(3,(s\backslash s)/s,4)} \quad \dfrac{p_2}{(4,s,5)}}{(3,s\backslash s,5)}}{(2,s,5)} \quad \dfrac{]}{(5,c,6)}}{(2,s\otimes c,6)}}{(1,s,6)}}{(0,s,6)}$$

We now prove the correctness of $\mathcal{AB}^{\otimes}_{CYK}$.

Correctness of $\mathcal{AB}^{\otimes}_{CYK}$

Proving correctness of a parsing system requires proving its *soundness* and *completeness*. Once a definition of the items generated by a parsing system $\mathcal{D}$ is given, soundness amounts to the proof that every item deduced according to the rules of D satisfies the definition. Instead, completeness requires that every item that conforms to the definition is generated through application of the rules of $\mathcal{D}$. Usually, the proof of soundness is easier, as it simply involves looking at the rules. [Sikkel, 1998] provides a general method for proving correctness of a parsing system. Such method results, in fact, in an abstraction of the traditional proof methods that can be found in [Aho and Ullman, 1972] and [Harrison, 1978], an abstraction made available by the deductive parsing approach.

Let us introduce some formal definitions.

Definition 53. Let $\mathcal{D} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ be a parsing system.

The set of *valid* items $V(\mathcal{D})$ is the set of items which can be deduced in any number of steps from hypotheses in $\mathcal{A}$ and, eventually, some further items.

The set of *viable items*, $\mathcal{W} \subseteq \mathcal{I}$, is the set of items that should be recognized by a parsing system $\mathcal{D}$.

Correctness of a parsing system is defined as follows.

Definition 54. A parsing system $\mathcal{D}$ is *correct*, if it is *sound* and *complete*. Formally, let $\mathcal{W}$ be a set of viable items of $\mathcal{D}$, then:

a) Soundness: $V(\mathcal{D}) \subseteq \mathcal{W}$.

b) Completeness: $\mathcal{W} \subseteq V(\mathcal{D})$.

c) Correctness: $\mathcal{W} = V(\mathcal{D})$.

The proof of soundness amounts to a proof the following statement.

Proposition 4. Let $\mathcal{D}$ be a parsing system and $\mathcal{W} \subseteq \mathcal{I}$.
If for all inference rules in $\mathcal{D}$,

$$\frac{\eta_1 \quad \dots \quad \eta_n}{\xi}$$

with $\eta_i \in \mathcal{A} \cup \mathcal{W}$, $1 \leqslant i \leqslant n$, it holds that $\xi \in \mathcal{W}$, then $V(\mathcal{D}) \subseteq \mathcal{W}$.

For the specific case of $\mathcal{AB}^{\otimes}_{CYK}$, we define the set of viable items as follows.

Definition 55. For an $\mathcal{AB}^{\otimes}_{CYK}$ system, for an $AB^{\otimes}$ grammar G and a string $w_1 \dots w_n$, we define the set of *viable* items $\mathcal{W}$ as follows

$$\mathcal{W} = \{\ (i, a, j) \mid w_{i+1} \dots w_i \Rightarrow^* a\ \}$$

We proceed to prove the soundness of $\mathcal{AB}^{\otimes}_{CYK}$. We use lowercase Greek letters as variables over items.

Proposition 5. *Soundess* of $\mathcal{AB}^{\otimes}_{CYK}$.
$V(\mathcal{AB}^{\otimes}_{CYK}) \subseteq \mathcal{W}$

Proof.
If $\xi \in \mathcal{A}$, then $\xi \equiv (i-1, a, i)$ and $w_i \rightarrow a \in Lex$. Thus $\xi \in \mathcal{W}$.

If ξ is deduced by cancellation from items $(i, a/b, k)$ and (k, b, j), then $\xi \equiv (i, a, j)$. By IH $(i, a/b, k) \in \mathcal{W}$ and $(k, b, j) \in \mathcal{W}$. Thus $w_{i+1} \dots w_k \Rightarrow^* a/b$ and $w_{k+1} \dots w_j \Rightarrow^* b$. We conclude that $w_{i+1} \dots w_k w_{k+1} \dots w_j \Rightarrow^* a$. Hence $\xi \in \mathcal{W}$.

If ξ is deduced by product rule from items (i, a, k) and (k, b, j), then $\xi \equiv (i, a \otimes b, j)$. By IH we have $w_{i+1} \dots w_k \Rightarrow^* a$ and $w_{k+1} \dots w_j \Rightarrow^* b$. We conclude that $w_{i+1} \dots w_k w_{k+1} \dots w_j \Rightarrow^* a \otimes b$. Hence $\xi \in \mathcal{W}$. □

In order to prove completeness of a parsing system, [Sikkel, 1998]defines a *deduction length function* on the set of viable items $\mathcal{W}$. A similar notion is the *rank* of [Aho and Ullman, 1972], used also in [Shieber et al., 1995] for proving completeness of deductive parsers.

Definition 56. Deduction length function, *dfl*.

Let $\mathcal{D}$ be a parsing system and $\mathcal{W} \subseteq \mathcal{I}$ a set of items. A function $d ::$ $(\mathcal{A} \cup \mathcal{W}) \rightarrow Int$ is a deduction length function iff

1. $d(h) = 0$, if $h \in \mathcal{A}$

2. for each $\xi \in \mathcal{W}$ there is an inference rule in $\mathcal{D}$

$$\frac{\eta_1 \quad \dots \quad \eta_n}{\xi}$$

   such that $\{\eta_1, \dots, \eta_n\} \subseteq \mathcal{W}$ and $d(\eta_i) < d(\xi)$ for $0 \leqslant i \leqslant n$.

The *dfl* allows to prove completeness by induction on $d(\xi)$.

Proposition 6. Let $\mathcal{D}$ be a parsing system and $\mathcal{W} \subseteq \mathcal{I}$.
If a *dfl* d exists, then $\mathcal{W} \subseteq V(\mathcal{D})$.

Completeness of $\mathcal{AB}^{\otimes}_{CYK}$ is proved as follows. From the assumption that items η, with $d(\eta) < m$, are valid, it has to be proven that all ξ with $d(\xi) = m$ are valid.

Proposition 7. *Completeness* of $\mathcal{AB}^{\otimes}_{CYK}$.
$\mathcal{W} \subseteq V(\mathcal{AB}^{\otimes}_{CYK})$

Proof. One defines a function d such that

$$\begin{aligned} d(i-1, a, i) &= 0 \\ d(i, a, j) &= j - i \end{aligned}$$

Clearly, d is a deduction length function. □

3.4 Earley style parsing

In the previous section, we examined the CYK deductive system for $\mathcal{CF}$ grammars and *AB* grammars with and without product. The CYK system tries to construct a deduction of the input string starting from the preterminal categories, assigned to the words in the string. A category is assigned to a larger portion of the input on the basis of the categories assigned to the premises and of rules of the grammar. It is called *bottom-up* because it proceeds from the premises to the conclusion.

Although very simple and elegant, the CYK system has some *limitations*. Firstly, in the case of $\mathcal{CF}$ grammars, we shall assume that the input grammar is in *Chomsky normal form*[2]. Secondly, the input grammar should not produce the *empty string*.

[2] In fact, there are also *generalized* variants of the CYK system.

In this section, we discuss another kind of deductive parser which is not affected by the previous limitations: the *Earley parser*, from the name of its inventor Jay Earley. It works with any $\mathcal{CF}$ grammar, and it is faster than the CYK algorithm, at least if the underlying grammar is not ambiguous. The algorithm works partly top-down (in the so called predictive phase) and partly bottom-up (in the completion phase). Nonetheless, it is often considered a *top-down* parser as it tries to construct a derivation from the root towards the leaves.

3.4.1 Earley system for $\mathcal{CF}$

The Earley algorithm was presented in Jay Earley's doctoral dissertation [Earley, 1968], see also [Earley, 1970]. As we said before it works with any $\mathcal{CF}$ grammar and in some case it is more efficient than the CYK parser.

Definition 57. Earley's deductive parser.

Let a $\mathcal{CF}$ grammar $G = \langle V_t, S, \mathcal{F}, AX\rangle$ and a string $w_1 \dots w_n$. The symbol S' is a new start symbol not in $\mathcal{F}$.

We define the parsing system $\mathcal{CF}_{Earley} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R}\rangle$ as follows:

$$
\begin{array}{rcl}
\mathcal{I} & = & \{\ (i, \Gamma \bullet \Delta \to C, j) \mid \Gamma\Delta \to C \in AX,\ 0 \leqslant i \leqslant j \leqslant n\ \} \\[1em]
\mathcal{A} & = & \{\ (i-1, w_i, i) \mid 1 \leqslant i \leqslant n\ \} \\[1em]
\mathcal{R} & = & \left\{
\begin{array}{lr}
(0, \bullet\Gamma \to S, 0) \quad \text{for all } \Gamma \to S \in AX & \textit{Init} \\[1em]
\dfrac{(i, \Delta \bullet w\Gamma \to C, j) \quad (j, w, j+1)}{(i, \Delta\, w \bullet \Gamma \to C, j+1)} & \textit{Scan} \\[2em]
\dfrac{(i, \Delta \bullet A\Gamma \to C, j)}{(j, \bullet\Lambda \to A, j)}\ \Lambda \to A \in AX & \textit{Predict} \\[2em]
\dfrac{(k, \Lambda\bullet \to A, j) \quad (i, \Delta \bullet A\Gamma \to C, k)}{(i, \Delta\, A \bullet \Gamma \to C, j)} & \textit{Complete}
\end{array}
\right.
\end{array}
$$

The system $\mathcal{CF}_{Earley}$ is known to be correct. A simple proof can be found in [Sikkel, 1998]. The set of viable items $\mathcal{W}$ is defined as follows:

$$
\begin{array}{rl}
\mathcal{W} = \{\ (i, \Delta \bullet \Lambda \to A, j) \mid & w_{i+1} \dots w_j \Rightarrow^* \Delta, \\
& w_1 \dots w_i\, A\, \Gamma \Rightarrow^* S \text{ for some } \Gamma \in (V_t \cup \mathcal{F})^*\ \}
\end{array}
$$

Let us examine an example deduction in $\mathcal{CF}_{Earley}$.

Example 31. Deduction of [[]] in grammar $[S]S \mid \epsilon \to S$.

Items generated:

1. $(0, \bullet[S]S \to S, 0)$: *Init*
2. $(0, \bullet \to S, 0)$: *Init*
3. $(0, [\bullet S]S \to S, 1)$: *Scan* 1
4. $(1, \bullet[S]S \to S, 1)$: *Predict* 3
5. $(0, [S\bullet]S \to S, 1)$: *Complete* 2-3
6. $(1, [\bullet S]S \to S, 2)$: *Scan* 4
7. $(2, \bullet[S]S \to S, 2)$: *Predict* 6
8. $(2, \bullet \to S, 2)$: *Predict* 6
9. $(1, [S\bullet]S \to S, 2)$: *Complete* 8-6
10. $(1, [S] \bullet S \to S, 3)$: *Scan* 9
11. $(3, \bullet[S]S \to S, 3)$: *Predict* 10
12. $(3, \bullet \to S, 3)$: *Predict* 10
13. $(1, [S]S\bullet \to S, 3)$: *Complete* 12-10
14. $(0, [S\bullet]S \to S, 3)$: *Complete* 13-3
15. $(0, [S] \bullet S \to S, 4)$: *Scan* 14
16. $(4, \bullet[S]S \to S, 4)$: *Predict* 15
17. $(4, \bullet \to S, 4)$: *Predict* 15
18. $(0, [S]S\bullet \to S, 4)$: *Complete* 17-15

The deduction can be shown as a tree[3]. We present part of the previous deduction in tree format:

$$\dfrac{(4,\bullet\to S,4) \qquad \dfrac{\dfrac{\begin{matrix}\vdots\\ (1,[S]S\bullet\to S,3)\end{matrix} \qquad \dfrac{(0,\bullet[S]S\to S,0) \quad (0,[,1)}{(0,[\bullet S]S\to S,1)}}{(0,[S\bullet]S\to S,3)} \qquad (3,],4)}{(0,[S]\bullet S\to S,4)}}{(0,[S]S\bullet\to S,4)}$$

Soundness of $\mathcal{CF}_{Earley}$ is, as usual, trivial. In order to prove *completeness*, one defines a deduction length function $d :: (\mathcal{A}\cup\mathcal{W})\to Int$ by

$$\begin{aligned} d(i,\Delta\bullet\Lambda\to A,j) \;=\; min\{\, \delta+2\gamma+2\mu+j \mid\ & w_{i+1}\ldots w_j \Rightarrow^{\gamma} \Delta, \\ & \Gamma A\Gamma' \Rightarrow^{\delta} S, \\ & w_1\ldots w_i \Rightarrow^{\mu} \Gamma\ \} \end{aligned}$$

One should check that d satisfies condition 2 of definition 56 for every item in $\mathcal{W}$. We refer to [Sikkel, 1998] for the details of the proof. In the next section, we will see the Earley deductive system for $AB^{\otimes}$ grammars.

3.5 Implementations

The labeled inference rules of the parsing systems formulated in the previous chapter specify what conclusion follows from what premises. However, they do not specify any order in which these rules should be applied nor how the inference-drawing process should be iterated. In this chapter, we present some implementations chart parsers. The first is the *agenda-driven, chart-based deductive procedure* of [Shieber et al., 1995]: we provide a functional implementation of this procedure. The second is based on the description of the CYK algorithm of [Aho and Ullman, 1972]. This implementation is in fact the most efficient as it results in a $O(n^3)$ time complexity recognition algorithm. Furthermore, the *table* resulting from this recognition procedure, called *parse table*, can be easily used to extract deduction trees. We will present algorithms for building, from the parse table, parse trees similar to those given in the examples of the previous chapter.

[3] In fact, in order to have such a representation, we shall assume that predicted items are leaves of the proof tree (the place of hypotheses), although they are in fact derived. A graph would be a better representation of Earley deductions, but for simplicity we stick to the the tree formalism.

3.6 Agenda-driven, chart-based procedure

An agenda-driven chart-based procedure, $\mathscr{AC}$ procedure for short, operates on two sets of items called the *agenda* and the *chart*. The agenda contains all items whose consequences are still to be computed. The chart the items whose consequences have already been computed.

Algorithm 1 is an adaptation of the $\mathscr{AC}$ procedure specified in [Shieber et al., 1995].

Algorithm 1. *Agenda-driven, chart-based* deduction procedure.

1. Initialize the chart to the empty set of items and the agenda to the axioms of the deduction system.

2. Repeat the following steps until the agenda is exhausted:

   (a) Select an item from the agenda, called the *trigger item*, and remove it.

   (b) Add the trigger item to the chart, if necessary.

   (c) If the trigger item was added to the chart, generate all items that are new immediate consequences of the trigger item together with all items in the chart, and add these generated items to the agenda.

3. If a *goal item* is in the chart, the goal is proved (and the string recognized); otherwise it is not.

A *goal item* is an item that coves the whole input string with the start category, for example $(0, s, n)$ in $\mathcal{AB}^{\otimes}_{CYK}$ or $(0, s\bullet \to S', n)$ in $\mathcal{CF}_{Earley}$, where n is the length of the input string. The proviso "if necessary" in 2b means "if not already present", and in 2c "new" means "not already present". In [Shieber et al., 1995], one may find a detailed discussion of the problem of redundant items as well as a proof of soundness and completeness of algorithm 1.

Functional implementation

Algorithm 2 below is a functional implementation of the $\mathscr{AC}$ procedure specified in algorithm 1 for categorial grammars. This implementation is similar to the one in [van Eijck, 2004] who provides a functional implementation of $\mathcal{CF}_{Earley}$. However, it differs in that we directly work with *sets*, rather than with lists from which duplicates are removed. Sets can be implemented in the functional setting as *red-black* trees, see [Okasaki, 1998, 1999], and also [Adams, 1993]. In short, red-black trees represent sets as trees whose nodes are *ordered*. For instance, what we write as $\{a, b, c, d, e, f, g\}$ is interpreted as the tree:

$$\frac{\dfrac{a \quad c}{b} \quad \dfrac{e \quad g'}{f}}{d}$$

Red and *black* are node labels which enforce further invariants and allow to implement set-theoretic operations efficiently, see [Cormen et al., 1990]. We assume that a data-type for sets of objects of type a, which we write $\{a\}$, has been defined. We write $X[x]$ for a set X with a distinguished element x, so that X results from $X[x]$ by removing the element x (we may, for example, assume that x is the least element of the red-black tree). We use the conventional notation for operations on sets.

Algorithm 2. Haskell implementation of the $\mathscr{AC}$ procedure in algorithm 1.

Let a parsing system $\mathcal{D} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ be given.

Let `type` *Chart* = $\{\mathcal{I}\}$. Initial value of the chart variable $Z = \emptyset$.

Let `type` *Agenda* = $\{\mathcal{I}\}$. Initial value of the agenda variable $Y = \mathcal{A}$.

```
- exhaust-agenda :: $(Chart, Agenda) \longrightarrow (Chart, Agenda)$
    exhaust-agenda  $(Z, \emptyset)$     $= (Z, \emptyset)$
    exhaust-agenda  $(Z, Y[y]) =$
                    if $y \in Z$
                    then exhaust-agenda $(Z, Y)$
                    else exhaust-agenda $(Z', Y')$
                      where
                a)        $C = \{ c \mid z \in Z,\ \rho \in \mathcal{R},\ c \in \rho\ y\ z \}$
                b)        $Y' = Y \cup C$
                c)        $Z' = \{y\} \cup Z$
```

Step by step analysis: `exhaust-agenda` calculates all the valid items of a given parsing system. The constructs `type` $x = y$ declare a type variable x to be of the form y. Thus *Chart* and *Agenda* are type variables for sets of items: we initialize these variable to the empty set and to the set of axioms of the parsing system, respectively. The derived items are stored in the chart variable Z at the end of the computation, that is when the agenda variable Y is empty. At each iteration, the recursive call of `exhaust-agenda` tests if an item y taken from the agenda is already in the chart Z. If this is the case, its immediate consequences have already been computed, and we can proceed to the next item in the agenda. Otherwise, we calculate in the set C the immediate consequences of y with every $z \in Z$ on the basis of the rules in $\mathcal{R}$, in line a). In line b), we build the new agenda as the union of the old agenda, minus the trigger item$y$, with the immediate consequences of y. Then the trigger y is added to the chart. The process is iterated with the new cart and new agenda until the agenda is empty.

What still remains to be done is to appropriately implement the rules of the parsing system. Let us consider the implementation of $\mathcal{AB}^{\otimes}_{CYK}$.

Example 32. Implementation of rules for $\mathcal{AB}^{\otimes}_{CYK} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$.

Let a set of formulas Σ be calculated from the axioms according to definition 51 in chapter 3. Then $\mathcal{R}$ consists of rues `el` and `p` Σ defined below.

$\mathtt{p} :: \{\mathcal{F}\} \to \mathcal{I} \to \mathcal{I} \to \{\mathcal{I}\}$

$$\mathtt{p}\ \Sigma\ (i,a,k)\ (k',b,j) \quad = \quad \text{if } k \equiv k' \ \&\ a \otimes b \in \Sigma \text{ then } \{(i, a \otimes b, j)\} \text{ else } \emptyset$$

$\mathtt{el}, \mathtt{e}_0, \mathtt{e}_1 :: \mathcal{I} \to \mathcal{I} \to \{\mathcal{I}\}$

$$\begin{array}{lcl}
\mathtt{el}\ x\ y & = & \mathtt{e}_0\ x\ y \cup \mathtt{e}_1\ x\ y \\
\\
\mathtt{e}_0\ (i,b,k)\ (k',b'\backslash a,j) & = & \text{if } k \equiv k' \ \&\ b \equiv b' \text{ then } \{(i,a,j)\} \text{ else } \emptyset \\
\mathtt{e}_0\ (i,a/b,k)\ (k',b',j) & = & \text{if } k \equiv k' \ \&\ b \equiv b' \text{ then } \{(i,a,j)\} \text{ else } \emptyset \\
\mathtt{e}_0\ \_\ \_ & = & \emptyset \\
\\
\mathtt{e}_1\ (k',b'\backslash a,j)\ (i,b,k) & = & \text{if } k \equiv k' \ \&\ b \equiv b' \text{ then } \{(i,a,j)\} \text{ else } \emptyset \\
\mathtt{e}_1\ (k',b',j)\ (i,a/b,k) & = & \text{if } k \equiv k' \ \&\ b \equiv b' \text{ then } \{(i,a,j)\} \text{ else } \emptyset \\
\mathtt{e}_1\ \_\ \_ & = & \emptyset
\end{array}$$

Observe that $\mathtt{e}_0$ and $\mathtt{e}_1$ exhaust the possible occurrences of patterns for their arguments, hence `el` is a complete formulation of the cancellation rules of $\mathcal{AB}^{\otimes}_{CYK}$.

In order to apply algorithm 2 to the $\mathcal{CF}_{CYK}$ system, one should store the set of productions of the input *CF* grammar in the variable W in the `exhaust-agenda` procedure. Then the transitions may look like the following.

Example 33. Implementation of rules for $\mathcal{CF}_{CYK} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$.

Let AX be the set of the productions of the input grammar. Then $\mathcal{R}$ consists of `cfInf` AX where:

$$\texttt{cfInf} :: AX \rightarrow \mathcal{I} \rightarrow \mathcal{I} \rightarrow \{\mathcal{I}\}$$

$$\begin{aligned}\texttt{cfInf} \quad AX \quad (i,a,k) \quad (l,b,j) \; = \\ \{\, (i,c,j) \mid l \equiv k,\; (a'\; b' \rightarrow c) \in AX,\; a \equiv a',\; b \equiv b' \,\} \\ \cup \\ \{\, (l,c,k) \mid j \equiv i,\; (b'\; a' \rightarrow c) \in AX,\; a \equiv a',\; b \equiv b' \,\}\end{aligned}$$

3.7 Tabular parsing

One can easily check that the complexity of algorithm 2 is $O(n^5)$, for an input string of length n. However, a well known fact about the CYK and Earley's approaches is that they can run in cubic time. In this section, we illustrate the more efficient implementations of these parsers presented in [Aho and Ullman, 1972]. The main limitation of algorithm 2 is that at each iteration, we check whether the trigger item had already been computed.

The algorithms that we are going to see avoid this problem by making use of more refined data structures which allow a more efficient bookkeeping strategy. They are called *tabular* because their method is based on the construction of a table, called *parse table*, or of a similar data structure. The table is a database that stores *systematically* the partial analyses obtained at a given point of the computation. Thus these analyses can be efficiently retrieved from the table and used for building larger analyses, which in turn are stored in the appropriate place in the table.

For example in the case of the CYK parser, if we are analyzing a string $w_1 \ldots w_n$ in a grammar G, then the table T may consist of cells, denoted $t_{(j,i)}$, with $0 \leqslant j < n$ and $0 < i \leqslant n$. The cells contain formulas and we have that $c \in t_{(j,i)}$ if and only if $w_{j+1} \ldots w_i \Rightarrow^* c$. Therefore, to test whether $w_1 \ldots w_n$ belongs to $L_t(G)$ according to the CYK algorithm, we compute the parse table for $w_1 \ldots w_n$ and check whether the start symbol of G is in $t_{(0,n)}$.

3.7.1 Tabular CYK

We present the CYK method of table construction. The algorithm works for $\mathcal{CF}$ grammars in *CNF* without ϵ-productions or basic categorial grammars (with and without product) without assignments for the empty string. We call these grammars *lexicalized* grammars. The symbol $\bigtriangledown$ is a variable over binary operations on sets of formulas. After the definition of the table construction method we will instantiate ∇ with the specific operations proper to a *CF* or a basic categorial grammar. Similarly, *Start* is a variable over the start symbol of the input grammar.

Algorithm 3.

Input: A lexicalized grammar $G = \langle V_t, Start, Lex, \mathcal{F}, AX\rangle$ without ϵ-assignments and a string $w_1 \dots w_n$.

Output: A parse table T for $w_1 \dots w_n$ such that $t_{(j,i)}$ contains c if and only if $w_{j+1} \dots w_i \rightarrow^* c$.

Method:

1. **Initialization**: for all $m,\ 0 < m \leqslant n$, set

$$t_{(m-1,m)} = \{\, a \mid w_m \rightarrow a \in Lex \,\}$$

2. **Table completion**:
   (a) Set $i = 1$.
   (b) Test if $i = n$. If not, increment i by 1 and perform **line** (i) at point 3 below.
   (c) Repeat step (2b) until $i = n$.

3. **line** (i):
   (a) Let $j = i - 2$.
   (b) Let $k = j + 1$.
   (c) Let $t_{(j,i)} = t_{(j,i)} \cup (t_{(j,k)} \nabla t_{(k,i)})$
   (d) Increment k by 1.
   (e) If $k = i$, then go to step (3f). Else, go to step (3c).
   (f) If $j = 0$, then halt. Else, decrease j by 1 and go to step (3b).

The algorithm recognizes a string $w_1 \dots w_n$ if and only if $Start \in t_{(0,n)}$.

Definition 58. Instantiation of ∇ for *CF* grammars.

If the grammar input of algorithm 3 is a *CF* grammar G, then $\nabla = \circledast_{AX}$, where AX are the productions of G and $(\circledast_{AX}) :: \{\mathcal{F}\} \rightarrow \{\mathcal{F}\} \rightarrow \{\mathcal{F}\}$ is defined as follows:

$$X \circledast_{AX} Y = \{\, A \mid B \in X,\ C \in Y,\ B\ C \rightarrow A \in AX \,\}$$

We call $\mathscr{CYK}_{CF}$ the algorithm resulting from algorithm 3 by instantiating ∇ with $\circledast_{AX}$.

Example 34. We examine algorithm 3 applied to the string *every man loves a woman* and to grammar G_3' in example 26.

Table generated:

1	2	3	4	5	$^{i}{}_{j}$
Det	*NP*			*S*	0
	N				1
		TV		*VP*	2
			Det	*NP*	3
				N	4

The CYK parsing algorithm is also called *chart* parsing for another graphical representation of the deduction to which it gives rise. The chart for example 34 is presented in figure 3.1. The vertices of the graph indicate the positions of the words in the input string. The edge label indicate the grammatical category of the subexpression between the vertices linked by the edge.

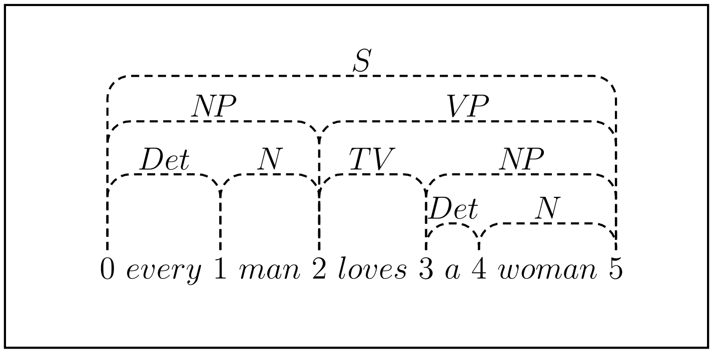


Figure 3.1: Chart

Example 35. Let us consider now a second example from grammar $G_{1'}$ which consists of the following rewriting rules.

$$\begin{array}{ll} O\,S' \mid S\,S \mid O\,C & \rightarrow S \\ S\,C & \rightarrow S' \\ (& \rightarrow O \\ ) & \rightarrow C \end{array}$$

This grammar generates the language of non-empty balanced brackets. Furthermore this grammar is in Chomsky normal form.

Algorithm 3 applied to string (()())() and grammar $G_{1'}$.

1	2	3	4	5	6	7	8	$^{i}{}_{j}$
O					S		S	0
	O	S		S	S'			1
		C						2
			O	S				3
				C				4
					C			5
						O	S	6
							C	7

The correctness of algorithm 3 is known and we refer the reader to [Aho and Ullman, 1972] for the proof soundness and completeness. Below we will see this proofs for $AB^{\otimes}$.

We now define the $\bigtriangledown$ operation for basic categorial grammars.

Definition 59. Instantiation of $\bigtriangledown$ for $AB^{\otimes}$ grammars.

If the grammar input of algorithm 3 is an $AB^{\otimes}$ grammar G, then $\bigtriangledown = \divideontimes_{\Sigma}$, where Σ is a set of formulas obtained from the lexical categories assigned to the input string by function δ^{-} in definition 51 and $(\divideontimes_{\Sigma}) :: \{\mathcal{F}\} \to \{\mathcal{F}\} \to \{\mathcal{F}\}$ is defined as follows:

$$
\begin{array}{rcc}
X \divideontimes_{\Sigma} Y & = & \{\, c \mid c/b \in X,\ b \in Y \,\} \\
 & & \cup \\
 & & \{\, c \mid b \in X,\ b\backslash c \in Y \,\} \\
 & & \cup \\
 & & \{\, a \otimes b \mid a \in X,\ b \in Y,\ a \otimes b \in \Sigma \,\}
\end{array}
$$

We call $\mathscr{CYK}_{AB^{\otimes}}$ the algorithm resulting from algorithm 3 by instantiating $\bigtriangledown$ with $\divideontimes_{\Sigma}$.

Example 36. Application of $\mathscr{CYK}_{AB^{\otimes}}$ to the string *aacbb* and to grammar A_0.

Table

1	2	3	4	5	$^{i}{}_{j}$
$(s/b)/s$			s/b	s	0
	$(s/b)/s$	s/b	s		1
		s			2
			b		3
				b	4

Chart:

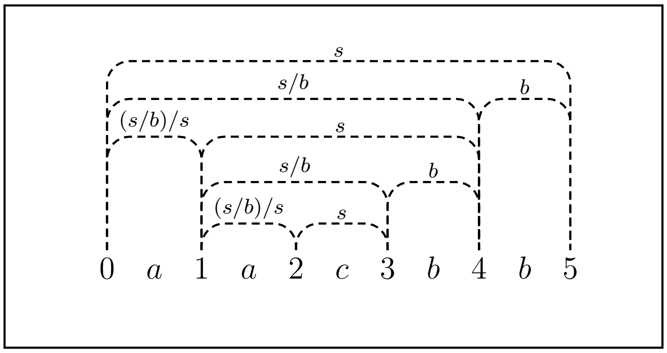


Figure 3.2: Chart of $\mathscr{CYK}_{AB^{\otimes}}$ applied to the string *aacbb* and to grammar A_0

Proposition 8. If algorithm 3 is applied to a lexicalized grammar G and to a string $w_1 \ldots w_n$, then upon termination,

$$c \in t_{(j,i)} \quad \text{iff} \quad \vdash_G w_{j+1} \ldots w_i \to c$$

Proof. Induction on i. We consider the a few cases for $AB^{\otimes}$ grammars. Let Σ be the set of subformulas obtained in accordance with definition 59.

[*only if*]: If $i = j+1$ and $c \in t_{(j,i)}$, then $w_i \to c \in Lex$. If $i > j+1$, then for $k,\ j < k < i$ we have three cases:

1. $a \in t_{(j,k)}$ and $b \in t_{(k,i)}$, $c \equiv a \otimes b$ and $c \in \Sigma$,
2. $c/b \in t_{(j,k)}$ and $b \in t_{(k,i)}$,
3. $b \in t_{(j,k)}$ and $b\backslash c \in t_{(k,i)}$.

In all cases, we conclude $\vdash_G w_{j+1} \ldots w_k w_{k+1} \ldots w_i \to c$ by IH and one rule application.

[*if*]: If $i = j+1$, then $w_i \to c \in Lex$. Then $c \in t_{(j,i)}$ by step 1. If $i > j+1$, then for $k,\ j < k < i$ we have three cases:

1. $c \equiv a \otimes b$, $c \in \Sigma$ and $\vdash_G w_{j+1} \ldots w_k \to a$ and $\vdash_G w_{k+1} \ldots w_i \to b$,
2. for some formula b, $\vdash_G w_{j+1} \ldots w_k \to c/b$ and $\vdash_G w_{k+1} \ldots w_i \to b$,
3. for some formula b, $\vdash_G w_{j+1} \ldots w_k \to b$ and $\vdash_G w_{k+1} \ldots w_i \to b\backslash c$.

In each case, by IH, the succedent formula is in the cell identified by the antecedent index extension. Hence $c \in t_{(j,i)}$ in step 3c by rule $\divideontimes_{\Sigma}$ from definition 59.

□

The complexity of the CYK algorithm is calculated in terms of the *elementary operations* required to build the table. According to the terminology of [Aho and Ullman, 1972] we have the following kinds of elementary operations.

1. Setting a variable to a constant, to the value held by some variable, or to the sum or difference of the value of two variables or constants.
2. Testing if two variables are equal.
3. Examining and/or altering the value of j, i, if j and i are the current values of two integer variables of constants.
4. Examining w_i, the ith input symbol, if i is the value of some variable.

Proposition 9. Let n be the length of the input string. Then algorithm 3 requires $O(n^3)$ elementary operations to compute $t_{(j,i)}$ for all i and j.

Proof. See [Aho and Ullman, 1972]. □

The argument is based on the fact that the algorithm consists of three embedded cycles. The most external one computes **line** (i) n times. The instructions in **line** (i), in turn, consist of two embedded cycles, one for j, from step 3a to step 3f, and one for k, from step 3b to step 3c. Both j and k range between 0 and n. Thus **line** (i) takes $O(n^2)$ elementary operation, and therefore algorithm 3 requires $O(n^3)$ elementary operations.

Remark 1. The complexity result expressed in proposition 9 concerns the variation of time in relation to the length of the input string. Another parameter which may become relevant in the calculation of the complexity of the CYK algorithm is the size of the input grammar, expressed in terms of the number of its axioms: $|G| = O(|AX|)$. Thus, the complexity of 3 can be expressed as $O(|G|n^3)$, see [Nederhof and Satta, 2004]. It should be remarked that tn the case of $\mathcal{CF}$ grammars, conversion to $\mathcal{CNF}$ may square the size of the original grammar. As $|G|$ is usually much bigger than the length of the input string, squaring it may affect drastically the performance of the CYK algorithm. However, this remark does not apply to the CYK algorithm for $AB^{\otimes}$ grammars: since these grammars have only binary rules, they are already in the required normal form.

The CYK algorithm for *AB* grammars can easily be extended to associative *AB* grammars, *AAB*, by simply adding the associative cancellation rules. One extends the operation $\circledast$ with the following sets

$$\begin{gathered}\{\, c/a \mid c/b \in X,\ b/a \in Y \,\}\\ \cup\\ \{\, a\backslash c \mid a\backslash b \in X,\ b\backslash c \in Y \,\}\end{gathered}$$

Observe that with these new rules the outputs a/c and $c\backslash a$ of the associative composition rules may not belong themselves to the set of subformulas of formulas of the grammar. This aspect may increase the complexity of the algorithm. However, [Vijay-Shanker and Weir, 1990] prove that the even a more general variant of the *AAB* system[4] can be parsed in time $O(n^6)$.

3.8 Parses

Although we often spoke of 'parsing' algorithmsin the preceding sections, what we described and implemented were, technically, *recognition* algorithms: procedures to determine whether the given string belongs to the language generated by the given grammar.

Technically, a *parser* returns the tree structure(s) assigned by a grammar to a string. A distinction can be drawn between a parser that returns *one* structure, if any, and a parser that returns *all* the tree structures assigned by a grammar to a string.

In the following sections, we will see that the problems of recognition and parsing are closely related. Indeed, we saw already that the output of the recognition algorithms is a *parse* table. Thus parsing, in its existential or universal meaning, is the process of retrieving from such a table one or all the structural descriptions of the input. The extraction of parse trees will be based on the technique of *parse forest*.

3.9 Parse forest

The extraction of structural descriptions from a parse table may become easy and fast if we have a concise way of representing all possible parses

[4] Such variant admits generalized associative composition rules that abstract on the number of the arguments of the categories and partially on the orientation of the slashes. We refer the reader to [Steedman, 2000b] for a discussion of such system and its complexity properties.

from a given output table. *Parse forests*, originally introduced in [Bar-Hillel et al., 1964b] and also called *shared forests* in [Billot and Lang, 1989], are a way of encoding all parse trees for a given input in a compact way. We shall remark that a string may be assigned an exponential number of structural descriptions (exponential on its length) or even infinite in case the input grammar is cyclic. Thus parse forest are a valuable tool for encoding and retrieving parses.

Algorithm 4. Let $G = \langle V_t, S, \mathcal{F}, Lex, AX\rangle$ be a *CNF* grammar without ϵ-productions. Let a string $ws = w_1 \dots w_n$ and the parse table T resulting by application of algorithm $\mathscr{CYK}_{\mathit{CF}}$ to G and ws be given. The parse forest $G_w = \langle V'_t, (0, S, n), \mathcal{F}_w, Lex_w, AX_w\rangle$ resulting from G, ws and T is constructed as follows.

$$\begin{array}{lcr}
V'_t & = & \{\ (i-1, w_i, i) \mid w_i \in V_t\ \} \\
\mathcal{F}_w & = & \{\ (j, A, i) \mid A \in \mathcal{F},\ 0 \leqslant j < i \leqslant n\ \} \\
Lex_w & = & \{\ (i-1, w_i, i) \to (i-1, A, i) \mid w_i \to A \in Lex\ \} \\
AX_w & = & \{\ (j, A, k)\ (k, B, i) \to (j, C, i) \mid A\ B \to C \in AX, \\
 & & A \in t_{(j,k)},\ B \in t_{(k,i)}\ \}
\end{array}$$

Example 37. Consider the grammar G_5 with the following production rules:

$$\begin{array}{l}
S\ PP \mid NP\ VP \to S \\
V\ NP \to VP \\
Det\ N \mid NP\ PP \mid John \mid Mary \to NP \\
Pr\ NP \to PP \\
telescope \to N \\
the \to Det \\
with \to Pr \\
saw \to V
\end{array}$$

From the parse table T returned by algorithm $\mathscr{CYK}_{\mathit{CF}}$ applied to G_5 and the sentence *Mary saw John with a telescope*, we can construct the parse forest G_{w_5} whose productions are below.

$$\begin{array}{ll}
(0,S,3)\ (3,PP,6) & \rightarrow (0,S,6) \\
(0,NP,1)\ (1,VP,6) & \rightarrow (0,S,6) \\
(0,NP,1)\ (1,VP,3) & \rightarrow (0,S,3) \\
(1,V,2)\ (2,NP,6) & \rightarrow (1,VP,6) \\
(1,V,2)\ (2,NP,3) & \rightarrow (1,VP,3) \\
(4,Det,5)\ (5,N,6) & \rightarrow (4,NP,6) \\
(2,NP,3)\ (3,PP,6) & \rightarrow (2,NP,6) \\
(3,Pr,4)\ (4,NP,6) & \rightarrow (3,PP,6) \\
(5,telescope,6) & \rightarrow (5,N,6) \\
(4,the,5) & \rightarrow (4,Det,5) \\
(3,with,4) & \rightarrow (3,Pr,4) \\
(2,John,3) & \rightarrow (2,NP,3) \\
(1,saw,2) & \rightarrow (1,V,2) \\
(0,Mary,1) & \rightarrow (0,NP,1)
\end{array}$$

From the parse forest we can easily recover the parse trees with the following procedure which consists in following the rewrite rules of the parse forest from the root to the leaves.

Algorithm 5. Let $G_w = \langle V_t', (0,S,n), \mathcal{F}_w, Lex_w, AX_w \rangle$ be a parse forest. Let $\mathsf{gen}^* :: \mathcal{F}_w \rightarrow G_w \rightarrow \{Tree\ \mathcal{F}_w\}$ be a function taking in input a non-terminal in $\mathcal{F}_w$, a parse forest G_w and returning a set of trees.

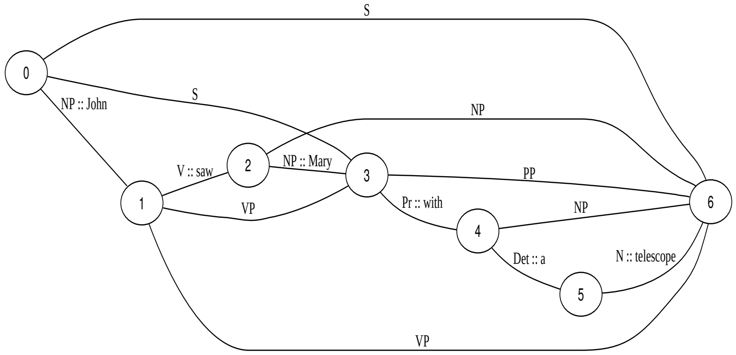
$$\begin{array}{lcl}
\mathsf{gen}^*\ C\ G_w & = & \{\ Branch\ C\ l\ r \mid A\ B \rightarrow C \in AX_w, \\
 & & \qquad\qquad\quad l \in \mathsf{gen}^*\ A\ G_w, \\
 & & \qquad\qquad\quad r \in \mathsf{gen}^*\ B\ G_w\ \} \\
 & & \quad \cup \\
 & & \quad \{\ Leaf\ C \mid \_ \rightarrow C \in Lex_w\ \}
\end{array}$$

Example 38. From the parse forest G_{w_5} above, $\mathsf{gen}^*\ (0,S,6)\ G_{w_5}$ returns the following two structures (to whom we added the terminal leaves for clarity).

$$\dfrac{\dfrac{\mathit{Mary}}{(0,NP,1)} \quad \dfrac{\dfrac{\mathit{saw}}{(1,V,2)} \quad \dfrac{\dfrac{\mathit{John}}{(2,NP,3)} \quad \dfrac{\dfrac{\mathit{with}}{(3,Pr,4)} \quad \dfrac{\dfrac{\mathit{the}}{(4,Det,5)} \quad \dfrac{\mathit{telescope}}{(5,N,6)}}{(4,NP,6)}}{(3,PP,6)}}{(2,NP,6)}}{(1,VP,6)}}{(0,S,6)}$$

$$\dfrac{\dfrac{\dfrac{\mathit{Mary}}{(0,NP,1)} \quad \dfrac{\dfrac{\mathit{saw}}{(1,V,2)} \quad \dfrac{\mathit{John}}{(2,NP,3)}}{(1,VP,3)}}{(0,S,3)} \quad \dfrac{\dfrac{\mathit{with}}{(3,Pr,4)} \quad \dfrac{\dfrac{\mathit{the}}{(4,Det,5)} \quad \dfrac{\mathit{telescope}}{(5,N,6)}}{(4,NP,6)}}{(3,PP,6)}}{(0,S,6)}$$

We can represent the two parses in parallel as a graph.



3.9.1 Parse forests for $AB^{\otimes}$ grammars

The generation of the parse forest for AB categorial grammars is parallel to the method for $\mathcal{CF}$ grammars presented in the previous section.

Algorithm 6. Let $G = \langle V_t, s, \mathcal{F}, Lex, AX\rangle$ be an $AB^{\otimes}$ grammar. Let a string $ws = w_1 \ldots w_n$ and the parse table T resulting by application of $\mathcal{CYK}_{AB^{\otimes}}$ to G and ws be given. We construct the parse forest $G_w = \langle V_t', (0, s, n), \mathcal{F}_w, Lex_w, AX_w\rangle$ resulting from G, ws and T, where V_t', $\mathcal{F}_w$ and Lex_w are as in algorithm 4. The productions of G_w are constructed as follows.

$$
\begin{array}{rcl}
AX_w & = & \{\ (j, a/b, k)\ (k, b, i) \to (j, a, i) \mid a/b \in t_{(j,k)}, \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad b \in t_{(k,i)}\ \} \\
 & & \qquad\qquad \cup \\
 & & \{\ (j, b, k)\ (k, b\backslash a, i) \to (j, a, i) \mid b \in t_{(j,k)}, \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad b\backslash a \in t_{(k,i)}\ \} \\
 & & \qquad\qquad \cup \\
 & & \{\ (j, a, k)\ (k, b, i) \to (j, a \otimes b, i) \mid a \in t_{(j,k)}, \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad b \in t_{(k,i)}\ \}
\end{array}
$$

Example 39. Let A_4 consist of the following lexical assignments:

$$
\begin{array}{ll}
\textit{saw} & \to (n\backslash s)/n \\
\textit{John} & \to n \\
\textit{Mary} & \to n \\
\textit{telescope} & \to c \\
\textit{the} & \to n/c \\
\textit{with} & \to (n\backslash n)/n \\
\textit{with} & \to (s\backslash s)/n
\end{array}
$$

Then, the parse forest returned by the construction in algorithm 6 applied to the table resulting from application of $\mathscr{CYK}_{AB^{\otimes}}$ to *John saw Mary with the telescope* and A_4 is A_{w_4} with the following AX set

$$
\begin{array}{ll}
(0, n, 1)\ (1, n\backslash s, 6) & \to (0, s, 6) \\
(0, s, 3)\ (3, s\backslash s, 6) & \to (0, s, 6) \\
(0, n, 1)\ (1, n\backslash s, 3) & \to (0, s, 3) \\
(3, (s\backslash s)/n, 4)\ (4, n, 6) & \to (3, s\backslash s, 6) \\
(3, (n\backslash n)/n, 4)\ (4, n, 6) & \to (3, n\backslash n, 6) \\
(1, (n\backslash s)/n, 2)\ (2, n, 6) & \to (1, n\backslash s, 6) \\
(1, (n\backslash s)/n, 2)\ (2, n, 3) & \to (1, n\backslash s, 3) \\
(2, n, 3)\ (3, n\backslash n, 6) & \to (2, n, 6) \\
(4, n/c, 5)\ (5, c, 6) & \to (4, n, 6) \\
(5, \textit{telescope}, 6) & \to (5, c, 6) \\
(4, \textit{the}, 5) & \to (4, n/c, 5) \\
(3, \textit{with}, 4) & \to (3, (s\backslash s)/n, 4)
\end{array}
$$

$$\begin{aligned}
&(3, \mathit{with}, 4) && \rightarrow (3, (n\backslash n)/n, 4)\\
&(2, \mathit{Mary}, 3) && \rightarrow (2, n, 3)\\
&(1, \mathit{saw}, 2) && \rightarrow (1, (n\backslash s)/n, 2)\\
&(0, \mathit{John}, 1) && \rightarrow (0, n, 1)
\end{aligned}$$

Then the procedure in definition 5 can immediately applied to A_{w_4}, as we moved into the axioms AX the result of all the inferences. Application of **gen*** $(0, s, 6)$ A_{w_4} returns the following trees.

$$\dfrac{\dfrac{\mathit{John}}{(0,n,1)} \quad \dfrac{\dfrac{\mathit{saw}}{(1,(n\backslash s)/n,2)} \quad \dfrac{\dfrac{\mathit{Mary}}{(2,n,3)} \quad \dfrac{\dfrac{\mathit{with}}{(3,(n\backslash n)/n,4)} \quad \dfrac{\dfrac{\mathit{the}}{(4,n/c,5)} \quad \dfrac{\mathit{telescope}}{(5,c,6)}}{(4,n,6)}}{(3,n\backslash n,6)}}{(2,n,6)}}{(1,n\backslash s,6)}}{(0,s,6)}$$

$$\dfrac{\dfrac{\dfrac{\mathit{John}}{(0,n,1)} \quad \dfrac{\dfrac{\mathit{saw}}{(1,(n\backslash s)/n,2)} \quad \dfrac{\mathit{Mary}}{(2,n,3)}}{(1,n\backslash s,3)}}{(0,s,3)} \quad \dfrac{\dfrac{\mathit{with}}{(3,(s\backslash s)/n,4)} \quad \dfrac{\dfrac{\mathit{the}}{(4,n/c,5)} \quad \dfrac{\mathit{telescope}}{(5,c,6)}}{(4,n,6)}}{(3,s\backslash s,6)}}{(0,s,6)}$$

3.10 From $AB^{\otimes}$ to AB

The presence of product formulas in $AB^{\otimes}$, and of a product rule especially, represents a disadvantage from the parsing perspective as we have seen in the previous sections. In this section, we will show how the multi-modal setting can improve the situation. As $AB^{\otimes}$ grammars and AB grammars generate the same class of languages, namely the $\mathcal{CF}$ languages, we can define a procedure transforming an $AB^{\otimes}$ grammar G into an AB grammar G' generating the same terminal language in an easy way. On the other hand, as we explained in chapter 1, our interest in the systems with product is motivated by the strong generative capacity of such systems. Indeed, the structural language generated by G' is not necessarily the same as the one generated by G. In order to avoid such collapse of structural descriptions, we will define an inverse mapping from derivations in G' to derivations in G. We explain informally how this two processes work.

In Lambek [1958], one finds the discussion of an operation attributed to Schönfinkel. In the *associative* Lambek calculus, the following principles hold.

(3.2) $c/(a \otimes b) \leftrightarrow (c/b)/a$

(3.3) $(b \otimes a)\backslash c \leftrightarrow a\backslash(b\backslash c)$

As usual, the double arrow means that the inference is in both directions. Schönfinkel "observed that a function of two variables may be regarded as an ordinary function of one variable whose value is again an ordinary function[5]", so that

$$f(a, b) = f(a)(b)$$

In the literature, the direction → of the inference is called *currying*, while the direction ← is called *uncurrying*.

Obviously, from the parsing perspective, the advantage of using the curried variant of a formula is that the product rule is no longer needed. The only case in which we might still need the product rule occurs when we are going to parse a string as of category $a \otimes b$. However, observe that if $\Gamma \rightarrow c$ is provable, then also $\Gamma, c\backslash c \rightarrow c$ is. Thus, currying could apply to $c\backslash c$, eluding the use of the product rule.

Although these two inferences do not hold in *NL*, we will use the curried variant of a sequent into the computation under the following constraints.

- Curried formulas are marked with a distinguished mode of composition which prevents them from ever being arguments of other categories.

- Each application of currying has to be canceled by a corresponding application of uncurrying.

In this way, we can exploit the currying-uncurrying symmetry as an expedient for simplifying the computation.

In what follows, we will work with *arrows*, objects of the form $f : \Gamma \rightarrow c$, where f is a syntactic term encoding the proof of $\Gamma \rightarrow c$. The proof terms will play an essential role in the procedure. We will define currying as an operation applying to an arrow $f : a \rightarrow a$ such that a is a lexical category and returning, the set of arrows $f' : a \rightarrow a'$, where a' is a curried variant

[5] Lambek [1958]

of a and f' encodes the way in which currying has been applied to a in order to produce a'. Then, we define the converse operation which we call uncurrying. Uncurrying takes in input a proof term, as $(f'^{/}(g))^{/}(g')$, the term encoding the right application of the curried arrow labeled f' to the arrow labeled g and to the one labeled g', in the order, and uses the codification in f' to perform the restructuring of its argument. One should expect the following rewriting rules.

Example 40.

Uncurry of $(f'^{/}(g))^{/}(g')$ gives $f^{/}(g * g')$.

Uncurry of $(f'^{\backslash}(g))^{\backslash}(g')$ gives $f^{\backslash}(g' * g)$.

We propose now the labeled variant of system $AB^{\otimes}$.

Definition 60. Labeled $AB^{\otimes}$:

$$1_a : a \to a$$

$$\frac{f : \Gamma \to a \quad g : \Delta \to b}{f * g : (\Gamma, \Delta) \to a \otimes b}$$

$$\frac{g : \Gamma \to a' \quad f : \Delta \to a' \backslash c}{f^{\backslash}(g) : (\Gamma, \Delta) \to c} \qquad \frac{f : \Gamma \to c/b' \quad g : \Delta \to b'}{f^{/}(g) : (\Gamma, \Delta) \to c}$$

3.10.1 Currying

Consider a formula as $a/(b \otimes (c \otimes d))$. Such formula takes as argument a formula $b \otimes (c \otimes d)$ to give a formula a. The argument $b \otimes (c \otimes d)$ can be a lexical category, or it can be derived from a formula b and a formula $c \otimes d$. This second formula, can itself be lexical, or derived. As we cannot exclude any option, currying should return at least a set of formulas, which, in this case, is $\{a/(b \otimes (c \otimes d)), a/(c \otimes d)/b, a/d/c/b\}$.

Besides, assume a formula a/b' appears in the parsing process and that for some formula b, b' results from currying b. Therefore, we could, by mistake, derive a from a/b' and b' as $b \to b'$ is not a licit transition in the non-associative logic. Therefore, curried formula should be marked as to avoid invalid pattern matching.

Last, the formula $a/(b \otimes c)$ project a structure which is different from the one projected by $a/c/b$. Thus, currying would change the strong generative capacity of the grammar, while the reason to work with AB with product is mainly its strong generative power. In order to avoid this structural collapse, we introduce some operators which encode the order of the currying operation and can be used in the uncurrying phase to recover the proof generated by the original formulas.

- Operators[6]:

$$\mathcal{OP} := \sigma^{\mathcal{C}} \mid \iota \mid \epsilon \mid \gamma$$

For each operator φ and combinator f, we write $\varphi(f)$ the application of φ to f. These operators are introduced by the following function returning all the possible curried variants of an arrow[7].

Definition 61.

$\Sigma :: (\mathcal{CB}, \mathcal{F}) \to \{(\mathcal{CB}, \mathcal{F})\}$

- $\Sigma(f : c/_i(a \otimes_j b)) =$

$$\{\ \sigma^{/_i}(f) : c/_* b/_* a\ \}$$
$$\cup$$
$$\Sigma(\sigma^{/_i}(f) : c/_* b/_* a)$$
$$\cup$$
$$\{\ \iota(f') : c'/_i(a \otimes b) \mid f' : c' \in \Sigma(\epsilon(f) : c)\ \}$$

- $\Sigma(f : c/_i a) = \{\ \iota(f') : c'/_i a \mid f' : c' \in \Sigma(\epsilon(f) : c)\ \}$

- $\Sigma(\_) = \{\}$

The function Σ takes in input a pair of a syntactic term and a multi-modal formula, which we write $f : x$, and scans the arguments of x looking for a product in argument position. When Σ encounters a subformula of the form $c/_i(a \otimes b)$, it restructures it to $c/_* b/_* a$, and applies the operator $\sigma^{/_i}$ to the term of the arrow and starts currying this new arrow. Here, i and j are a variables ranging over modes of composition, while $*$ is a distinct mode of composition which should appear only on curried formulas. While accessing embedded arguments, the function Σ encodes in the syntactic term the original position of the scanned argument by means of the ϵ and ι operators. Every ϵ operator marks the original position of an argument, while the

[6] Such operators have a purely syntactical propose in that they encode the application of currying.

[7] We often call arrow a *pair* of a syntactic term and a formula. As the term encodes a proof, it encodes also its conclusion. However, we are often interested only in one formula of the conclusion. Thus, for simplicity we dropped the other one, this being always recoverable by the term.

corresponding ι operator marks its new position after restructuring. Thus, at the end of currying, the occurrences of ε and ι will be balanced. Indeed, the distribution of the operators encoding currying can be captured by the following context-free grammar, where $\epsilon \neq \varepsilon$ and ϵ is the empty string:

Definition 62.

Grammar for the distribution of currying operators:

$$S := SS \mid \sigma \mid \iota\, S\, \varepsilon \mid \epsilon$$

Observe that if no istance of product is found in argument position, the function returns the empty set.

The above transformation guarantees that we will not need the product rule for the input categories. The remaining case is when the output category is a product. Instead of proving that $\Gamma \to c$, we can prove that $\Gamma, c\backslash c \to c$. The advantage of this move is that if $c = c_1 \otimes c_2$, we can apply currying to $c\backslash c$. Observe that $c\backslash c$ should be taken as argument. Hence, the following function uses again a multi-modal encoding to constrain pattern matching.

Definition 63.

$\Sigma'(f : c) =$

$\{\, \gamma(f) : c\backslash_{*}\Box_{*}\Diamond_{*}c \,\}$

$\cup$

$\Sigma(\gamma(f) : c\backslash_{*}\Box_{*}\Diamond_{*}c)$

This operation produces the arrow coming from the left discharge of the antecedent together with all the arrows deriving from its currying. For any formula c, we will write $\Box_{*}\Diamond_{*}c$ as $c^{\star}$.

We prove the equivalence between any $AB^{\otimes}$ grammar G and its curried variant G'.

Proposition 10. Let $G = \langle V_t, s, Lex, AB^{\otimes}\rangle$ be given. Let $G' = \langle V_t, s, Lex \cup Lex', AB\rangle$, where

$Lex' = \{\, w \to a' \mid w \to a \in Lex,\ f : a' \in \Sigma(1_a : a) \,\}$

Then, $L_t(G) = L_t(G')$.

Proof. induction on the $AB^{\otimes}$ derivation D of $a_0 \dots a_n \to s$ such that $w_i \to a_i \in Lex$ for all $0 \leqslant i \leqslant n$. Assume D ends with

$$\dfrac{\dfrac{a_0 \dots a_k \to a \quad a_{k+1} \dots a_i \to b}{a_0 \dots a_i \to a \otimes b} \qquad a_{i+1} \dots a_n \to (a \otimes b)\backslash s}{a_0 \dots a_n \to s}$$

If $a_{i+1}\dots a_n \to (a \otimes b)\backslash s$, then for some $w_j \to a_j$, $i < j \leqslant n$, $(a \otimes b)\backslash s$ is a subformula of a_j. Thus, $b\backslash_* a\backslash_* s$ is a subformula of some a'_j in G' obtained by currying a_j and the derivation in AB runs as follows.

$$\frac{a_0\dots a_k \to a \qquad \dfrac{a_{k+1}\dots a_i \to b \quad a_{i+1}\dots a_n \to b\backslash_* a\backslash_* s}{a_{k+1}\dots a_n \to a\backslash_* s}}{a_0\dots a_n \to s}$$

Besides, no curried formula will ever be consumed as an argument in G', due to the multi-modal labeling of curried formulas with the distinguished composition mode $*$. □

3.10.2 Uncurrying

We implement the operation of uncurrying as taking place at the end of the derivation process. It transforms the proof term of the final arrow to its uncurried variant, if any of its subterms have been curried before, otherwise it returns the term itself. We first give the basic cases of the uncurrying normalization in the form of rewriting rules $f \rightsquigarrow g$, to be read as "the term f rewrites as the term g". Then we generalize to unbounded contexts.

The first rule applies to the curried output.

(3.4) $(\gamma(f))^{\backslash}(g) \rightsquigarrow f \circ g$

In fact, if $f : b \to c$, then $\Sigma'(f : b \to c) = \gamma(f) : b\backslash_* c^\star$. Assume we have derived $g : b$ from assumptions Γ. Then, the two following derivations are equivalent.

(3.5)

$$\frac{\begin{array}{c}\Gamma\\ \vdots\\ g : b\end{array} \quad \gamma(f) : b\backslash_* c^\star}{(\gamma(f))^{\backslash}(g) : c^\star} \qquad\qquad \begin{array}{c}\Gamma\\ \vdots\\ g : b\\ \vdots\; f : b \to c^\star\\ f \circ g : c^\star\end{array}$$

To be fully explicit we should remark that for every formula a, $\vdash a \to a^\star$. However, we may neglect such transition from the term f as it is inessential to the aim of the computation.
Uncurrying has the following two primitive instances.

(3.6) $((\sigma^{/}(f))^{/}(g))^{/}(h) \rightsquigarrow f^{/}(g * h)$

(3.7) $((\sigma^{\backslash}(f))^{\backslash}(g))^{\backslash}(h) \;\rightsquigarrow\; f^{\backslash}(h * g)$

Example 41. Let us consider the first reduction. Let $f : a/(b \otimes c)$ be given. Then $\Sigma(f\colon a/(b \otimes c)) = \{\sigma(f)\colon a/_{*}c/_{*}b\}$[8].

$$\dfrac{\dfrac{\sigma^{/}(f)\colon a/_{*}c/_{*}b \quad \begin{matrix}\Gamma\\ \vdots\\ g\colon b\end{matrix}}{(\sigma^{/}(f))^{/}(g) \,:\, a/_{*}c} \quad \begin{matrix}\Delta\\ \vdots\\ h\colon c\end{matrix}}{((\sigma^{/}(f))^{/}(g))^{/}(h)\colon a} \qquad\qquad \dfrac{f\colon a/(b\otimes c) \quad \dfrac{\begin{matrix}\Gamma\\ \vdots\\ g\colon b\end{matrix} \quad \begin{matrix}\Delta\\ \vdots\\ h\colon c\end{matrix}}{g * h\colon b\otimes c}}{f^{/}(g*h)\colon a/c}$$

The cancellation of the ι and ϵ operators is slightly more complicate to show, primarily because these operators appear embedded into terms as $(\ldots((\iota(f))^{|}(g_1))^{|}(g_2)\ldots)^{|}(g_n)$ or $(\ldots((\epsilon(f))^{|}(g_1))^{|}(g_2)\ldots)^{|}(g_n)$, where each $|$ is any of the two slashes, and the restructuring they give rise to is *unbounded*. Therefore, we assume that the rewriting rules for these operators keep track of the argument combinators with the aid of two lists containing the unfolded arguments to be restructured.

(3.8) $\iota(f)\;\; (g{:}gs)\;\; hs \rightsquigarrow f\;\; gs\;\; (g{:}hs)$

(3.9) $\epsilon(f)\;\; gs\;\; (h{:}hs) \rightsquigarrow f\;\; (h{:}gs)\;\; hs$

In the following definitions, we make use of lifted terms of the form $\lambda h.h^{|}(g)$, which we call them $\mathcal{ACB}$-terms. These are functions from $\mathcal{CB}$ to $\mathcal{CB}$, that is, if $g \in \mathcal{ACB}$ and $f \in \mathcal{CB}$, then $(g\ f) = f(g)$[9]. We decompose the uncurrying process in three main parts. The function Υ^{*} initializes and finalizes the procedure.

Definition 64. $\Upsilon^{*} :: \mathcal{CB} \rightarrow \mathcal{CB}$

$$\begin{array}{lcl} \Upsilon^{*}\,(f^{|}(g)) & = & \text{let } f' \text{ be } \Upsilon((f^{|}(g)),\ [\,],\ [\,]) \text{ in} \\ & & \quad h \circ g', \quad \text{if } f' \equiv (\gamma(h))^{\backslash}(g') \\ & & \quad f', \quad \text{otherwise} \\ \Upsilon^{*}\,f & = & f \end{array}$$

If the input is an application term, of the form $(f^{|}(g))$, it is sent to the Υ procedure, defined below, which in turn is initialized with two empty

[8] This is what actually Σ computes. The initial item $f\colon a/(b \otimes c)$ is be added by default.

[9] Observe that the meta-level function application, $(g\ f)$, is distinct from object-level application, $f(g)$.

lists. If the term returned by the Υ subroutine is of the appropriate form, Υ^* applies the rewrite rule in example 3.4.
The main normalization of the uncurrying function is defined by the following clauses. We call the list gs of $\Upsilon(f^{|}(g),\ gs,\ hs)$ main $\mathcal{ACB}$-list and hs, auxiliary $\mathcal{ACB}$-list.

Definition 65. $\Upsilon :: (\mathcal{CB}, [\mathcal{ACB}], [\mathcal{ACB}]) \to \mathcal{CB}$

$$\Upsilon(f^{|}(g), gs, gs') = \Upsilon(f, (\lambda h.h^{|}(g') : gs), gs') \text{ where } g' = \Upsilon^* \ g$$

$$\Upsilon(\sigma^{/}(f), (\lambda h.h^{/}(g_i) : \lambda h.h^{/}(g_k) : gs), gs') = \Upsilon(f, (\lambda h.h^{/}(g_i * g_k) : gs), gs')$$

$$\Upsilon(\sigma^{\backslash}(f), (\lambda h.h^{\backslash}(g_i) : \lambda h.h^{\backslash}(g_k) : gs), gs') = \Upsilon(f, (\lambda h.h^{\backslash}(g_k * g_i) : gs), gs')$$

$$\Upsilon(\iota(f), (g : gs), gs') = \Upsilon(f, gs, (g : gs'))$$

$$\Upsilon(\epsilon(f), gs, (g : gs')) = \Upsilon(f, (g : gs), gs')$$

$$\Upsilon(f, gs, [\,]) = \texttt{fold}\ f\ gs$$

Once the auxiliary list of $\mathcal{ACB}$-terms is exhausted, and no other instance pattern is applicable, we may fold back the input term.

Definition 66.

$\texttt{fold} :: \mathcal{CB} \to [\mathcal{ACB}] \to \mathcal{CB}$
 $\texttt{fold}\ f\ [\,] = f$
 $\texttt{fold}\ f\ (g : gs) = \texttt{fold}\ (g\ f)\ gs$

Consider the following example. We want to prove that given the arrows $f : d/(a \otimes (b \otimes c))$, $g_1 : a$, $g_2 : b$ and $g_3 : c$, we can derive $f^{/}(g_1 * (g_2 * g_3)) : d$ *without using the product rule*, but instead the currying-uncurrying symmetry as defined above. We proceed along the following steps:

1. Currying of the assumptions. We assume that the only arrow returning a non empty list is f: $d/(a \otimes (b \otimes c))$. $\Sigma(f$: $d/(a \otimes (b \otimes c)))$ gives

$$\{\ \sigma^{/} f : d/_*(b \otimes c)/_*a,$$
$$\iota\sigma^{/}\epsilon\sigma^{/} f : d/_*c_*/b_*/a\ \}$$

2. Proof by means of cancellation schemes only. We write f' for $\iota\sigma^{/}\epsilon\sigma^{/} f$:

$$\cfrac{\cfrac{\cfrac{f' : d/_*c/_*b/_*a \quad g_1 : a}{f'^{/}(g_1) : d/_*c/_*b} \qquad g_2 : b}{(f'^{/}(g_1))^{/}(g_2) : d/_*c} \qquad g_3 : c}{((f'^{/}(g_1))^{/}(g_2))^{/}(g_3) : d}$$

3. Uncurrying of the output term $((\iota(\sigma^{/}(\epsilon(\sigma^{/}f)))^{/}(g_1))^{/}(g_2))^{/}(g_3)$. We assume stage 0 has estracted all the argument terms (first line of the function Υ), and we look only at the operation of recomposition of the structure.

	Input	Main $\mathcal{ACB}$'s	Auxiliar $\mathcal{ACB}$'s
0	$\iota\sigma^{/}\epsilon\sigma^{/}f$	$[\,\lambda h.\, h^{/}(g_1),\ \lambda h.\, h^{/}(g_2),\ \lambda h.\, h^{/}(g_3)\,]$	$[\,]$
1	$\sigma^{/}\epsilon\sigma^{/}f$	$[\,\lambda h.\, h^{/}(g_2),\ \lambda h.\, h^{/}(g_3)\,]$	$[\,\lambda h.\, h^{/}(g_1)\,]$
2	$\epsilon\sigma^{/}f$	$[\,\lambda h.\, h^{/}(g_2 * g_3)\,]$	$[\,\lambda h.\, h^{/}(g_1)\,]$
3	$\sigma^{/}f$	$[\,\lambda h.\, h^{/}(g_1), \lambda h.\, h^{/}(g_2 * g_3)\,]$	$[\,]$
4	f	$[\,\lambda h.\, h^{/}(g_1 * (g_2 * g_3))\,]$	$[\,]$

4. $(\lambda h.h^{/}(g_1 * (g_2 * g_3))\ f) = f^{/}(g_1 * (g_2 * g_3))$.

5. Mapping $f^{/}(g_1 * (g_2 * g_3))$ to a proof tree gives the following result.

$$\dfrac{f : d/(a\otimes(b\otimes c)) \qquad \dfrac{g_1 : a \qquad \dfrac{g_2 : b \quad g_3 : c}{g_2 * g_3 : b\otimes c}}{g_1 * (g_2 * g_3) : a\otimes(b\otimes c)}}{f^{/}(g_1 * (g_2 * g_3)) : d}$$

We give another example to make more clear the way curry and uncurry work. Assume the process involves $f : d/((a\otimes b)\otimes c)$ and $g_1 : a$, $g_2 : b$ and $g_3 : c$.

1. Currying of the arrow $f : d/((a\otimes b)\otimes c)$ would give the following set.

$$\{\ \sigma^{/}f : d/_{*}c/_{*}(a\otimes b),$$
$$\sigma^{/}\sigma^{/}f : d/_{*}c/_{*}b/_{*}a\ \}$$

2. Although the formula occurring in $\sigma^{/}(\sigma^{/}(f)) : d/_{*}c/_{*}b/_{*}a$ is the same as the one of the previous example, the term is different and the normalization would run as follows.

	Input	Main $\mathcal{ACB}$'s	Auxiliar $\mathcal{ACB}$'s
0	$\sigma^{/}\sigma^{/}f$	$[\,\lambda h.\, h^{/}(g_1),\ \lambda h.\, h^{/}(g_2),\ \lambda h.\, h^{/}(g_3)\,]$	$[\,]$
1	$\sigma^{/}f$	$[\,\lambda h.\, h^{/}(g_1 * g_2),\ \lambda h.\, h^{/}(g_3)\,]$	$[\,]$
2	f	$[\,\lambda h.\, h^{/}((g_1 * g_2) * g_3)\,]$	$[\,]$

3. And the term $f^{/}((g_1 * g_2) * g_3)$ would project the original structure.

$$\frac{f : d/((a \otimes b) \otimes c) \quad \dfrac{\dfrac{g_1 : a \quad g_2 : b}{g_2 * g_3 : a \otimes b} \quad g_3 : c}{(g_1 * g_2) * g_3 : (a \otimes b) \otimes c}}{f^{/}((g_1 * g_2) * g_3) : d}$$

While we proved in proposition 10 that the curried variant G' of an $AB^{\otimes}$ grammar G generates the same terminal language as G, we now state the equivalence between the uncurrying of a derivation in G' the corresponding derivation in G.

Proposition 11. Let $G = \langle V_t, s, Lex, AB^{\otimes} \rangle$ be given. Let $G' = \langle V_t, s, Lex \cup Lex', AB \rangle$, where

$Lex' = \{\, f' : w \to a' \mid f : w \to a \in Lex,\ f' : a' \in \Sigma(f : a) \,\}$

Then, G generates $f : ws \to s$ if and only if G' generates $f' : ws \to s$ such that $\Upsilon^{*}(f') = f$.

Proof. From proposition 10, we know that G and G' are equivalent. Thus, we have to prove that Υ and Υ^{*} are correct. We always consider only one of the two slashes, the other being symmetric.

1. Assume that the Υ normalization is in stage m below.

$$\begin{array}{llll} m & \iota(f) & (g : gs) & hs \\ m+1 & f & gs & (g : hs) \end{array}$$

   Then, if $g : a$, then $\iota(f) : c/a$ for some formula c such that c contains a subformula $z/_{*}y/_{*}x$ obtained from a subformula $z/(x \otimes y)$ by application of Σ. Therefore, g is not in its original position and some of the gs should be restructured. Thus, we postpone the consumption of g by moving it to the auxiliary list hs in stage $m+1$.

2. Assume that the Υ normalization is in stage m below.

$$\begin{array}{llll} m & \epsilon(f) & gs & (g : hs) \\ m+1 & f & (g : gs) & hs \end{array}$$

   Then, if $g : a$, then $\epsilon(f) : c$ for some formula c such that $f : c/a$ (see definition 61). Thus that is the point in which g should be restored among the arguments. We move it back to the main argument list in stage $m+1$.

3. Assume that the Υ normalization is in stage m below.

$$\begin{array}{llll} m & \sigma^{/}(f) & (g\!:\!g'\!:\!gs) & hs \\ m+1 & f & (g \otimes g'\!:\!gs) & hs \end{array}$$

   Then, if $g\colon a$ and $g' : b$, then $\sigma^{/}(f)\colon c/_{*}b/_{*}a$ where $f : c/(a \otimes b)$. Thus f would have consumed $g \otimes g'$, where $\sigma^{/}(f)$ consumed first $g' : b$ and then $g\colon a$. Hence, in stage $m+1$, we replace the first two terms of the main list with a new term which is the product of the two.

4. After scanning all occurrences of such operators in f', the auxiliary list will become empty, and the main list will contain all arguments of the original term f from the last to the first. Hence the `fold` procedure applies. Observe also that the operator γ, if it appears, will be the most external one. Thus its restructuring operates in the final stage, in the main function Υ^{*}.

□

3.11 Parsing approaches for Lambek systems

Lambek style categorial grammars are intrinsically richer than basic categorial grammars. As we said before, the great difference is that while basic categorial systems only compose larger structures from simpler ones (as $\mathcal{CF}$ grammars) Lambek grammars may also *decompose* complex structures into simpler ones.

In this chapter, we how to regiment the composition rules by means of indices. On the other hand, the introduction rules of Lambek systems

$$\frac{\Gamma, b \to c}{\Gamma \to c/b} \qquad \frac{a, \Gamma \to c}{\Gamma \to a\backslash c}$$

with $\Gamma \neq \epsilon$, do not seem to have a clear indexed counterpart.

An attempt of applying chart methods to hypothetical reasoning of Lambek calculi is [König, 1994]. However, this "method requires rather complicated book-keeping"[10].

In automatic proof search for Lambek style grammars, one often adopts a *cut-free* axiomatization of the underlying logic. Then, the proof of a sequent may consist in reaching axioms by forward application (that is,

[10] [Hepple, 1999]

from conclusion to premises) of each rule of the calculus. Examples of such approach, also called *goal directed* proof search, can be found in [Moortgat, 1988], [Hendriks, 1993] and in [Andreoli, 1992] within the broader field of linear logic, see also [Moortgat, 1997b]. However, we should mention that unless the underlying logic is associative, these approaches require the input sequent, the goal, to be already assigned a structure to begin with. Secondly, one may immediately notice that the conclusion-to-premises approach is not *sound*: although it guarantees that a solution can be found, if it exists, it encounters also several non-valid sequents along the process. For example, the following is a possible result of applying, conclusion-to-premises, the left rule for \ of the sequent calculus[11]:

$$\frac{a/(b\backslash a) \to b \quad a \to a}{a/(b\backslash a),\, b\backslash a \to a}$$

Observe that there is no type invariant (as the balancing constraints of [van Benthem, 1991, Moortgat, 1988]) preventing the occurrence of $a/(b\backslash a) \to b$.

Other approaches to parsing with Lambek grammars can be found in the literature. [Finkel and Tellier, 1996] formulate a CYK style algorithm for the product-free associative Lambek calculus. Their method is based on the algorithm in [Pentus, 1993] for the conversion of a Lambek grammar into a *CF* grammar. The recognition algorithm is cubic on the length of the input string, although the grammar conversion results in “a long process” and “complicated”, as the authors admit. Indeed, Pentus’ translation is exponential on the size of the input Lambek grammar, and this may have serious consequences for the recognition procedure itself.

Another approach is the one of [Hepple, 1996, 1999] based on a method of *first-order compilation*. This method “involves excising the subformulae that correspond to hypotheticals, leaving a first-order residue. The excised subformulae are added as additional assumptions”[12]. As the one of König, Hepple’s approach constrains the use of “hypotheticals” through a rather complicate mechanism of indexing and index sharing. This mechanism allows then the application of standard parsing techniques (viz. Earley style parsing) to Lambek grammars.

[11] The rule we are referring to is usually presented as follows:

$$\frac{\Delta \to b \quad \Gamma[a] \to c}{\Gamma[(\Delta,\, b\backslash a)] \to c}$$

[12] [Hepple, 1999]

Other parsing algorithms have been based on *proof nets*. [Morrill, 1996] designs CYK style parsers for proof nets for the Lambek calculus (associative and non-associative). [Moot, 2002] is undoubtedly the most extensive study of linguistic applications of proof nets. This work addresses the entire multi-modal setting from the proof net perspective and proves that if the structural rules are linear and non-expanding (the shape commonly required for natural language analysis) recognition is *P*-space complete.

Most of the works discussed briefly in this section have been designed for the *associative* Lambek calculus. Nowadays, it is not surprising that they are not polynomial, as we know from [Pentus, 2006] (circulated before as [Pentus, 2003]) that the calculus of [Lambek, 1958] is *NP*-complete. A better situation holds for other fragments of the Lambek calculus. [Savateev, 2006] proves that sequents of the unidirectional associative Lambek calculus can be recognized in cubic time. Instead, [de Groote, 1999, de Groote and Lamarche, 2002] prove that two-formula sequents of non-associative Lambek calculus (that is sequents whose antecedent structure is expressed through the branching of products) can be recognized in polynomial time. [Buszkowski, 2005] and [Buliśka, 2006] extend this result to structured sequents of *NL* enriched with non-logical axioms (and empty antecedent in the work of Buliśka).

A common feature of most of the approaches mentioned before is that the parsing process is divided in two main components: a grammar preprocessing module, 'simplifying' the structure of syntactic categories, and an actual parsing module, operating on the simplified categories and implementing (some variant of) some traditional context-free parsing algorithm.

In the next chapters, we will follow this approach in implementing an efficient grammar preprocessing module for grammars based on the non-associative Lambek calculus. This module will allow the application of the parsing systems discussed in this chapter to non-associative Lambek grammars with product.

3.12 Conclusion

This chapter presents parsing methods for $AB^{\otimes}$ grammars based on the CYK method. We proved that the recognition procedure works in cubic time as its context-free counterparts. Parse trees instead can be extracted from the parse forest in time proportional to the length of the input string. We have also shown that the product rule is dispensable, in the sense that we can obtain an $AB^{\otimes}$ deduction by exploiting the symmetry that we called currying and uncurrying.

In chapter 5, we will show how to extend the methods presented here to grammars based on *NL*.

Part II

The Non-associative Lambek Calculus

Chapter 4

A Case Study: Cliticization

In this chapter, we apply the the tools and techniques formally presented in chapter 2 to linguistics and examine some characteristic phenomena of Italian syntax.

Our analysis is primarily concerned with the exploration of the expressive power of the type-logical setting in application to specific language phenomena and does not mean to be exhaustive. On the other hand, we hope to make more concrete many of the things abstractly discussed in the previous chapters and lay the ground for further analysis.

We will examine the phenomena of *cliticization* and so called *clitic left dislocation* (CLD) from a base-logical perspective.

Clitics are atone pronouns. In Italian, for instance, *la*, *gli*, *si*, *ne* as in

(4.1) veder *-la*
to see her
to see her

(4.2) *gli* parla
to him he speaks
he speaks to her

and so forth, are clitic pronouns. Syntactically, clitics attach tightly to the verb, nothing being allowed to intervene between a clitic and its 'host'. We use the star notation for ungrammaticality.

(4.3) **gli* non parla
to him not he speaks

(4.4)	**gli*	gentilmente	parla
	to him	gently	he speaks

In Italian they precede a finite tense verbal head and follow a non finite one. If more than one, we have a, so called, *cluster*. In that case clitics appear in a fixed order.

(4.5)	*te*	*lo*	racconto
	to you	it	I tell

I tell it to you

(4.6)	**lo*	*te*	racconto
	it	to you	I tell

Clustering is also subject to further morphological constraints, which we will partially illustrate in section 4.2.1 and we refer to Monachesi [1999] for a more articulated analysis of these aspects. As pronouns, they *saturate* an argument positions. Nevertheless, their occurrence does not exclude the possibility of the co-occurrence of (one or more) full noun phrases, each concordant with one of the clitics, to the left and/or right of the cliticized clause.

(4.7)	Mario,	*lo*	ho	visto	ieri
	Mario	him	I have	seen	yesterday

I have seen Mario yesterday

By *clitic left dislocation* (CLD) is meant the co-occurrence, typical of Romance languages, of a left extraposed constituent and of a clitic pronoun coindexed with it. After cliticization, a left peripheral position is made available for a dislocated phrase. Since the clitic, as a pronoun, saturates an argument position of the verb, the dislocated phrase is *optional*. Nonetheless this optional position is licensed only after cliticization has taken place. The character of 'licensed optionality' of the extraposed constituent, also called *satellite*, has to be made explicit in the structure of the categories involved in the construction. We will show how the phenomenon goes beyond the resource sensitive regime of type-logical grammar, violating the *relevance* constraint and we present several options to accommodate it into the system. We will discuss the limits and advantages of various treatments, some of which have been developed within the categorial literature (see, in particular, Hendriks [1999], Sanfilippo [1990]).

4.1 Information Structure

The research field we are going to explore is that of so called *information packaging* (IP for short), also known as *information structure*: an area of linguistic research which lies in between syntax and prosody. In the theory of IP developed in Vallduvi [1990], linguistic information is divided into two components: the *focus* which the only obligatory part which carries new information, and *ground*. It is helpful to look at such phenomena in dialogue rather than in simple sentences. For example, the partition focus-ground can be exemplified in the following short exchange where, for the answer (A), we white the focus in *italic* and the ground in SMALL CAPS.

(4.8) Q: Who did you see?
A: I SAW *Mario*.

The ground is optional (one may just reply *Mario*, an all focus answer) and divides further into *link* (which we write in **boldface**) and *tail*.

Q:	(4.9)	Quando	hai	visto	Mario?
		When	have you	seen	Mario

When did you see Mario

A: (4.10) **Mario,** LO HO VISTO *ieri*

I have seen Mario yesterday

In terms of *information*, links perform an 'ushering' role[1]: they have the informational role of signaling to the hearer, in Vallduví's terminology, *where* the new information, provided by the focused part of the sentence, has to be stored, in which 'file card', assuming the mental model of the context is actually structured as in Heim [1983]. Tails signal to the hearer *how* the information must fit into his knowledge store, in particular if the focused information has to replace some preexisting information. In languages like Catalan or Italian, links are realized sentence initially and accompained by cliticization of the corresponding verbal arguments, as we can see from the previous example. In English they are expressed through the L+H* accent of Pierrehumbert and Hirshberg [1990]. Tails are 'clitic *right* dislocated'

[1] Vallduví adopts Heim's file change semantics (FCS), and therefore a *locational* theory of context. The non-locational theory of Hendriks appeals to non-monotonic anaphora, see Hendriks and Dekker [1998]. We continue to speak about locations, presently, as a good *metaphor*.

noun phrase expressions in Italian and Catalan, and result unaccented in English.

Several logical approaches to the syntax and prosody of IP have been proposed in recent years. A common aim of approaches developed within different traditions of categorial grammar is the attempt of capturing, through the lexicon, constituent structures which are 'odd' with respect to traditional syntactic constituency.

Most notably, the approaches developed in the combinatory categorial grammar (CCG) framework consider the more flexible criterion for syntactic structure arising from the combinatorial system[2] as a characterization of *prosodic constituency*: all prosodically possible branchings are generated in CCG derivations and the notion of prosodic constituent is claimed to correspond with that of syntactic constituent. The analyses of [Steedman, 2000b,a] are focused on the prosodic realization of IP, and further extensions are still needed in order to apply CCG to languages like Italian or Catalan which realize IP through the syntax.

Prosody is a central concern also for the approaches rooted in the type-logical and model-theoretic tradition, we mention Oehrle [1988], Morrill [1994], Moortgat [1997a], Moortgat and Morrill [1991],Hendriks [1999] among others. These works adopt and refine the setup firstly introduced in Oehrle [1988]: linguistic resources are conceived as objects characterized by a prosodic, a syntactic and a semantic dimension and the deduction of a complex expression requires well-formedness of each component.

Hendriks [1999] provides a type-logical account of IP in Catalan in which verbal morphology and clitic pronouns are functors taking the focused part of the clause as argument and leaving esternal argument positions for dislocated phrases. We will discuss Hendrik's approach in more detail in section 42 and we note here that the introduction of such external positions by elements internal to the core clause makes the role of dislocated phrases that of obligatory arguments of the core clause.

An original logical approach to cliticization is the one of [Casadio and Lambek, 2001], see also [Casadio, 2002]. This work is based on the system of *pregroups* recently developed in [Lambek, 2001, Casadio and Lambek, 2002]. We will see examples of their treatments in section 4.1.2, because nowadays several analyses of specific linguistic phenomena are being developed within the pregroups setting and because pregroups very similar to the type-logical systems that we are studying.

[2] And its analysis of 'non-constituent' coordination.

A common feature of all the logical approaches to information structure is the use of *higher order categories*, lexicalized as intonational boundaries or empty (phonetically null) elements. The combinatorial analyses of [Steedman, 2000b,a] make use of higher order types in order to constrain the number of the accessible structures for a derivation in the flexible constituency regime made available by the combinatory categorial system. The higher order functors of Hendriks, instead, precompile permutative operations (in the case of Catalan at least) on a basic unassociative, head dependent, calculus. Thus, while the higher order types of Steedman are restrictive devices, those of Hendriks are permissive in that they allow structures which would not be generated through the basic lexical assignments.

4.1.1 Hendriks' approach

Hendriks addresses problems of Information Structure from a model theoretic perspective, in a number of papers published between 94 and 99, (see Hendriks [1999] for a collection of these works). His system addresses both the syntax, the prosodic phonology and the semantics of IP. In the present section we sketch some features of his typelogical treatment of the syntactic realization of IP in Catalan adopting data and terminology from Vallduvi [1990].

In Hendriks [1999], the analysis of the prosodically dominant English and syntactically dominant Catalan is developed on the basis of the dependency calculus of Moortgat and Morrill [1991], without structural rules. The *dependency calculus* (DC for short) has been developed as an headed variant of *NL*. The product '$\otimes$' operator is split into a left dominant '$\otimes_l$' and a right dominant '$\otimes_r$'. The resulting structures are 'headed trees':

> binary trees in which each mother node has a distinguished daughter: either the left or right subtree it immediately dominates is designated as *head*, and hence the other as *non-head*, or *dependent*.

The headed trees are used to encode metrical trees in Moortgat and Morrill [1991] and in Hendriks [1999] to deal with subsegmental phonology.

Lexical entries are triples formed by prosodic category, syntactic category, and semantic term. Prosodic terms are interpreted through a prosodic substitution function which unfolds the headed tree which is the antecedent of the prosodic deduction. The substitution splits the tree, along the dominance relations, in a dominant (focused) part and an unaccented and/or topical part ('forgetful mapping')[3]. All focus sentences are assumed to be the default case. Morphology and cliticization in Catalan and intonational boundaries in English represent what Hendriks calls *defocusing operators* which license informational partitions different from the default all focus one. These defocusing operators are higher order functors which perform dominance and word order transformations over the default structures.

As an example, we present the syntactic derivation of a Catalan link-focus-tail sentence:

(4.11) L′ amo l′ odi -a el broquil.
The boss it hate -s the broccoli.
The boss hates the broccoli.

The following dependency lexicon, from [Hendriks, 1999], accounts for the correct distribution of the informational component within the left dislocated sentence, as the reader can easily verify.

Example 42.

l′	→	$n/_{r}c$
amo	→	c
el	→	$n/_{r}c$
broquil	→	c
l′	→	$((n\backslash_{r}s)/_{l}n)/_{r}((n\backslash_{r}s)/_{r}n)$
odi−	→	$(s/_{r}n)/_{r}n$
−a	→	$((s/_{r}n)/_{r}n)\backslash_{l}((n\backslash_{r}s)/_{r}n)$

After spell out of the tree, the prosodic term results in the following, where *italics* denote the H* pitch accent (fuocus) on the sillable, and SMALL CAPS denote unaccented items (ground).

Prosody L′ AMO L′ *odi*A EL BROQUIL

4.1.2 Clitics and pregroups

Pregroups have been introduced in [Lambek, 2001]. A pregroup is

[3] The process reaches the subsegmental level in delivering the H*, L+H* pitch accents aligned with the strongest syllable in their domain and so providing a solution to the problem of *alignment*.

> an ordered monoid in which each element a has a *left adjoint* a^l such that:
>
> $$a^l a \leqslant 1 \leqslant a a^l$$
>
> and a *right adjoint* a^r such that:
>
> $$a a^r \leqslant 1 \leqslant a^r a$$

[Casadio and Lambek, 2001, Casadio, 2002] propose a logical approach to cliticization based on the system of *pregroups*. If we write the partial ordering $\leqslant$ as $\rightarrow$, as we do in type-logical grammar, the pregroup system is defined as follows.

Definition 67. The pregroup system.

$$\begin{array}{llll} \text{Reductions:} & a^l a \rightarrow 1 & & a a^r \rightarrow 1 \\ \text{Expansions:} & 1 \quad \rightarrow a a^l & & 1 \quad \rightarrow a^r a \end{array}$$

The process of recognition of a list of formulas consists in a series of contraction which finally ends in 1. To quote [Casadio and Lambek, 2001]:

> Fortunately, for the propose of sentence verification the expansions are not needed, but only the contractions [. . .]

Let us see how the pregroups system applies to linguistic analysis. We use the $\Rightarrow$ rewrite symbol in a way parallel to *CF* grammars, that is $\Gamma[\Delta] \Rightarrow \Gamma$ if $\Delta \rightarrow 1$ in the pregroup.

Example 43. Pregroup lexicon.

$$\begin{array}{ll} \textit{ama} & \rightarrow s n^l \\ \textit{raconta} & \rightarrow s n^l n^l \\ \textit{Mario} & \rightarrow n \\ \textit{lo} & \rightarrow s n^{ll} s^l \\ \textit{te} & \rightarrow s n^{ll} s^l \end{array}$$

Derivation of *ama Mario*:

$$s n^l \quad n \Rightarrow s$$

Derivation of *lo ama*:

$$s n^{ll} s^l \quad s n^l \Rightarrow s n^{ll} n^l \Rightarrow s$$

Derivation of *te lo racconta*:

$$sn^{ll}s^{l} \quad sn^{ll}s^{l} \quad sn^{l}n^{l} \Rightarrow sn^{ll}n^{ll}n^{l}n^{l} \Rightarrow sn^{ll}n^{l} \Rightarrow s$$

We do not enter here the details of cliticization in pregroup grammars and refer the reader to [Casadio and Lambek, 2001, Casadio, 2002] for an analysis of a wide range of Italian constructions involving cliticization. These analyses also account for the ordering constraints among cases, which we oversimplified. Observe, however, that we assigned 'higher order' types to clitics, as it is customary to treat pronouns in type-logical grammar. Then, the cluster *te-lo* is easily solved on the basis of the associative regime underlying pregroups grammar. [Degeilh and Preller, 2005] is another interesting work on pregroups which addresses also computational issues.

4.2 Italian clitic cluster and core clause

In this section, we look at Italian and we attempt to develope a grammar for clitic pronouns and the structure of the *core clause*. By core clause, we mean the complex constituted by the clitic cluster, the verb and non clitic arguments of the verb. For simplicity we will not take into consideration the subject of the verb nor nominative clitic, if any.

As we said before, if clitic pronouns occur, they attach tightly to the verbal head. If the verb is finite they precede it, if it is non finite they follow it. The occurrence of one or more clitics constitutes a *cluster*. The clitics in the cluster have a rigid order, which depends on their case. In order to provide a mathematical generalization, we assume that the argumets of the verb also occur in a fixed order, and that different orders obey other informational reasons which we do not investigate any further.

In the next section we introduce the abbreviations for case. This is meant to provide an informal notation which may help to identify their linear ordering and to speak about full noun phrase and clitic arguments of the verb. Our generalization can be anticipated as follows.

Proposition 12. The linear order of the clitic arguments of a verb v is the *mirror image* of the order of the full arguments of v.

On the basis of this generalization, which is confirmed by the examples that we consider[4], a general treatment for clitic clustering and core clause

[4] Although we are not considering *optional* clitics as the reflexive and the genitive.

structure is proposed. The proposal generalize over the number of arguments of the verb, those which are cliticized, and those which are full noun phrases. This approach boils down to the recoursive definition of a macro over structures of an indefinite number of arguments. Of course, lexical instances of this macro are bound by the number of arguments of a verb and by the number of clitics present.

4.2.1 Notation for cases

Clitics are marked for case, we use the following abbreviations for identifying the case:

O	Accusative	Direct object
I	Dative	Indirect object
L	Locative	Location

On full noun phrase arguments, case is expressed by prepositions, while clitics are morpho-phonologically marked for case.

The category that we will assign to the verb will contain no information about the case. Indeed, we want to identify a *natural order* of the verbal arguments that can be considered as the default case. The examples that follow are based on the lexicon given in 71. We represent in (b) congruous answers to the questions in (a). The structural adequacy of the answers is presented in terms of brackets. These structures show the syntactic outscoping of the focused element. The cases in (c) (if any) represent cliticized answers to the questions in (a). In (d) we give instances of left adjunction putting the extraposed constituent in square brackets.

(4.12) a. Cosa fa?
What does he do?

b. mette ((una moneta) ((in tasca) (a Mario)))
(*he*) *puts a coin in the pocket to Mario*.

b′. put (O (L I))

(4.13) a. A chi mette ((una moneta) (in tasca))?
To whom does he put a coin in the pocket?

b. (mette ((una moneta) (in tasca))) (a Mario)

b′. (put (O L)) I

c. (((ce la) mette) (a Mario))

c′. (((L O) put) I)

d. [(una moneta) (in tasca)] (((ce la) mette) (a Mario))

d′. [O L] (((L O) put) I)

(4.14) a. Dove mette ((una moneta) (a Mario))?
Where does he put a coin to Mario?

b. (mette ((una moneta) (a Mario))) (in tasca)

b′. (put (O I)) L

c. (((glie la) mette) (in tasca))

c′. (((I O) put) L)

d. [(una moneta) (a Mario)] (((glie la) mette) (in tasca))

d′. [O I] (((I O) put) L)

(4.15) a. Cosa mette ((in tasca) (a Mario))?
What does he put in the pocket to Mario?

b. (mette ((in tasca) (a Mario))) (una moneta)

b′. (put (L I)) O

c. (((gli ci) mette) (una moneta))

c′. (((I L) put) O)

d. [(in tasca) (a Mario)] (((gli ci) mette) (una moneta))

d′. [L I] (((I L) put) O)

(4.16) a. Chi mette ((una moneta) ((in tasca) (a Mario)))
Who puts a coin in the pocket to Mario?

b. Maria (mette ((una moneta) ((in tasca) (a Mario))))

b′. (put (O (L I)))

c. (((gli ce) la) mette)

c′. (((I L) O) put)

d. [O (L I)] (((I L) O) put)

Vallduvi [1990] and Hendriks [1999] claim that the relative order of the links (left adjoints) is not relevant. However, in question answer pairs like the previous ones, we find it preferable to maintain the order of the adjoints in the answer as it appears in the question.

What looks clear is that a case ordering can be identified for full arguments of the verb and especially for clitic arguments. We propose the following generalization.

Proposition 13. Case ordering for Italian.

$$\begin{array}{ll} \mathrm{O} \prec \mathrm{L} \prec \mathrm{I} & \text{full arguments} \\ \mathrm{I} \prec \mathrm{L} \prec \mathrm{O} & \text{clitic arguments} \end{array}$$

4.2.2 Formal treatment

We present here a categorial treatment of cliticization in *NL* that formalizes the discussion in the previous sections.

Definition 68. Let us define a sequence N_m, of m occurrences of the category n as follows:

$$\begin{array}{lll} N_1 & = & n \\ N_i & = & n \otimes N_{i-1}. \end{array}$$

We represent a sentential functor looking for m arguments of category n to the right to give a sentence as S_m. So

$$S_m = s/N_m$$

For example, $S_1 \equiv s/N_1 \equiv s/n$. For the semantics, we also write:

$$\begin{array}{lll} \pi_1 & = & \pi \\ \pi_n & = & \pi\pi_{n-1} \\ \pi'_1 & = & \pi' \\ \pi'_n & = & \pi'\pi'_{n-1} \end{array}$$

where π is the first projection and π' the second. Summing up, we have adopted the following convention.

Definition 69. Multi-argument verb[5].

$$s/N_n \quad S_n \quad \lambda x.(((S'_n\ \pi'_{n-1}\,x)\ \pi\pi'_{n-2}\,x)\ldots\pi\,x)$$

A verb subcategorizing for three arguments will have, according to proposition 13, as first argument an accusative noun phrase, as second a locative phrase and as third an indirect object. On the other hand, the bare typelogical structure of the sentential functor expresses only a series of n's. Thus, we assign S_1 to both *talks* and *love* and leave open the issue

[5] We remark that we are not considering the subject, for simplicity.

of expressing this kind of selectional restrictions (what can easily be done through subcategorizational devices in the style of [Bernardi, 2002]).

We use the macro $\mathscr{C} := s/(s/n)$ for clitics. A clitic cluster is a complex constituted by one or more clitics. We use the macro C_i for clusters of i clitics.

Definition 70. Clitic macro.

$$\begin{array}{rcl} C_1 & = & \mathscr{C} \\ C_n & = & C_{n-1} \otimes \mathscr{C} \end{array}$$

Observe that while the sequence N_i is right branching, clitic clustering is left branching. Indeed, this supports our claim that the structure of full arguments of the verb is *mirrored* by the clitic arguments. We propose the following lexicon.

Definition 71. Lexicon.

LEX	CAT	TERM
la	$s/(s/n)$	$\lambda z.(z\ x_i)$
gli	$s/(s/n)$	$\lambda z.(z\ x_i)$
ci	$s/(s/n)$	$\lambda z.(z\ x_i)$
infila	$S_3 \equiv s/(n \otimes (n \otimes n))$	$\lambda x.fill'\ \pi'_2 x\ \pi\pi' x\ \pi x$
una	n/c	$\exists$
in	n/c	$\lambda x.x$
a	n/c	$\lambda x.x$
moneta	c	*coin′*
tasca	c	*pocket′*

4.2.3 Clitic attachment

In type-logical grammar, clitic attachment is usually enforced by special multi-modal postulates. [Kraak, 1995] presents a sophisticated method for ensuring that no material occurs between the clitic pronoun and the verbal head. The method of [Kraak, 1995] can be extended to Italian with minor modifications. In particular in the case of Italian the analysis should account for the two positions in which the clitic may occur: preverbal, if the verb is finite, and post-verbal if the verb is non-finite.

We present here a simple method for achieving this result without the use of structural postulates. Our treatment will be based on an assignment to the empty string, which we denoted ϵ in the previous chapter. As we discussed before, the assignment of syntactic categories to the empty word is a common strategy in dealing with IP related phenomena. Furthermore, while postulates increase the generative power and the complexity of the logic, assignments to the empty word is harmless: it does not increase the generative power, nor the complexity of the logic. Thus, we develop a base-logical theory of clitic attachment.

The following generalized assignment captures the attachment of a clitic cluster C_i of i clitics to the left of a sentential functor S_n looking for n arguments to the right, providing a sentential functor looking for $n - i$ arguments to its right. It is understood that this macro type will have lexical instances only for $n \geqslant i$ and that n will be bound for any natural language grammar[6].

Definition 72. Finite clitic attachment.

LEX	CAT	MACRO
ϵ	$(C_i \otimes S_n)\backslash S_{n-i}$	$CC_{i,n}$

We may observe that if $i = n$, then the output is a full sentence $S_0 = s$ whose arguments are all saturated by clitic pronouns. If $i < n$, then S_0 will have other $n - i$ non clitic arguments. Consider the following example.

Example 44.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{gli \to \mathscr{C} \quad ce \to \mathscr{C}}{C_2 \to C_2} \quad lo \to \mathscr{C}}{C_3 \to C_3} \quad infila \to S_3}{C_3 \otimes S_3 \to C_3 \otimes S_3} \quad \epsilon \to (C_3 \otimes S_3)\backslash S_3}{(C_3 \otimes S_3) \otimes CC_{3,3} \to s}}{(((gli\ ce)\ lo)\ infila) \to s}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{gli \to \mathscr{C} \quad ci \to \mathscr{C}}{C_2 \to C_2} \quad infila \to S_3}{C_2 \otimes S_3 \to C_3 \otimes S_3} \quad \epsilon \to (C_2 \otimes S_3)\backslash S_1}{(C_2 \otimes S_3) \otimes CC_{2,3} \to S_1} \quad \dfrac{il \to n/c \quad libro \to c}{n/c \otimes c \to n}}{((C_2 \otimes S_3) \otimes CC_{2,3}) \otimes (n/c \otimes c) \to s}}{((gli\ ci)\ infila)\ (il\ libro) \to s}$$

[6] Indeed, we may even state that n will never be greater that 3, as one can hardly find verbs with more than 4 *arguments*.

The case of postverbal clitics with infinitive verbal heads is dealt as follows:

Definition 73. Non-finite clitic attachment. Let i be the primitive syntactic category of infinitivals and I_n be defined as S_n before. We have the following assignment for post-verbal cliticization.

LEX	CAT	MACRO
ϵ	$(I_n \otimes C_i)\backslash I_{n-i}$	$IC_{i,n}$

Example 45.

$$\dfrac{\dfrac{\dfrac{\mathit{infilar} \to I_3 \qquad \dfrac{\mathit{gli} \to \mathscr{C} \quad \mathit{ci} \to \mathscr{C}}{C_2 \to C_2}}{I_3 \otimes C_2 \to I_3 \otimes C_2} \qquad \epsilon \to IC_{2,3}}{(I_3 \otimes C_2) \otimes IC_{2,3} \to I_1} \qquad \dfrac{\mathit{il} \to n/c \quad \mathit{libro} \to c}{n/c \otimes c \to n}}{(\mathit{infilar}\ (\mathit{gli}\ \mathit{ci}))\ (\mathit{il}\ \mathit{libro}) \to s}$$

4.3 Clitic left-dislocation

As we saw in many examples from the previous sections, cliticization often occurs in combination with left dislocation. Indeed, if a clitic fills a verb argument, a full noun phrase, concordant under gender and number and indeed coreferential to the clitic, may occur at the left or right periphery. We only consider the case of the left periphery and limit our attention to what such a co-occurrence means for a resource sensitive grammar formalism as type-logical grammar. We refer to the bibliography given at the beginning of the chapter for an analysis of dislocation from an IP perspective.

4.3.1 Redundant arguments

The most simple instance of the problem we are going to address can be shown with the following example from Italian:

(4.17) Maria, la am -o.
Maria, her love I.
I love Maria.

We assume that the verb phrase *amo* is a syntactic object of category s/n, the subject position being already occupied by the subject through morphological composition of the stem 'am-' and the first person suffix '-o'. The semantic term corresponding to it is $\lambda x.(love'\ x\ S')$, where S' represents informally the speaker.

The proper name *Maria* is assigned the type n, so that *amo Maria* is a full sentence s whose 'meaning' is $love'\ M'\ S'$.

Clitic pronouns are expressions of type $((e \to t) \to t)$ represented in the semantics by the terms $\lambda P.(P\ x_i)$, which expresses the binding of a verb argument position with a free variable x_i.

Consider the following example lexicon

Example 46. Lexicon:

Maria	n	M'
amo	s/n	$\lambda x.(love'\ x\ S')$
la	$s/(s/n)$	$\lambda P.(P\ x_i)$

With this assignments, the expression *la amo* is analyzed as $s : love'\ x_i\ S'$. However, these lexical assignments do not allow to derive *Maria, la amo* as an expression of type s, since the sequent

(4.18) $n \otimes (s/(s/n) \otimes s/n) \to s$

is not derivable. This sequent violates a basic character of the resource sensitivity of type-logical grammar, namely the type balancing constraints of van Benthem [1991]. This can be noticed from the fact that two n's occur in negative position but only one in a positive position.

4.3.2 Proposals

In Sanfilippo [1990] the phenomenon of CLD is approached within the framework of Unification Categorial Grammar. The author claims that clitics "satisfy the subcategorizational requirements of the verb", but do not "reduce the thematic domain of the verb". Their semantic import to the thematic structure of the verb consists in adding further featural specifications "relative to the thematic entailments they instantiate and postpone the discharge of such entailments". The type-logical grammar system of Hendriks [1999] is built according to these lines. As we saw in example 42, agreement morphology and cliticization perform word order and head dependency exchanges: the clitic is a functor which does not saturate any argument of the verb, but performs the syntactic rearrangement of the cliticized verb argument to a preverbal (or postverbal) position, where the dislocated constituent occurs. With some simplification, Hendriks type for the clitic looks like the following:

$$(n\backslash s)/(s/n) : \lambda P \lambda x.(P\ x)$$

The sequent corresponding to the example 4.17, *Maria, la amo*, is

(4.19) $n \otimes ((n\backslash s)/(s/n) \otimes s/n) \rightarrow s : love'\ M'\ S'$

This lexical assignment expresses the dependency of the satellite from the clitic in the category of the cliticized clause, $n\backslash s$. The problem, then, is that a cliticized clause is not a full s. So, unless we provide multiple lexical assignments for the clitic, the optional status of the left extraposed phrase cannot be expressed.

As an alternative, one could maintain the full argument type assignment for the clitic and modify the type of the satellite to s/s.

(4.20) $s/s \otimes (s/(s/n) \otimes s/n) \rightarrow s : M'\ (love'\ x_i\ S')$

This gives the right result for the optionality of the extraposed phrase, but does not express the dependency between the satellite and the clitic. In the present case infact, the category of the satellite becomes that of a *sentential modifier* and this is a rather odd result from the semantic point of view as can be seen from the semantic representation arising with these assignments.

Reinhart [1982] and Vallduvi [1990] theories of information packaging assume that topics (or equivalently the links, depending on the terminology) indicate the address in the mental knowledge store where the semantic iterpretation of the sentence has to be located. In these theories, the semantic import of a *sentence with topic* can be represented as a *pair* constituted of the interpretation of the topic noun phrase and of the proposition which is the interpretation of the sentence.

In categorial terms such a structured object could be obtained by requiring, in particular contexts, the output sequent to be $n \otimes s$ rather that s. The following sequent can in fact be proved in *NL* without any change to the categories inolved.

(4.21) $n \otimes (s/(s/n) \otimes s/n) \rightarrow n \otimes s : \langle M', love'\ x_i\ S' \rangle$

Such a move is somehow what we need in order to balance the positive and negative occurrences of n types in the sequent. We notice anyway that this ouput type would admit non grammatical sentences, since it would license the occurrence of an n to the left of any sentence. We can improve this situation by admitting such an output only if the sentential term of the pair contains a free variable: the semantic component of the grammar could then capture the free variable occurring in the term via the well known methods proper to *dynamic semantics*. However, such an approach is rather intricate. We observe also that these lexical assignments do not express any relation of *dependence* between the satellite and the clitic.

In order to express the clitic-satelite dependency, the category of the clitic should express licensing conditions for the presence of the satellite. In categorial terms this can be expressed by having the clitic a left looking functor over n as output formula, for some formula b as right input, so $(n\backslash a)/b$. The fact that the clitic binds (and fully instantiate) an argument position of the verb is expressed by being b a sentential functor looking on the right for an argument of type n, namely $(n\backslash a)/(s/n)$. Now if we want a cliticized clause to preserve the character of a fully instantiated sentence, we cannot assume that a is the category s, as we discussed before in regard to Hendriks account. However, if a is the category $n \otimes s$, the output for a cliticized clause is $n\backslash(n \otimes s)$, which maintains the full instantiation of the verb argument and is related to s via the scheme

$$\lambda x \lambda y.\langle y, x\rangle : a \rightarrow b\backslash(b \otimes a)$$

which holds in the pure logic of residuation.

Thus, assuming that the clitic is assigned the syntactic category $(n\backslash(n \otimes s))/(s/n)$ and the semantic term $\lambda P \lambda y.\langle y, (P\ x_i)\rangle$ we have the following results. If the satellite does not occur, our output is something more specific than an s:

(4.22) $(n\backslash(n \otimes s))/(s/n) \otimes s/n \rightarrow n\backslash(n \otimes s) : \lambda y.\langle y, love'\ x_i\ S'\rangle$

Instead, if it does occur, we obtain the result in 4.21.

(4.23) $n \otimes ((n\backslash(n \otimes s))/(s/n) \otimes s/n) \rightarrow n \otimes s : \langle M', love'\ x_i\ S'\rangle$

4.3.3 Non-relevant reasoning

The problem exemplified in the previous sections is related to the structural properties of the syntactic calculus. Type balancing is a deep constraint of type-logical grammar, which reflects the fact that we can neither freely add resources, nor use them more than once (Contraction) in the derivation process. While various categorial approaches to pronouns and anaphora appeal to contraction (see Jäger [2001]), that is multiple use of the same resource, the phenomenon of CLD seems to involve the addition of a resource which is not in itself strictly required, and so is not *relevant*, in the sense of Relevant logic. In the case of CLD, a (clitic) pronoun and a noun phrase antecedent are involved. But while in the case of anaphora we have the antecedent playing its local role of noun phrase and a non local role of pronoun antecedent, the noun phrase in CLD plays no active role in the deduction. We notice, in particular, that if the clitic pronoun were an identity function (as in Hendriks' approach), no step of contraction would be required. If instead, we assign the clitic the full argument type, we find that the satellite occuring in the root sequent is not required. In other words, while in the case of anaphora a single resource accomplishes two roles, in CLD two resources play one and the same role.

We can abstractly formalize the kind of reasoning involved in CLD, with the following deduction which makes use of the weakening axiom $a \otimes b \to b$.

$$\dfrac{\dfrac{n \otimes (s/(s/n) \otimes s/n) \to s/(s/n) \otimes s/n \qquad \dfrac{s/(s/n) \to s/(s/n) \quad s/n \to s/n}{s/(s/n) \otimes s/n \to s}}{n \otimes (s/(s/n) \otimes s/n) \to s}}{\mathit{Maria}\ (\mathit{la\ amo}) \to s}$$

4.3.4 A lexical solution

Although a multi-modal type-logical grammar may contain rules for linear restructuring of the antecedent, it is quite rare to have rules of multiplication or reduction of the resources. Usually, the lexical type assignment allows to incorporate the non-linear forms structural reasoning which these rules provide in more powerful logics. In chapter 2, we saw several such cases, among which *reflexivization* performing an operation of multiple binding and so a form of contraction (identification of two arguments). Another one in the case of coordination, usually assigned the polymorphic category $(a\backslash a)/a$, in which two resources of the same type are consumed to produce the same category of the arguments.

In many categorial approaches to information structure (see in particular Steedman [2000b], Hendriks [1999]), the prosodic boundaries which mark the edges of the phonological phrases are assigned lexical categories in order to impose constraints on the prosodic structure (primarily expressed in terms of branching) of the surrounding context.

In the context of CLD, the satellite constitutes a phrasal unit, separated from the cliticized core clause by the LH% boundary tone of Pierrehumbert and Hirshberg [1990].

In what follows, we make use of a lexical assignment for the LH% boundary tone in order to solve the *non relevant* type equation discussed before. The category assigned to the boundary will resolve the occurrence of the clitic with that of the satellite. Since the prosodic boundary follows the satellite, it will be present only if the satellite occurs. This allows us to leave unchanged our primitive type assignments. The prosodic boundary will perform the reduction of the number of resources and semantically will provide the appropriate translation, in which the interpretation of the free variable of the clitic is bound to the context provided by the satellite.

If we concern only with the semantics, the type of the boundary LH%, for the present proposes, should be

(4.24) $((e \rightarrow t) \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t)$

If we assume the first argument is the satellite, and the second the clitic the following term would be appropriate for our proposes:

(4.25) $\lambda \mathscr{P} \lambda \mathscr{P}' \lambda P.[(\mathscr{P}' \; \lambda x.(\mathscr{P} \; \lambda y.x = y)) \wedge (\mathscr{P}' \; P)]$

This term would give the following translation for our example sentence:

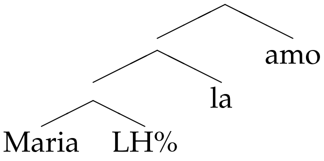
(4.26) $[x_i = M' \wedge love' \; x_i \; S']$

An immediate syntactic instantiation of this type could be the following.

(4.27) LH% : $(s/(n\backslash s))\backslash((s/(s/n))/(s/(s/n)))$

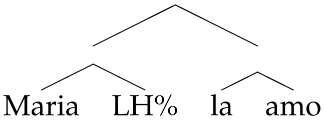
which would anyway give rise to the structure

(4.28)



This structure is still inadequate, since there is no evidence for left branching for CLD. Moreover, as we saw, the clitic attaches tightly to the verb host and forms with it a cliticized clause. The structure we want to obtain should be the folowing:

(4.29)



Such a tree can be obtained by uncurrying the previous assignment to the boundary tone, that is by transforming it into the following type.

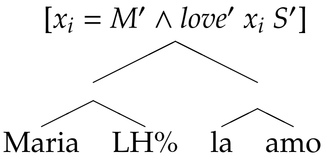
(4.30) LH% : $(s/(n\backslash s))\backslash(s/((s/(s/n)) \otimes (s/n)))$

The lambda term associated with this category is given below.

(4.31) $\lambda\mathscr{P}\lambda\boldsymbol{Q}.[(\pi\boldsymbol{Q}\ \lambda x.(\mathscr{P}\ \lambda y.x = y)) \wedge (\pi\boldsymbol{Q}\ \pi'\boldsymbol{Q})]$

Hence we obtain the following deduction:

(4.32)



This deduction is in accordance with our intuitions both on the syntactic/prosodic side and on the semantic/pragmatic side. Indeed, it elegantly solves the non-linear equations involved in the phenomenon of CLD in type-logical grammar. We wish to emphasize in particular the simplicity of this solution that relies entirely on the pure logic of residuation and on the power made available by a categorial lexicon.

Such a treatment may be extended to more complex constructions, but this is beyond the scope of this chapter which was meant to show that the pure logic of residuation is a powerful system for linguistic analysis and which can successfully be applied to complex linguistic phenomena.

4.4 Conclusion

In this chapter, we applied the non-associative Lambek calculus to linguistic analysis. In particular, we examined the phenomenon of cliticization and clitic left-dislocation. Our analysis relied upon all and only the tools of the pure logic of residuation. This means that no one of our solutions was dependent on structural postulates which increase the power of the system, or on special stipulations. Instead, we opted for a strongly lexical strategy, which involved assigning categories to empty words and prosodic boundaries.

We presented several options for a type-logical approach to CLD. We saw that in the phenomenon of CLD two lexical resources occur which have only one logical role. This motivated our claim for the non relevant reasoning involved in this kind of construction. Our proposal was to assign to the LH% boundary tone which follows the satellite an higher order type that solves the non linear type equation involved in CLD. In this respect, the boundary plays an essential role in the derivation, as it reduces the number of the resources involved and constrains the branching of the prosodic structure. Although the introduction of tone boundaries in the lexicon is partly questionable, due to the *sub- and super-segmental* character of prosodic phonology, this step allowed us to give an elegant solution to a non linear linguistic problem without abandoning the resource sensitive regime of *NL*. In fact, all the resources involved in our analysis have been taken from the *lexicon* and we have not changed the default assignment of the items involved in the deduction of the simple cliticized clause.

Chapter 5

Normal Derivations in $\mathcal{NL}$

In this chapter, we present a recognition method for sequents of the non-associative Lambek calculus based on the construction of *normal derivations*.

We start by limiting the discussion to *two-formula* sequents. The problem we are going to deal with can be stated as follows.

- Given a sequent $a \to c$, such that $a, c \in \mathcal{F}$, how do we prove whether $\vdash_{\mathcal{NL}} a \to c$?

The restriction to two-formula sequents indicates that the antecedent structure is given and expressed, here as in [de Groote, 1999], through the branching of the product formulas. It should be noted the difference between the problem that we are going to address here and the recognition problem as discussed in the previous chapter. Although the structure is already given, recognition of two-formula sequents of $\mathcal{NL}$ represents a non-trivial problem. In fact, the application of the rules of the calculus is non-deterministic, and we cannot easily discard options, as these may become relevant at later stages of the computation. Consider the following sequent.

(5.1) $a/(c/b \otimes b) \otimes (c/b \otimes b) \to a$

Assume that as soon as we encounter $c/b \otimes b$ at the right of the main connective, we apply the transition $c/b \otimes b \to c$ so that we replace c for that occurrence of $c/b \otimes b$ in $a/(c/b \otimes b) \otimes (c/b \otimes b) \to a$, obtaining $a/(c/b \otimes b) \otimes c \to a$. At this point we cannot simplify any further. In fact, the sequent in 5.1 is itself an instance of the scheme $x/y \otimes y \to x$, and that replacement has excluded the only possibility of reducing the input sequent. Of course, if we had tried immediately to apply the pattern $x/y \otimes y \to x$ to that sequent, we would have succeeded. However, this is not always possible, as embedded formulas may have to be reduced before, in order to make other external patterns available for reduction.

In this chapter, we present a method for constructing derivations of two -formula sequents. We will proceed as follows: we examine an important result from [Kandulski, 1988] about the construction of normal derivations in *NL*. Kandulski extended to *NL* the proof of the equivalence between non-associative Lambek grammars *without* product and context free grammars of [Buszkowski, 1986]. The normalization procedure used in these works has interesting properties also from the computational perspective. We provide a new method for constructing normal derivations in *NL*, adopting ideas from the method of Buszkowski and Kandulski.

Our construction generalizes that of Kandulski in applying also to the unary operators of [Moortgat, 1997b]. The procedure works *bottom-up*, namely from the leaves to the root, and its operations are always *simplifying* and *goal oriented*. These aspects guarantee termination and limit the non-determinism of the rules. We interpret the task of proving a (two-formula) sequent $a \rightarrow c$ as that of finding a formula b such that a and c 'simplify' to b. The notion of *simplification* concerns the length of the formulas involved and will be the central notion of the formal procedure. We will define two forms of simplification (or contraction), one from left to right, which we call *reduction* and one from right to left called *expansion*. The key intuition behind our work is that patterns called expanding, such as the so called lifting or co-application, are also simplifying, although the direction of the simplification is the reverse of the arrow symbol.

The system that we are going to design presents similarities to the one used in [Le Nir, 2004] for the compilation of a non-associative Ajdukiewicz Bar-Hillel grammar without product from a non-associative Lambek grammar without product. Indeed, our method allows to transform the computation of an *NL* grammar into the computation of an Ajdukiewicz Bar-Hillel grammar *with product* by finite lexical extension, enabling thus to indirectly apply the parsing methods discussed in chapter 3 to *NL* grammar. On the other hand, the presence of product formulas makes our formulation capable of handling with full generality proofs of sequents whose structure is given. We will see also that Le Nir method, discussed after definition 83 contains some mistakes.

In respect to Kandulski's and Buskowski's work, the normal derivations constructed by our system are free from spurious ambiguity, as we will prove in chapter 6.

5.1 Alternative formulations of *NL*

As we said in the introduction, we limit our present discussion to sequents made of two formulas. Hence, a *sequent* is a pair of formulas, which we write $a \to c$. In chapter 2, we presented *NL* with lambda term labeling. We propose it once more in a compact way and without lambda semantics. The double inference line in the residuation rules, is simply a shorthand indicating that the rules work in both directions. We call this axiomatization C, to distinguish it from the others that we will discuss later.

Definition 74. The system C.

For a, b and c ranging over formulas, we have

- **Identities**:

$$\begin{array}{cc} \text{Axioms} & \text{Cut} \\ a \to a & \dfrac{a \to b \quad b \to c}{a \to c} \end{array}$$

- **Residuation Rules**:

$$\dfrac{a \otimes b \to c}{\overline{a \to c/b}} \qquad \dfrac{a \otimes b \to c}{\overline{b \to a\backslash c}}$$

An example of derivation in C involving hypothetical reasoning is the following.

Example 47. We prove that $\vdash_C n \otimes (s/(n\backslash s))\backslash s \to s$.

$$\dfrac{\dfrac{\dfrac{\dfrac{n\backslash s \to n\backslash s}{n \otimes n\backslash s \to s}}{n \to s/(n\backslash s)} \quad \dfrac{\dfrac{(s/(n\backslash s))\backslash s \to (s/(n\backslash s))\backslash s}{s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s \to s}}{s/(n\backslash s) \to s/((s/(n\backslash s))\backslash s)}}{n \to s/((s/(n\backslash s))\backslash s)}}{n \otimes (s/(n\backslash s))\backslash s \to s}$$

This could be seen as a proof that the expression *John is missing* is a sentence if *John* is assigned the category n and *is missing* the category $(s/(n\backslash s))\backslash s$.

Characteristic theorems of *NL* are the following.

Definition 75.

AX^0, for all formulas a, c.

$$c/a \otimes a \rightarrow c \qquad a \otimes a\backslash c \rightarrow c$$
$$a \rightarrow (a \otimes c)/c \qquad a \rightarrow c\backslash(c \otimes a)$$
$$a \rightarrow c/(a\backslash c) \qquad a \rightarrow (c/a)\backslash c$$

Moreover, one can prove the following *derived rules* of inference.

Definition 76.

M^0, for every formula a and b.

$$\frac{a \rightarrow a'}{a \otimes b \rightarrow a' \otimes b} \otimes M \qquad \frac{b \rightarrow b'}{a \otimes b \rightarrow a \otimes b'} \otimes M$$

$$\frac{a' \rightarrow a}{b\backslash a' \rightarrow b\backslash a} \backslash M \qquad \frac{a' \rightarrow a}{a'/b \rightarrow a/b} /M$$

$$\frac{b \rightarrow b'}{b'\backslash a \rightarrow b\backslash a} \backslash M' \qquad \frac{b \rightarrow b'}{a/b' \rightarrow a/b} /M'$$

In fact, we can give a complete axiomatization of NL as follows.

Definition 77. Let AX denote the smallest set such that:

1. AX contains the axioms of C and AX^0.
2. AX is closed under rules M^0.
3. AX is closed under the following two rules:
   (a) if $a \rightarrow b \in AX$, $|a| > |b|$ and $b \rightarrow c \in AX$, $|b| > |c|$, then $a \rightarrow c \in AX$.
   (b) if $a \rightarrow b \in AX$, $|a| < |b|$ and $b \rightarrow c \in AX$, $|b| < |c|$, then $a \rightarrow c \in AX$.

Remark 2. In [Kandulski, 1988] one finds a similar construction of a set Ax of axioms. Indeed, our definition of AX results from Kandulski's Ax set by adding clause 3. This clause is added to include in the set AX sequents derivable by monotonous instances of cut, see also [Le Nir, 2004]. This extension does not affect Kandulski's argument. Although the difference should be borne in mind in the definitions and propositions that follow. The advantage of such an extension will become clear in the proof of the equivalence of definition 82 with the set AX. Roughly, we want to include in this set sequents like, for instance, $a/b \rightarrow a'/b'$ obtained by monotonous cut on the premises $a/b \rightarrow a'/b$ and $a'/b \rightarrow a'/b'$, since such sequents will be derived without using cut in definition 82.

Every sequent in AX can be derived in C. Moreover, we can prove the following.

Proposition 14. Every sequent derivable in C can be obtained from AX by means of the Cut rule only.

Proof. See [Kandulski, 1988]. □

Thus, let use define the following alternative axiomatization of *NL*.

Definition 78. Let K be the smallest set such that:

1. K contains AX.
2. K is closed under Cut.

From proposition 14, we know that K is equivalent to C. In the next section, we will examine the method of [Kandulski, 1988] for the construction of *normal derivations*. The construction is based on K. The reader might have noticed that K would be equivalent to C even without assuming $a \to c/(a\backslash c)$ and its symmetric as primitive axioms in AX, as they could be derived.

$$\frac{a \to (a \otimes a\backslash c)/(a\backslash c) \qquad \dfrac{a \otimes a\backslash c \to c}{(a \otimes a\backslash c)/(a\backslash c) \to c/(a\backslash c)}}{a \to c/(a\backslash c)}$$

The same holds for $\backslash M'$ and $/M'$.

$$\frac{a/b' \to (a/b' \otimes b)/b \qquad \dfrac{\dfrac{\dfrac{b \to b'}{a/b' \otimes b \to a/b' \otimes b'} \quad a/b' \otimes b' \to a}{a/b' \otimes b \to a}}{(a/b' \otimes b)/b \to a/b}}{a/b' \to a/b}$$

On the other hand, "it is expedient to have them"[1] in AX since they simplify the computation as we will see in more detail. [Kandulski, 1988] extended a result in [Buszkowski, 1986] for NL grammars without product to grammar based on the full non-associative Lambek calculus with product. The normalization procedure in the next section proves the reducibility of non-associative Lambek grammars with product formulas to *Ajdukiewicz Bar-Hillel* grammars with product formulas, and therefore the equivalence

[1] [Kandulski, 1988]

of non-associative Lambek grammars with product and context-free grammars. We will use Kandulski's method for the construction of normal derivations to design a regimented deductive system for automated theorem proving. The procedure has been suggested by the recursive definition of [Le Nir, 2004] for the compilation of the AB grammar (*without* product) inferable from a non-associative Lambek grammar *without* product. However, we avoid to fall into the inconsistencies of Le Nir construction (see the discussion after definition 83).

5.2 Normal derivations

The notion of normal derivation in [Kandulski, 1988] is defined for a logic whose sequents are of the form $\Gamma \rightarrow c$ where Γ is a tree structure with formulas on the leaves. We work with only one formula in the antecedent, therefore his argument applies also to our system. Moreover, the restriction to two-formula sequents does not imply that our system is weaker than the one used by Kandulski. It only means that we assume that the structure of our sequents is already given and expressed through the branching of products in the antecedent formula. As discussed in [de Groote, 1999], even the task of proving two formula sequents of *NL* is non-trivial from the computational perspective. Furthermore, provability of two formula sequents can be seen as a preliminary issue with respect to the problem of parsing, that is the problem of finding a proof for a sequence of input formulas and an output formula.

Definition 79. A two-formula sequent $a \rightarrow c$ from AX is called *expanding* (resp. *reducing*), if $|a| < |c|$ (resp. $|c| < |a|$).

Observe that all the elements of AX, except from the axioms, are either expanding or reducing.

A derivation of a sequent $a \rightarrow c$ in K can be seen as a sequence of formulas $x_0 \dots x_n$ (*derivation list*) such that

- $x_0 = a$,

- $x_n = c$ and

- for all i, $0 \leqslant i < n$, $x_i \rightarrow x_{i+1} \in AX$.

Example 48. Kandulski's construction, presented below, shows that a derivation of a sequent $a \rightarrow c$ in K can be configured as a derivation list $x_0 \dots x_k \dots x_n$ where

- $x_0 = a$,
- $x_n = c$,
- for all i, $0 \leqslant i < k$, $x_i \rightarrow x_{i+1}$ are reducing patterns from AX and
- for all j, $k \leqslant j < n$, $x_j \rightarrow x_{j+1}$ are expanding patterns from AX.

The main property of AX (which in the original formulation did not include the sequents obtained by clause 3 in the construction of definition 77) is proved in the following proposition, from [Kandulski, 1988].

Proposition 15. Given formulas x, y and z, $x \neq z$ and both $x \rightarrow y$ and $y \rightarrow z$ are in AX, then if $x \rightarrow y$ and $y \rightarrow z$ are expanding and reducing, respectively, then there is a formula y' such that $|y'| < |y|$ and $x \rightarrow y'$ and $y' \rightarrow z$ are in AX.

Proof. see [Kandulski, 1988] for *NL*, the proof for the system without product was given in [Buszkowski, 1986]. □

Example 49. Some relevant instances of the proof of proposition 15 are given below.

If $a \rightarrow (a \otimes a\backslash b)/(a\backslash b)$ and $(a \otimes a\backslash b)/(a\backslash b) \rightarrow b/(a\backslash b)$ are in AX, then $y' = b/(a\backslash b)$.

If $a \rightarrow (a \otimes b)/(a\backslash(a \otimes b))$ and $(a \otimes b)/(a\backslash(a \otimes b)) \rightarrow (a \otimes b)/b$ are in AX, then $y' = (a \otimes b)/b$.

If $a \otimes a\backslash b \rightarrow b/(a\backslash b) \otimes a\backslash b$ and $b/(a\backslash b) \otimes a\backslash b \rightarrow b$ are in AX, then $y' = a \otimes a\backslash b$.

Proposition 15 guarantees that for every sequent $a \rightarrow c$ derivable in *NL* has a derivation $D = x_0 \ldots x_n$ in K such that no sublist $x_{i-1}\, x_i\, x_{i+1}$, $0 < i < n$ of D is such that $|x_{i-1}| < |x_i|$ and $|x_i| > |x_{i+1}|$. Hence, *normal* derivations are defined as follows.

Definition 80. A derivation $D = x_0 \ldots x_n$ of a sequent $a \rightarrow c$ in K is *normal* if and only if for no k, $0 < k < n$, there is a sublist $x_{k-1}\, x_k\, x_{k+1}$ of D such that $|x_{k-1}| < |x_k| > |x_{k+1}|$[2].

Let $|x_0 \ldots x_n| = |x_0| + \ldots + |x_n|$.

Definition 81. A derivation D of $a \rightarrow c$ in K is minimal if and only if for all the derivations D' of $a \rightarrow c$ in K, $|D| \leqslant |D'|$.

[2] We used the notation $x < y > z$ for $x < y$ and $y > z$.

Remark 3. We note that a derivations $x_0 \dots x_n$ such that for some i, $0 \leqslant i < n$, $x_i \equiv x_{i+1}$ cannot be minimal. In fact, whenever a derivation is of the form $x_0 \dots x_i\, x_{i+1} \dots x_n$ with $x_i \equiv x_{i+1}$, we can replace it with $x_0 \dots x_i \dots x_n$, obtaining a shorter one.

Proposition 16. Each minimal derivation is normal.

Proof. [Kandulski, 1988]: induction on $D = x_0 \dots x_n$. Assume that D is minimal. There are two cases:

If $x_{n-1} \to x_n$ is expanding, then D is normal by induction hypothesis.

If $x_{n-1} \to x_n$ is reducing, one reasons by contraposition. □

Given Kandulski's construction for normal derivations, the process of finding a proof for a sequent $a \to c$ can be divided into two subprocesses of *contraction*, as illustrated in example 48. One *from left to right*, corresponding to the process of finding the formula to which the antecedent formula *reduces*. And one *from right to left*, corresponding to the process of finding the formula which *expands* to the succedent formula. The advantage of this strategy is that the two processes proceed in a *monotonic decreasing* way: every formula encountered in each of the two processes results by application of a transition axioms from AX, expressing a form of contraction, to the previous formula.

Example 50. The derivation of the sequent $a \otimes a\backslash b \to c/(b\backslash c)$ results in the composition of the two sequents $a \otimes a\backslash b \to b$ and $b \to c/(b\backslash c)$ from AX. One has the following derivation:

$$a \otimes a\backslash b,\ b,\ c/(b\backslash c)$$

5.3 Automatic construction of normal derivations

In this section, we present an alternative characterization of normal derivations for NL. Our system will be partially based on the construction of Kandulski and Buszkowski. Indeed, from the computational perspective, our definition can be seen as a characterization with finite means of the set AX. Hence as an algorithm for automatic theorem proving. Observe, in fact, that the set AX is infinite and cannot be used straightforwardly to build a derivation automatically.

In addressing the question of automated theorem proving, some preliminary issues have to be solved. Among others, the problem of the *instantiation* of the axiom and transition schemes, and, closely related, the direction of the proof construction process.

In the search of a proof for a sequent $a \to c$, we use only a *finite* number of elements of AX to perform the transitions from a to c and the choice of these elements is dictated by what is in a and in c.

Concerning the direction of the search procedure one may work from the leaves to the root (*bottom-up*) or from the root to the leaves (*top-down*). A bottom-up search strategy guarantees that every stage of the process will contain only valid sequents. This is not guaranteed in every top-down approach, witness

$$\frac{(a/b)\backslash a \not\to b}{a/b \to a/((a/b)\backslash a)}$$

which is a possible way of unfolding the root sequent though $/M$. Therefore, our procedure will construct derivations from the leaves to the root, avoiding thus to ever encounter invalid sequents.

The key intuition underlying our method is the following. One may observe that *expanding* schemes such as lifting, $a \to b/(a\backslash b)$, or coapplication, $a \to (a \otimes b)/b$, simplify formulas in the same way as the reducing scheme $a{\otimes}a\backslash b \to b$ does, though in opposite direction. In the case of $a{\otimes}a\backslash b \to b$ one sees b as the result of contracting $a \otimes a\backslash b$. Instead, in the case of $a \to b/(a\backslash b)$ or $a \to (a \otimes b)/b$ one can see a as the result of contracting $b/(a\backslash b)$ or $(a \otimes b)/b$. What we want to emphasize is the fact that expanding patterns can be seen as *right-to-left* transition rules, while the reducing patterns as *left-to-right* transition rules.

We wish also to remark with respect to Kandulski's work that our construction method is not affected by spurious ambiguity. We will examine this aspect in chapter 6.

5.3.1 Expansion and reduction

In this section, we present the core routines of our recognition method and prove their correctness. We are going to define two functions e and r of type $\mathcal{F} \to \{\mathcal{F}\}$. To make more compact the presentation below, we use the construct

$$\text{let } x \text{ be } v \text{ in } t$$

which is another way of expressing the substitution of v for x in t, denoted before as $t[x := v]$[3]. Moreover, we label the clauses in the algorithm as to simplify reference to them in the proofs which follow. To avoid any possible source of misunderstanding, subclauses (a), (b) and (c) in clause 3) and 2′) are interleaved by *union*, ∪. Furthermore, the operations of subclauses (b) and (c) are simple *maps* and not closures[4].

Definition 82. The functions *expand*, e, and *reduce*, r (we omitted the symmetric cases):

$$\begin{array}{llll}
1) & e(a) & = & \{a\}, \text{ if } a \text{ is an atom} \\
2) & e(a \otimes b) & = & \{\, a' \otimes b' \mid a' \in e(a) \;\&\; b' \in e(b) \,\} \\
3) & e(a/b) & = & \text{let } mon \text{ be } \{\, a'/b' \mid a' \in e(a) \;\&\; b' \in r(b) \,\} \text{ in} \\
 & \text{(a)} & & \qquad mon \\
 & & & \qquad \cup \\
 & \text{(b)} & & \qquad \{\, c \mid (c \otimes b')/b' \in mon \,\} \\
 & & & \qquad \cup \\
 & \text{(c)} & & \qquad \{\, c \mid a'/(c\backslash a') \in mon \,\} \\
1') & r(a) & = & \{a\}, \text{ if } a \text{ is an atom} \\
2') & r(a \otimes b) & = & \text{let } mon \text{ be } \{\, a' \otimes b' \mid a' \in r(a) \;\&\; b' \in r(b) \,\} \text{ in} \\
 & \text{(a)} & & \qquad mon \\
 & & & \qquad \cup \\
 & \text{(b)} & & \qquad \{\, c \mid c/b' \otimes b' \in mon \,\} \\
 & & & \qquad \cup \\
 & \text{(c)} & & \qquad \{\, c \mid a' \otimes a'\backslash c \in mon \,\} \\
3') & r(a/b) & = & \{\, a'/b' \mid a' \in r(a) \;\&\; b' \in e(b) \,\}
\end{array}$$

Observe that the sets $e(x)$ and $r(x)$ are *finite* for every formula x as all their elements are shorter of x or identical to x.

[3] The so called *let* construct is borrowed from functional programming. It offers a simple way of assigning a value v to a variable x and of using x with such a value v into the body t. While it makes more concise the formulation, it makes also clear that functions e and r are in fact a functional program, with some syntactic sugaring.

[4] This means that if, for instance, the formula c returned by clause 3b) or 3c) is itself of the form $(y \otimes x)/x$ or $x/(y\backslash x)$, **no** further contraction is applied to c.

Example 51. We calculate some reductions and expansions showing the trace of the recursion as a tree.

$r((s/(n\backslash s))\backslash s)$:

$$\dfrac{r(s) = \{s\} \qquad \dfrac{e(s) = \{s\} \qquad \dfrac{r(s) = \{s\} \quad e(n) = \{n\}}{r(n\backslash s) = \{n\backslash s\}}}{e(s/(n\backslash s)) = \{s/(n\backslash s),\ n\}}}{r((s/(n\backslash s))\backslash s) = \{(s/(n\backslash s))\backslash s,\ n\backslash s\}}$$

$e((s/(n\backslash s))\backslash s)$:

$$\dfrac{e(s) = \{s\} \qquad \dfrac{r(s) = \{s\} \qquad \dfrac{e(s) = \{s\} \quad r(n) = \{n\}}{e(n\backslash s) = \{n\backslash s\}}}{r(s/(n\backslash s)) = \{s/(n\backslash s)\}}}{e((s/(n\backslash s))\backslash s) = \{(s/(n\backslash s))\backslash s,\ n\backslash s\}}$$

$r((a \otimes a\backslash c)/b \otimes b)$:

$$\dfrac{\dfrac{\dfrac{r(a) = \{a\} \qquad \dfrac{r(c) = \{c\} \quad e(a) = \{a\}}{r(a\backslash c) = \{a\backslash c\}}}{r(a \otimes a\backslash c) = \{a \otimes a\backslash c,\ c\}} \qquad e(b) = \{b\}}{r((a \otimes a\backslash c)/b) = \{(a \otimes a\backslash c)/b,\ c/b\}} \qquad r(b) = \{b\}}{r((a \otimes a\backslash c)/b \otimes b) = \{(a \otimes a\backslash c)/b \otimes b,\ c/b \otimes b,\ a \otimes a\backslash c,\ c\}}$$

Correctness of definition 82

Let us introduce the following abbreviations.

Notation 1.

$x \overset{e}{\to} y \ := \ x \to y \in AX$ and $|x| < |y|$ or $x \equiv y$.

$x \overset{r}{\to} y \ := \ x \to y \in AX$ and $|x| > |y|$ or $x \equiv y$.

We assume now, for the sake of exposition, that the following equivalences hold.

- $e(y) = \{ \ x \mid x \overset{e}{\to} y \ \}$.
- $r(x) = \{ \ y \mid x \overset{r}{\to} y \ \}$.

We will formally prove these equivalences in proposition 18 and 19. We describe each step of definition 82.

Definition 82 is a recursive definition. Function e and r return the singleton set on atomic input, clauses 1) and 1′). The other clauses have non-atomic input formulas. Let us examine clause 3), for all. The input of e is a/b. The function calls $e(a)$ and $r(b)$ and stores in the variable *mon* all the a'/b' such that $a' \in e(a)$ and $b' \in r(b)$. As we assumed, and will prove in proposition 18, if $a' \in e(a)$, then $a' \xrightarrow{e} a$ and if $b' \in r(b)$, then $b \xrightarrow{r} b'$. Thus, $a'/b' \xrightarrow{e} a/b'$ and $a/b' \xrightarrow{e} a/b$ by clause 2 of definition 77. Hence, $a'/b' \xrightarrow{e} a/b$ by clause 3 of definition 77 (an instance of monotonous cut). Instead of using monotonous cuts, the generation of the set *mon* is based on the following rule, from [Moortgat and Oehrle, 1997].

$$\frac{a' \to a \quad b \to b'}{a'/b' \to a/b}$$

The set *mon* is added in clause 3a) of algorithm 82 to two other sets, generated in 3b) and 3c). The set in 3b) contains all the c's such that $(c \otimes b')/b'$ is an element of *mon*. While the set in 3c) contains all the c's such that $a'/(c\backslash a')$ is an element of *mon*. The inferences captured in 3b) and 3c) could be expressed by the following rule.

$$\frac{y \to x}{z \to x} \quad \text{if } z \to y \in AX^0 \text{ and } |z| < |y|$$

For example, assume that $a'/(c\backslash a') \in e(x)$ by clause 3a). Then $a'/(c\backslash a') \xrightarrow{e} x$ by assumption. The set AX contains the expanding sequent $c \to a'/(c\backslash a')$, by clause 1) of definition 77, and is closed under monotonous cuts by clause 3) of definition 77. Thus $c \xrightarrow{e} x$. We conclude $c \in e(x)$.

The other clauses are similar. Some output set consists only of the inductive case, as in clauses 2) and 3′). Instead, the set in clause 2′), as the one in 3), results from an inductive clause generating a set *mon* and two inference steps, which can be captured by the inference rule below.

$$\frac{x \to z}{x \to y} \quad \text{if } z \to y \in AX^0 \text{ and } |y| < |z|$$

Clearly the algorithm draws inferences from the premises to the conclusion: a complex problem, represented by a non-atomic input formula, is divided into two subproblems. The sets of solutions for the subproblems provide the premises for the solution of the problem.

In order to prove the correctness of the functions e and r, we shall prove the following equivalences.

- $x \in e(y)$ iff $x \xrightarrow{e} y$.

- $y \in r(x)$ iff $x \xrightarrow{r} y$.

We refer to the 'if' direction as completeness and to the 'only if' direction as soundness. In each case of the following analyses, we omit the symmetric cases. We start by proving the identity case.

Proposition 17. For all x, $x \in r(x)$ and $x \in e(x)$.

Proof. Induction on x. If x is atomic, then $x \in r(x)$ and $x \in e(x)$. Otherwise $x \equiv y\sharp z$, $\sharp \in \{\otimes, /, \backslash\}$. By IH, $y \in r(y)$ and $y \in e(y)$, and $z \in r(z)$ and $z \in e(z)$, we conclude $y\sharp z \in r(y\sharp z)$ and $y\sharp z \in e(y\sharp z)$. □

Soundness of definition 82 is proved as follows.

Proposition 18. Soundness:

(A) If $y \in r(x)$, then $x \xrightarrow{r} y$.

(B) If $y \in e(x)$, then $y \xrightarrow{e} x$.

Proof. Induction on x. If x is atomic, we have $x \xrightarrow{r} x$ and $x \xrightarrow{e} x$ by clause 1) of definition 77. Otherwise we have the following cases.

Proof of (A):

1. $x \equiv x' \otimes x^*$

   (a) $y \equiv y' \otimes y^*$, with $y' \in r(x')$ and $y^* \in r(x^*)$. By IH, $x' \xrightarrow{r} y'$ and $x^* \xrightarrow{r} y^*$. By clause 2) of definition 77, we have $x' \otimes x^* \xrightarrow{r} y' \otimes x^*$ and $y' \otimes x^* \xrightarrow{r} y' \otimes y^*$. Hence, $x' \otimes x^* \xrightarrow{r} y' \otimes y^*$ by clause 3a) of definition 77.

   (b) Otherwise $y/z \otimes z \in r(x' \otimes x^*)$, with $y/z \in r(x')$ and $z \in r(x^*)$. We obtain $x' \otimes x^* \xrightarrow{r} y/z \otimes z$ like in the previous case. Since we have $y/z \otimes z \xrightarrow{r} y$ by clause 1) of definition 77, we conclude $x' \otimes x^* \xrightarrow{r} y$ by clause 3a) of definition 77.

2. If $x \equiv x'/x^*$, then $y \equiv y'/y^*$, with $y' \in r(x')$ and $y^* \in e(x^*)$. By IH, $x' \xrightarrow{r} y'$ and $y^* \xrightarrow{e} x^*$. By clause 2) of definition 77, we have $x'/x^* \xrightarrow{r} y'/x^*$ and $y'/x^* \xrightarrow{r} y'/y^*$. Hence, $x'/x^* \xrightarrow{r} y'/y^*$ by clause 3a) of definition 77.

Proof of (B):

1. If $x \equiv x' \otimes x^*$, then $y \equiv y' \otimes y^*$, with $y' \in e(x')$ and $y^* \in e(x^*)$. By IH, $y' \overset{e}{\rightarrow} x'$ and $y^* \overset{r}{\rightarrow} x^*$. By clause 2) of definition 77, we have $y' \otimes y^* \overset{e}{\rightarrow} y' \otimes x^*$ and $y' \otimes x^* \overset{e}{\rightarrow} x' \otimes x^*$. Hence, $y' \otimes y^* \overset{e}{\rightarrow} x' \otimes x^*$ by clause 3b) of definition 77.

2. If $x \equiv x'/x^*$

   (a) $y \equiv y'/y^*$, with $y' \in e(x')$ and $y^* \in r(x^*)$. By IH, $y' \overset{e}{\rightarrow} x'$ and $x^* \overset{r}{\rightarrow} y^*$. By clause 2) of definition 77, we have $y'/y^* \overset{e}{\rightarrow} x'/y^*$ and $x'/y^* \overset{e}{\rightarrow} x'/x^*$. Hence, $y'/y^* \overset{e}{\rightarrow} x'/x^*$ by clause 2) of definition 77.

   (b) Let $(y \otimes v)/v \in e(x'/x^*)$, with $y \otimes v \in e(x')$ and $v \in r(x^*)$. We obtain $(y \otimes v)/v \overset{e}{\rightarrow} x'/x^*$ like in the previous case. By clause 1) of definition 77, $y \overset{e}{\rightarrow} (y \otimes v)/v$. We conclude $y \overset{e}{\rightarrow} x'/x^*$ by clause 3b) of definition 77.

   (c) Otherwise, let $v/(y\backslash v) \in e(x'/x^*)$, with $v \in e(x')$ and $y\backslash v \in r(x^*)$. We obtain $v/(y\backslash v) \overset{e}{\rightarrow} x'/x^*$ like in the previous case. By clause 1) of definition 77, $y \overset{e}{\rightarrow} v/(y\backslash v)$. Hence, $y \overset{e}{\rightarrow} x'/x^*$ by clause 3b) of definition 77.

□

Completeness of definition 82 is proved as follows.

Proposition 19. Completeness:

(A) If $x \overset{r}{\rightarrow} y$, then $y \in r(x)$.

(B) If $y \overset{e}{\rightarrow} x$, then $y \in e(x)$.

Proof. Induction on the *AX* derivation. If $x \equiv y$, then (A) and (B) hold by proposition 17. Otherwise:

Proof of (A):

1. $x \overset{r}{\rightarrow} y \equiv b/a \otimes a \rightarrow b$. By proposition 17, $b/a \otimes a \in r(b/a \otimes a)$. Hence $b \in r(b/a \otimes a)$ by clause 2'b) of definition 82.

2. $x \overset{r}{\rightarrow} y$ is obtained by clause 2) of definition 77. We have the following subcases.

   (a) $x \overset{r}{\rightarrow} y \equiv a \otimes b \overset{r}{\rightarrow} a' \otimes b$ and $a \overset{r}{\rightarrow} a'$. By IH, $a' \in r(a)$. By proposition 17, $b \in r(b)$. Hence, $a' \otimes b \in r(a \otimes b)$ by clause 2'a) of definition 82.

(b) $x \xrightarrow{r} y \equiv a/b \xrightarrow{r} a'/b$ and $a \xrightarrow{r} a'$. By IH, $a' \in r(a)$. By proposition 17, $b \in e(b)$. Hence, $a'/b \in r(a/b)$ by clause 3′) of definition 82.

(c) $x \xrightarrow{r} y \equiv a/b \xrightarrow{r} a/b'$ and $b' \xrightarrow{e} b$. By IH, $b' \in e(b)$. By proposition 17, $a \in r(a)$. Hence, $a/b' \in r(a/b)$ by clause 3′) of definition 82.

3. $x \xrightarrow{r} y$ is obtained by clause 3a) of definition 77. We shall consider the following subderivations.

(a)

$$\dfrac{\dfrac{x' \xrightarrow{r} y'}{x'/x^* \xrightarrow{r} y'/x^*} \qquad \dfrac{y^* \xrightarrow{e} x^*}{y'/x^* \xrightarrow{r} y'/y^*}}{x'/x^* \xrightarrow{r} y'/y^*}$$

By IH, $y' \in r(x')$ and $y^* \in e(x^*)$. Then, by clause 3′) of definition 82, $y'/y^* \in r(x'/x^*)$.

(b)

$$\dfrac{\dfrac{x' \xrightarrow{r} y'}{x' \otimes x^* \xrightarrow{r} y' \otimes x^*} \qquad \dfrac{x^* \xrightarrow{r} y^*}{y' \otimes x^* \xrightarrow{r} y' \otimes y^*}}{x' \otimes x^* \xrightarrow{r} y' \otimes y^*}$$

By IH, $y' \in r(x')$ and $y^* \in r(x^*)$. Then, by clause 2′a) of definition 82, $y' \otimes y^* \in r(x' \otimes x^*)$.

(c)

$$\dfrac{\dfrac{\dfrac{x' \xrightarrow{r} y/z}{x' \otimes x^* \xrightarrow{r} y/z \otimes x^*} \qquad \dfrac{x^* \xrightarrow{r} z}{y/z \otimes x^* \xrightarrow{r} y/z \otimes z}}{x' \otimes x^* \xrightarrow{r} y/z \otimes z} \qquad y/z \otimes z \xrightarrow{r} y}{x' \otimes x^* \xrightarrow{r} y}$$

By IH, $y/z \in r(x')$ and $z \in r(x^*)$. Then, by clause 2′a) of definition 82, $y/z \otimes z \in r(x' \otimes x^*)$ and by clause 2′b) $y \in r(x' \otimes x^*)$.

Proof of (B):

1. $y \xrightarrow{e} x \equiv a \xrightarrow{e} (a \otimes b)/b$. By proposition 17, $(a \otimes b)/b \in e((a \otimes b)/b)$. Hence $a \in e((a \otimes b)/b)$ by clause 3b) of definition 82.

2. $y \xrightarrow{e} x \equiv a \xrightarrow{e} b/(a\backslash b)$. By proposition 17, $b/(a\backslash b) \in e(b/(a\backslash b))$. Hence $a \in e(b/(a\backslash b))$ by clause 3c) of definition 82.

3. $y \xrightarrow{e} x$ is obtained by clause 2) of definition 77. We have the following subcases.

   (a) $y \xrightarrow{e} x \equiv a' \otimes b \xrightarrow{e} a \otimes b$ and $a' \xrightarrow{e} a$. By IH, $a' \in e(a)$. By proposition 17, $b \in e(b)$. Hence, $a' \otimes b \in e(a \otimes b)$ by clause 2) of definition 82.

   (b) $y \xrightarrow{e} x \equiv a'/b \xrightarrow{e} a/b$ and $a' \xrightarrow{e} a$. By IH, $a' \in e(a)$. By proposition 17, $b \in r(b)$. Hence, $a'/b \in e(a/b)$ by clause 3a) of definition 82.

   (c) $y \xrightarrow{e} x \equiv a/b' \xrightarrow{e} a/b$ and $b \xrightarrow{r} b'$. By IH, $b' \in r(b)$. By proposition 17, $a \in e(a)$. Hence, $a/b' \in e(a/b)$ by clause 3a) of definition 82.

4. $y \xrightarrow{e} x$ is obtained by clause 3b) of definition 77. We shall consider the following subderivations.

   (a)

$$\dfrac{\dfrac{y' \xrightarrow{e} x'}{y' \otimes y^* \xrightarrow{e} x' \otimes y^*} \qquad \dfrac{y^* \xrightarrow{e} x^*}{x' \otimes y^* \xrightarrow{e} x' \otimes x^*}}{y' \otimes y^* \xrightarrow{e} x' \otimes x^*}$$

   By IH, $y' \in e(x')$ and $y^* \in e(x^*)$. Then, by clause 2) of definition 82, $y' \otimes y^* \in e(x' \otimes x^*)$.

   (b)

$$\dfrac{\dfrac{y' \xrightarrow{e} x'}{y'/y^* \xrightarrow{e} x'/y^*} \qquad \dfrac{x^* \xrightarrow{r} y^*}{x'/y^* \xrightarrow{e} x'/x^*}}{y'/y^* \xrightarrow{e} x'/x^*}$$

   By IH, $y' \in e(x')$ and $y^* \in r(x^*)$. Then, by clause 3a) of definition 82, $y'/y^* \in e(x'/x^*)$.

   (c)

$$\dfrac{y \xrightarrow{e} (y \otimes z)/z \qquad \dfrac{\dfrac{y \otimes z \xrightarrow{e} x'}{(y \otimes z)/z \xrightarrow{e} x'/z} \qquad \dfrac{x^* \xrightarrow{r} z}{x'/z \xrightarrow{e} x'/x^*}}{(y \otimes z)/z \xrightarrow{e} x'/x^*}}{y \xrightarrow{e} x'/x^*}$$

   By IH, $y \otimes z \in e(x')$ and $z \in r(x^*)$. Then, by clause 3a) of definition 82, $(y \otimes z)/z \in e(x'/x^*)$ and by clause 3a) $y \in e(x'/x^*)$.

(d)

$$\dfrac{y \xrightarrow{e} z/(y\backslash z) \qquad \dfrac{\dfrac{x^* \xrightarrow{r} y\backslash z}{z/(y\backslash z) \xrightarrow{e} z/x^*} \quad \dfrac{z \xrightarrow{e} x'}{z/x^* \xrightarrow{e} x'/x^*}}{z/(y\backslash z) \xrightarrow{e} x'/x^*}}{y \xrightarrow{e} x'/x^*}$$

By IH, $z \in e(x')$ and $y\backslash z \in r(x^*)$. Then, by clause 3a) of definition 82, $z/(y\backslash z) \in e(x'/x^*)$ and by clause 3c) $y \in e(x'/x^*)$.

□

We have proved the equivalence of the set of reducing and expanding sequents of AX with the set r and e from definition 82, respectively. By construction, AX is closed under monotonous cut. In turn, we have that, if $y \in w(x)$ and $z \in w(y)$, then $z \in w(x)$, where $w \in \{e, r\}$. We express this as follows.

Proposition 20.

If $y \in r(x)$, then $r(y) \subseteq r(x)$.

If $y \in e(x)$, then $e(y) \subseteq e(x)$.

Proof. Immediate after proposition 18 and 19. □

What still remains to be done is to link two sets of expanding and reducing sequents generated by the functions e and r to provability in K. As AX is closed under monotonous cuts, and we know from proposition 16 that normal derivations in K are structured as lists of the form $x_0 \ldots x_k \ldots x_n$, where the sublist $x_0 \ldots x_k$ consists of a series of reducing inferences and the sublist $x_k \ldots x_n$ consists of a series of expanding inferences, there remains only one instance of cut in K which we need to use for general theorem proving. That is the cut between a left reducing and a right expanding premise. This can be expressed as follows.

Proposition 21.

$$\vdash_{NL} a \rightarrow c \quad \text{iff} \quad r(a) \cap e(c) \neq \emptyset.$$

Proof.

If part: Let $b \in r(a) \cap e(c)$. Then $b \in r(a)$ and $b \in e(c)$. Hence $a \xrightarrow{r} b$ and $b \xrightarrow{e} c$, by proposition 18.

Only if part: If $\vdash_{NL} a \longrightarrow c$, then, by proposition 16, there is a normal derivation of $x_0 \dots x_k \dots x_n$, such that $a = x_0$, $c = x_n$ and for all i, $0 \leqslant i < k$, $x_i \overset{r}{\rightarrow} x_{i+1}$, and for all j, $k \leqslant j < n$, $x_j \overset{e}{\rightarrow} x_{j+1}$. By proposition 19 and proposition 20, $x_k \in r(x_0)$ and $x_k \in e(x_n)$. □

Example 52.

We prove that $\vdash_{NL} a \otimes a\backslash b \longrightarrow c/(b\backslash c)$ as follows.

$$r(a \otimes a\backslash b) \cap e(c/(b\backslash c)) = \{a \otimes a\backslash b,\ b\} \cap \{c/(b\backslash c),\ b\} = \{b\}$$

The following application of proposition 21 concludes a example 51.

$$\begin{array}{l} r((s/(n\backslash s))\backslash s) \cap e((s/(n\backslash s))\backslash s) = \\ \{(s/(n\backslash s))\backslash s,\ n\backslash s\} \cap \{(s/(n\backslash s))\backslash s,\ n\backslash s\} = \\ \{(s/(n\backslash s))\backslash s,\ n\backslash s\} \end{array}$$

5.3.2 Remarks on expansions and reductions

[Le Nir, 2004] presents recursive functions for the generation of what we called expansion and reduction sets which may seem to resemble our definitions. Le Nir worked on the product free fragment of *NL*. Let us discuss here his definition to show that, in fact, the differences with our construction are more remarkable than the similarities.

Definition 83. [Le Nir, 2004]: expansion and reduction operations for the product free fragment of *NL* (symmetric cases omitted and some irrelevant notational changes).

$$\begin{array}{lcl} E(a) & = & \{a\}, \text{ if } a \text{ is an atom} \\ R(a) & = & \{a\}, \text{ if } a \text{ is an atom} \\ R(a/b) & = & \{\, a'/b' \mid a' \in R(a) \;\&\; b' \in E(b) \,\} \\ E(a/b) & = & \{\, a'/b' \mid a' \in E(a) \;\&\; b' \in R(b) \,\} \\ & & \cup \\ & & \{\, z \mid b \equiv x\backslash c \;\&\; z \in E(x) \;\&\; a \in R(c) \vee a \in E(c) \,\} \end{array}$$

Observe that the second clause of $E(a/b)$, which is ambiguous, admits a formula x in the expansion set of $z/(x\backslash y)$, if z is in the expansion set of y. Thus, for instance, $x \in E(z/(x\backslash(y/(z\backslash y))))$. On the other hand $x \rightarrow z/(x\backslash(y/(z\backslash y)))$ is not a valid sequent. Indeed, Le Nir's arguments involving expanding patters are rather confusing: in many places he seems to write $x \in E(y)$ meaning $y \in E(x)$. On the other hand, he explicitly states that $b \in E(a/(b\backslash a))$. The same observations hold for [Le Nir, 2003].

A generalization on Kandulski's construction is the extension to the unary operators of [Moortgat, 1997b]. The set AX can be extended to include the axioms and rules for the diamond and box operators.

$$a \longrightarrow \Box\Diamond a \qquad\qquad \Diamond\Box a \longrightarrow a$$

$$\frac{a \longrightarrow c}{\Box a \longrightarrow \Box c}\ \Box M \qquad\qquad \frac{a \longrightarrow c}{\Diamond a \longrightarrow \Diamond c}\ \Diamond M$$

In turn, algorithm 82 can be extended to deal with the unary operators by just adding the following clauses.

Definition 84. Expansion and reduction clauses for unary operators.

$$\begin{array}{rcl}
e(\Diamond a) & = & \{\ \Diamond a' \mid a' \in e(a)\ \} \\
e(\Box a) & = & \text{let } mon \text{ be } \{\ \Box a' \mid a' \in e(a)\ \} \text{ in} \\
 & & \qquad mon \\
 & & \qquad \cup \\
 & & \{\ a' \mid \Box\Diamond a' \in mon\ \} \\
r(\Diamond a) & = & \text{let } mon \text{ be } \{\ \Diamond a' \mid a' \in r(a)\ \} \text{ in} \\
 & & \qquad mon \\
 & & \qquad \cup \\
 & & \{\ a' \mid \Diamond\Box a' \in mon\ \} \\
r(\Box a) & = & \{\ \Box a' \mid a' \in r(a)\ \}
\end{array}$$

Correctness can easily be proved also for this extended system.

The deductive system implicitly used in algorithm 82 and proposition 21 is the following which we call *ER*.

Definition 85. The system *ER*.

- **Identities**:

Axioms	Cut
$a \longrightarrow a$	$\dfrac{a \longrightarrow b \quad b \longrightarrow c}{a \longrightarrow c}$

- **Unary Rules**

Application: $$\frac{a \to b \otimes b\backslash c}{a \to c} \qquad \frac{a \to c/b \otimes b}{a \to c}$$

Lifting: $$\frac{(b/a)\backslash b \to c}{a \to c} \qquad \frac{b/(a\backslash b) \to c}{a \to c}$$

Coapplication: $$\frac{b\backslash(b \otimes a) \to c}{a \to c} \qquad \frac{(a \otimes b)/b \to c}{a \to c}$$

- **Binary Rules**:

Product Rule: $$\frac{a \to a' \quad b \to b'}{a \otimes b \to a' \otimes b'}$$

Monotonicity: $$\frac{a' \to a \quad b \to b'}{b'\backslash a' \to b\backslash a} \qquad \frac{a' \to a \quad b \to b'}{a'/b' \to a/b}$$

The unary extension instead gives rise to the following deduction rules.

Definition 86. *Rules for unary operators*:

Contraction Rules $$\frac{\Box\Diamond a \to c}{a \to c} \qquad \frac{a \to \Diamond\Box c}{a \to c}$$

Monotonicities $$\frac{a \to c}{\Box a \to \Box c} \qquad \frac{a \to c}{\Diamond a \to \Diamond c}$$

5.4 Proof terms

In this section, we present the calculus $\mathcal{G}$, which has the same properties of ER. $\mathcal{G}$ is a labeled deductive system, as defined in chapter 2. We define the term language on which deductions of $\mathcal{G}$ operate.

Definition 87. Proof term language.

$$\begin{array}{lllllll} \mathcal{CB} & := & 1_{\mathcal{F}} & \mid & \mathcal{CB}\cdot\mathcal{CB} & \mid & \\ & & \mathcal{CB}\otimes\mathcal{CB} & \mid & \backslash\!\backslash_{(\mathcal{CB},\mathcal{CB})} & \mid & /\!/_{(\mathcal{CB},\mathcal{CB})} \quad \mid \\ & & \mathcal{CB}^{\mathsf{C}}(\mathcal{CB}) & \mid & \mathscr{L}^{\mathsf{C}}_{(\mathcal{CB},\mathcal{CB})} & \mid & \mathscr{C}^{\mathsf{C}}_{(\mathcal{CB},\mathcal{CB})} \quad \mid \end{array}$$

The rules of $\mathcal{G}$ operate on arrows, $f : a \to c$, where $f \in \mathcal{CB}$ is a syntactic term encoding the proof of the sequent $a \to c$.

Definition 88. The system $\mathcal{G}$.

- **Identities**:

$$\begin{array}{cc} \text{Axioms} & \text{Cut} \\ 1_a \colon a \to a & \dfrac{f \colon a \to b \quad g \colon b \to c}{g \cdot f \colon a \to c} \end{array}$$

- **Binary Rules**:

$$\frac{f \colon a \to a' \quad g \colon b \to b'}{f \otimes g \colon a \otimes b \to a' \otimes b'}$$

$$\frac{g \colon a \to a' \quad f \colon b \to a' \backslash c}{f^{\backslash}(g) \colon a \otimes b \to c} \qquad \frac{f \colon a \to c / b' \quad g \colon b \to b'}{f^{/}(g) \colon a \otimes b \to c}$$

$$\frac{f \colon a' \to a \quad g \colon b \to b'}{\backslash\!\backslash_{(f,g)} \colon b' \backslash a' \to b \backslash a} \qquad \frac{f \colon a' \to a \quad g \colon b \to b'}{/\!/_{(f,g)} \colon a' / b' \to a / b}$$

$$\frac{g \colon b \to a' / c \quad f \colon a' \to a}{\mathscr{L}^{\backslash}_{(f,g)} \colon c \to b \backslash a} \qquad \frac{g \colon b \to c \backslash a' \quad f \colon a' \to a}{\mathscr{L}^{/}_{(f,g)} \colon c \to a / b}$$

$$\frac{g \colon b \to b' \quad f \colon b' \otimes c \to a}{\mathscr{C}^{\backslash}_{(f,g)} \colon c \to b \backslash a} \qquad \frac{g \colon b \to b' \quad f \colon c \otimes b' \to a}{\mathscr{C}^{/}_{(f,g)} \colon c \to a / b}$$

The link between proof terms in $\mathcal{CB}$ and lambda terms is given by the following function.

Definition 89. Mapping from proof term to lambda term.

- $(\dagger) :: \mathcal{CB} \to \mathrm{Lam}$

$$\begin{array}{rcl} (1)^{\dagger} & = & \lambda x.\, x \\ (\mathscr{L}_{(f,g)})^{\dagger} & = & \lambda x\, y.\, (f^{\dagger}\ (g^{\dagger}\ y\ x)) \\ (f^{/}(g))^{\dagger} & = & \lambda x.\, ((f^{\dagger}\ \pi x)\ (g^{\dagger}\ \pi' x)) \\ (f^{\backslash}(g))^{\dagger} & = & \lambda x.\, ((f^{\dagger}\ \pi' x)\ (g^{\dagger}\ \pi x)) \\ (C^{/}_{(f,g)})^{\dagger} & = & \lambda x\, y.\, (f^{\dagger}\ \langle x,\ (g^{\dagger}\ y) \rangle) \end{array}$$

$$\begin{array}{rcl}
(C^{\backslash}_{(f,g)})^{\dagger} & = & \lambda x\, y.\, (f^{\dagger}\, \langle (g^{\dagger}\, y), x\rangle) \\
(f \otimes g)^{\dagger} & = & \lambda x. \langle (f^{\dagger}\, \pi x),\, (g^{\dagger}\, \pi' x)\rangle \\
(\|_{(f,g)})^{\dagger} & = & \lambda x\, y.\, (f^{\dagger}\, (x\, (g^{\dagger}\, y)))
\end{array}$$

As we said, the system $\mathcal{G}$ is closely related to the deductive system $\mathbb{ER}$. In some sense we can see $\mathcal{G}$ as a generalization of $\mathbb{ER}$. The monotonicity and cut rules are common to both systems. Instead, the other rules of $\mathcal{G}$ are all derivable in two steps in $\mathbb{ER}$. For example, we have the following correspondences (omitting the proof term from the $\mathcal{G}$ deductions).

- Application:

$$\frac{a \longrightarrow a' \quad b \longrightarrow a'\backslash c}{a \otimes b \longrightarrow c} \qquad \frac{\dfrac{a \longrightarrow a' \quad b \longrightarrow a'\backslash c}{a \otimes b \longrightarrow a' \otimes a'\backslash c}}{a \otimes b \longrightarrow c}$$

- Lifting:

$$\frac{b \longrightarrow a'/c \quad a' \longrightarrow a}{c \longrightarrow b\backslash a} \qquad \frac{\dfrac{a' \longrightarrow a \quad b \longrightarrow a'/c}{(a'/c)\backslash a' \longrightarrow b\backslash a}}{c \longrightarrow b\backslash a}$$

- Coapplication:

$$\frac{b \longrightarrow b' \quad b' \otimes c \longrightarrow a}{c \longrightarrow b\backslash a} \qquad \frac{\dfrac{b' \otimes c \longrightarrow a \quad b \longrightarrow b'}{b'\backslash(b' \otimes c) \longrightarrow b\backslash a}}{c \longrightarrow b\backslash a}$$

We are going to see that the expansion-reduction procedure given in definition 82 can be restated to work on $\mathcal{G}$. The only difference with respect to definition 82 is that that the functions are now of type $e^{\star}, r^{\star} :: \mathcal{F} \longrightarrow \{(\mathcal{CB}, \mathcal{F})\}$, where $\mathcal{CB}$ encodes the proof. The functions are defined below. Observe that we added also the clauses for the unary operators, according to the term labeling of the system $\mathcal{G}$.

Definition 90. Labeled expansion-reduction procedure.

- $e^{\star}(a) = \{\, 1_a : a \,\}$, if a is an atom.

- $e^{\star}(a \otimes b) = \{\, f \otimes g : a' \otimes b' \mid f : a' \in e^{\star}(a) \;\&\; g : b' \in e^{\star}(b) \,\}$

- $e^\star(a/b)$ = let as be $e^\star(a)$ and $bs = r^\star(b)$ in

$$\begin{gathered}\{\, /\!\!/_{(f,g)} : a'/b' \mid f : a' \in as \;\&\; g : b' \in bs \,\} \\ \cup \\ \{\, \mathscr{C}^{/}_{(f,g)} : c \mid f : c \otimes b' \in as \;\&\; g : b' \in bs \,\} \\ \cup \\ \{\, \mathscr{L}^{/}_{(f,g)} : c \mid f : a' \in as \;\&\; g : c\backslash a' \in bs \,\}\end{gathered}$$

- $r^\star(a) = \{\, 1_a : a \,\}$, if a is an atom.
- $r^\star(a/b) = \{\, /\!\!/_{(f,g)} : a'/b' \mid f : a' \in r^\star(a) \;\&\; g : b' \in e^\star(b) \,\}$
- $r^\star(a \otimes b)$ = let as be $r^\star(a)$ and bs be $r^\star(b)$ in

$$\begin{gathered}\{\, f \otimes g : a' \otimes b' \mid f : a' \in as \;\&\; g : b' \in bs \,\} \\ \cup \\ \{\, f^{/}(g) : c \mid f : c/b' \in as \;\&\; g : b' \in bs \,\} \\ \cup \\ \{\, f^{\backslash}(g) : c \mid g : a' \in as \;\&\; f : a'\backslash c \in bs \,\}\end{gathered}$$

One can easily prove the following statements.

Proposition 22.

If $f : a \in e^\star(c)$, then $f : a \xrightarrow{e} c$.

If $f : c \in r^\star(a)$, then $f : a \xrightarrow{r} c$.

$\vdash_{NL} f \cdot g : a \to c$ iff $g : b \in r^\star(a)$ and $f : b \in e^\star(c)$.

In chapter 7, we will apply the cut elimination algorithm to $\mathcal{G}$ and see how proof terms can be used for proof normalization.

5.5 Connection to parsing

Although definition 82 provides a recognition procedure for two-formula sequents, the method can be easily generalized for addressing the more general problem of *parsing*. The result of [Kandulski, 1988] of equivalence of *NL* grammars and *CF* grammars relies on the reducibility of an *NL* grammar into an *AB* grammar with product. In the present setting, we can state the reducibility of *NL* computations to *AB* computations as follows (see also [Buszkowski, 1997]).

Proposition 23. If $a_1, \ldots, a_n \rightarrow c$ is provable in NL, then there are formulas b_i, $1 \leqslant i \leqslant n$ such that $b_i \in r(a_i)$, and a formula b', such that $b' \in e(c)$, and $b_1, \ldots, b_n \rightarrow b'$ is derivable only by means of the application and product rule of $AB^{\otimes}$.

More in general, the expansion and reduction operations can be used to transform a NL grammar into an $AB^{\otimes}$ grammar as follows.

Proposition 24. From a NL categorial grammar $G = \langle V_t, s, Lex, NL\rangle$, we compute an AB grammar $G' = \langle V_t, s, Lex', AB^{\otimes}\rangle$ such that $L(G) = L(G')$, where

$Lex' = \{\, w \rightarrow x' \mid w \rightarrow x \in Lex,\ x' \in r(x)\,\}$

Example 53.

Let us use starred variables for lexical items and write $x \rightarrow y_1 \mid \ldots \mid y_n$ for $x \rightarrow y_1, \ldots, x \rightarrow y_n$. A_6 is the NL grammar with the following lexicon:

$$\begin{array}{lcl} n^* & \rightarrow & n \mid s/(n\backslash s) \mid tv\backslash(s/(n\backslash s))\backslash s \mid (s/(n\backslash s) \otimes tv)\backslash s \\ tv^* & \rightarrow & tv \\ hv^* & \rightarrow & (s/(n\backslash s))\backslash s \end{array}$$

Lexical expansion, according to proposition 24, of grammar A_6 gives the AB grammar with product whose lexicon is the following.

$$\begin{array}{ll} n^* & \rightarrow n \mid s/(n\backslash s) \mid tv\backslash(s/(n\backslash s))\backslash s \mid tv\backslash n\backslash s \mid (s/(n\backslash s) \otimes tv)\backslash s \mid (n \otimes tv)\backslash s \\ tv^* & \rightarrow tv \\ hv^* & \rightarrow (s/(n\backslash s))\backslash s \mid n\backslash s \end{array}$$

Thus the parsing methods developed in the chapter 3 can be immediately applied to the resulting grammar.

An important issue should be investigated. The efficiency of all parsing algorithms of the chapter 3 is influenced by the size of the input AB grammar. The size of the input AB grammar in turn depends on the size of the original NL. Indeed, if G is the original NL grammar and G' the resulting AB grammar with product, then $|G| \leqslant |G'|$. How bigger G' is, depends on the length and on the order of the formulas of G. In many practical applications, G' may retain a reasonable size, as higher order types rarely exceed the third order, and formulas are usually relatively short. However in general, lexical expansion may give rise to an exponential growth of the resulting grammar.

5.6 Conclusion

We started this chapter by presenting Buszkowski's and Kandulski's method for the construction of normal derivations. We defined two recursive functions, called e and r, generating respectively the set of expanding and the set of reducing sequents of Kandulski's construction with finite means. We used these two functions to define a recognition method for two formula sequents and a lexical compilation transforming an NL grammar into an $AB^{\otimes}$ grammar.

In chapter 6, we will see that the expansion/reduction approach improves on Kandulski's construction under a second significant respect. Normal derivations, as constructed in proposition 21 are a subset of Kandulski's normal derivations. However, we will prove that all derivations which are excluded by our method are in fact *redundant*.

Chapter 6

Normal Derivations and Ambiguity

In the previous chapter, we have defined a simple and elegant method for proving two-formula sequents. This method has been grounded on the result of [Kandulski, 1988] about normal derivations in *NL*. We proved the equivalence of our recursive functions e and r in definition 82 with Kandulski's characterization of the sets of expanding and reducing sequents of *NL*.

In this chapter, we show that while the functions in definition 82 do the same job of Kandulski's construction, in fact they do it better. We address the problem of *spurious ambiguity* in *NL*. We observe that *normal* in Kandulski's sense, does not imply *uniqueness* as we may find several equivalent derivations of the same two formula sequent according to Kandulski's method. Instead, the recursive functions e and r in definition 82 return exactly one deduction for every semantic reading that a sequent may have. Thus, the method we designed for proving two formula sequents is a *redundancy-free* theorem prover.

We will address also a second problem, which can be stated as follows.

- Given a provable sequent $a \rightarrow c$, how many *proofs* may this sequent have?

This question has been previously addressed in [van Benthem, 1991] and [Tiede, 1999] and represents an important issue for proof theoretic grammar. In [van Benthem, 1991], one finds the discussion of the problem of providing "an *explicit function* computing numbers of non-equivalent readings for sequents in the Lambek calculus"[1]. We present such an "explicit function", whose only parameter is the length of the input sequent. While van Benthem and later [Tiede, 1999] prove the so called *finite reading property* for sequents of the Lambek calculus (with permutation), we establish a direct link between the length of an *NL* sequent and the *binomial coefficient*.

6.1 Eliminating redundancies

The reader may have observed that Kandulski's notion of normal derivation does not imply uniqueness. Consider the following examples.

(6.1) $(a \otimes a\backslash c)/b \otimes b,\ c/b \otimes b,\ c$

(6.2) $(a \otimes a\backslash c)/b \otimes b,\ a \otimes a\backslash c,\ c$

These two derivations have the same length. Moreover, no shorter derivation is available for the sequent $(a \otimes a\backslash c)/b \otimes b \to c$. Thus, they are both minimal, hence *normal* by proposition 16. However, while distinct, these derivations are also in some sense *equivalent*. We have seen that in the case of *CF* grammars (as well as for basic categorial grammars), the structural description represents a *criterion of equivalence* among different derivations. In the case of *NL* and of logical systems in general, the issue of derivation equivalence is much more subtle and deep. The theory of *proof nets*, for instance, has been developed primarily as redundancy-free representation of proofs. We will appeal to lambda term semantics to define equivalence among different derivations of the same sequent.

Let us assign to each axiom and rule that we used in the construction in definition 77 in the previous chapter a lambda term.

Definition 91. Labeled axioms and rules of *NL*.

- *Identities*:

Axioms

$$\lambda x.x \colon a \to a$$

Cut

$$\frac{u \colon a \to b \quad v \colon b \to c}{\lambda x.(v\ (u\ x)) \colon a \to c}$$

[1] Cursive mine. The quotation of [van Benthem, 1991] is taken from [Tiede, 1999] who quotes the American edition of the book.

- AX^0:

$$\lambda x.(\pi' x\ \pi x) \colon b \otimes b\backslash c \to c \qquad \lambda x.(\pi x\ \pi' x) \colon c/b \otimes b \to c$$

$$\lambda x \lambda y.(y\ x) \colon a \to (b/a)\backslash b \qquad \lambda x \lambda y.(y\ x) \colon a \to b/(a\backslash b)$$

$$\lambda\, x \lambda y.\langle y,\ x\rangle \colon a \to b\backslash(b \otimes a) \qquad \lambda\, x \lambda y.\langle x,\ y\rangle \colon a \to (a \otimes b)/b$$

- M^0:

$$\frac{u \colon a \to a'}{\lambda\, x.\langle (u\ \pi x), \pi' x\rangle \colon a \otimes b \to a' \otimes b} \qquad \frac{v \colon b \to b'}{\lambda\, x.\langle \pi x, (v\ \pi' x)\rangle \colon a \otimes b \to a \otimes b'}$$

$$\frac{u \colon a' \to a}{\lambda\, x\, y.\, (u\ (x\ y)) \colon b\backslash a' \to b\backslash a} \qquad \frac{u \colon a' \to a}{\lambda\, x\, y.\, (u\ (x\ y)) \colon a'/b \to a/b}$$

$$\frac{v \colon b \to b'}{\lambda\, x\, y.\, (x\ (v\ y)) \colon b'\backslash a \to b\backslash a} \qquad \frac{v \colon b \to b'}{\lambda\, x\, y.\, (x\ (v\ y)) \colon a/b' \to a/b}$$

The construction of normal derivations in the previous chapter can be rephrased to work with the arrows given above in a straightforward way. The lambda terms will play a role only in telling us which derivations are equivalent. Let us define equivalence among normal derivations.

Definition 92. Let D_1 and D_2 be two labeled normal derivations of a sequent $a \to c$ in AX. Let t_1 and t_2 be the normal form lambda terms in the conclusion of D_1 and D_2, respectively. Then D_1 is equivalent to D_2 if and only if $t_1 \equiv t_2$.

One may observe that the symmetric variants of all axioms and rules in which only slashes appear receive the same semantic labelings. On the other hand, for instance, the trivial derivations $\lambda x \lambda y.(y\ x) \colon a \to (b/a)\backslash b$ and $\lambda x \lambda y.(y\ x) \colon a \to b/(a\backslash b)$ are not equivalent, as $a \to (b/a)\backslash b \neq a \to b/(a\backslash b)$. A similar argument applies to what is deduced from the rules of inferece.

Let us consider the previous example of spurious ambiguity, this time with term labeling and in tree format. For reasons of space, we represent cuts as unary rules, in a way similar to the system *ER*.

(6.3) Labeled derivation 6.1:

$$\dfrac{\dfrac{\dfrac{\lambda x.(\pi x\ \pi' x) \colon a \otimes a\backslash c \to c}{\lambda\, x.\lambda\, y.(\pi'(x\ y)\ \pi(x\ y)) \colon (a \otimes a\backslash c)/b \to c/b}}{\lambda\, x.\langle \lambda\, y.(\pi'(\pi x\ y)\ \pi(\pi x\ y)), \pi' x\rangle \colon (a \otimes a\backslash c)/b \otimes b \to c/b \otimes b}}{\lambda\, x.(\pi'(\pi x\ \pi' x)\ \pi(\pi x\ \pi' x)) \colon (a \otimes a\backslash c)/b \otimes b \to c}$$

(6.4) Labeled derivation 6.2:

$$\frac{\lambda x.(\pi x\ \pi' x)\colon (a \otimes a\backslash c)/b \otimes b \to a \otimes a\backslash c}{\lambda\, x.(\pi'(\pi x\ \pi' x)\ \pi(\pi x\ \pi' x))\colon (a \otimes a\backslash c)/b \otimes b \to c}$$

The term in the conclusion of the two deductions is indeed the same, hence the two deductions are equivalent.

We are going to address the problem of *spurious ambiguity* in the context of normal derivations. The previous example shows that Kandulski's normal derivations are not exempt from redundancies. However, we will see that the construction method that we provided in definition 82 is free from spurious ambiguity.

We will single out, among Kandulski's normal derivations, those derivations that are *redundant*. The choice of which derivations are redundant is, to some extent, *arbitrary*. We can observe, however, that the deductions in 6.3 and in 6.4 exhibit two different structures. Deduction 6.1 follows an *innermost* reduction strategy: the most embedded formulas are contracted before the most external ones. Instead, deduction 6.1 follows an *outermost* reduction strategy: the most external formulas are contracted before the most embedded ones. In this chapter, our choice will fall on the outermost reduction. Whenever two derivations D_1 and D_2 are available that differ only in that D_1 follows an innermost reduction, where D_2 follows an outermost reduction, we will say that D_2 is redundant. Our choice is primarily dictated by the fact that in the present context the innermost reduction results very easy to control and implement. We will see in the next section that definition 82 implements an innermost reduction strategy. In the next chapter, we will explore also the benefits of an outermost reduction strategy.

Let us formally define what we mean by redundant derivation in the present context.

Definition 93. A normal derivation $D = x_1 \dots x_n$ is *redundant*, if D contains the sublist $x_{i-1}x_ix_{i+1}$ such that $x_{i-1} \to x_i$ and $x_i \to x_{i+1}$ are obtained through n, $0 \leqslant n$, applications of M^0 (clause 2) of definition 77) from two premises $x \to y$ and $y \to z$, respectively, which are both expanding or reducing sequents in AX^0.

We call the sequence $x_{i-1}x_ix_{i+1}$, generating the redundancy in the derivation, *redundant sequence*.

According to this definition, derivation 6.2 is redundant, while derivation 6.1 is not. Other examples of redundant derivations are the followings.

$$a,\ (a \otimes b)/b,\ c/(((a \otimes b)/b)\backslash c)$$

$$(c/(((a \otimes b)/b)\backslash c))\backslash x,\ ((a \otimes b)/b)\backslash x,\ a\backslash x$$

Indeed, we are assuming that the 'genuine' derivations are those following the innermost reduction strategy:

$$\begin{gathered} a,\ c/(a\backslash c),\ c/(((a \otimes b)/b)\backslash c) \\ (c/(((a \otimes b)/b)\backslash c))\backslash x,\ (c/(a\backslash c))\backslash x,\ a\backslash x \end{gathered}$$

We can easily prove that redundant derivations are indeed superfluous, in the sense that we are free to discard them without loss of information.

Proposition 25. Let D be a redundant derivation of $a \to c$. Let $x_{i-1}x_ix_{i+1}$ be a redundant sequence in D. Then there is another normal derivation D' of $a \to c$ in which the redundant sequence $x_{i-1}x_ix_{i+1}$ does not appear.

Proof.

Assume $x_{i-1} \to x_i$ and $x_i \to x_{i+1}$ in D are both instances of application, as in example 6.2. Then x_i occurs as a subformula of x_{i-1}. Thus, by replacing x_{i+1} for x_i in x_{i-1}, we obtain the derivation D', as in example 6.1.

If $x_{i-1} \to x_i$ and $x_i \to x_{i+1}$ are both instances of coapplication or lifting, or one of coapplication and the other of lifting, we proceed in the same way as before.

Suppose that $x_{i-1} \to x_i$ and $x_i \to x_{i+1}$ in D are obtained by n applications of M^0, $n > 0$. Then there are premises $x'_{i-1} \to x'_i$ and $x'_i \to x'_{i+1}$ (or $x'_{i+1} \to x'_i$ and $x'_i \to x'_{i-1}$) obtained by $n - 1$ applications of M^0. By IH, there is a non-redundant sequence $x'_{i-1}x^*_ix'_{i+1}$ (or $x'_{i+1}x^*_ix'_{i-1}$) in AX, and by the closure of AX under M^0, we obtain D' containing the non-redundant sequence $x_{i-1}x'^*_ix_{i+1}$. □

6.2 Unique normal derivations

In the previous section, we showed how to filter out redundant derivations from Kandulski's normal derivations. We have selected as non-redundant the derivations following the innermost contraction strategy. Let us consider once more examples 6.1 and 6.2, proposed here as 6.5 and 6.6, respectively.

(6.5) $(a \otimes a\backslash c)/b \otimes b,\ c/b \otimes b,\ c$

(6.6) $(a \otimes a\backslash c)/b \otimes b,\ a \otimes a\backslash c,\ c$

One may observe that definition 82 would generate $(a \otimes a\backslash c)/b \otimes b \rightarrow c$ according to (the deduction corresponding to) derivation 6.5, rather than to derivation 6.6. In fact, algorithm 82 follows the innermost contraction strategy.

In order to show that no redundancy affects our algorithm, let us restate definition 82 to be sensitive to multiplicity.

Algorithm 7. Let us define functions e' and r', which are like e and r in definition 82, respectively, except for the fact that they use [ ] instead of { }, that is, list comprehension instead of set comprehension, and ++ instead of ∪, that is, list concatenation instead of set union.

Then we can see the following trace for $r'((a \otimes a\backslash c)/b \otimes b)$.

$$\dfrac{\dfrac{\dfrac{r'(a) = [\ a\ ] \qquad \dfrac{r'(c) = [\ c\ ] \quad e'(a) = [\ a\ ]}{r'(a\backslash c) = [\ a\backslash c\ ]}}{r'(a \otimes a\backslash c) = [\ a \otimes a\backslash c,\ c\ ]} \qquad e'(b) = [\ b\ ]}{r'((a \otimes a\backslash c)/b) = [\ (a \otimes a\backslash c)/b,\ c/b\ ]} \qquad r'(b) = [\ b\ ]}{r'((a \otimes a\backslash c)/b \otimes b) = [\ (a \otimes a\backslash c)/b \otimes b,\ c/b \otimes b,\ a \otimes a\backslash c,\ c\ ]}$$

Although both $c/b \otimes b$ and $a \otimes a\backslash c$ are in the list $r'((a \otimes a\backslash c)/b \otimes b)$, *only one occurrence* of the formula c is in this list. In fact, c is obtained at the root by contracting $c/b \otimes b$ (which is also obtained at the root). Instead, $a \otimes a\backslash c$ is obtained, again at the root, by contracting $(a \otimes a\backslash c)/b \otimes b$. Thus, the deduction we obtain for $(a \otimes a\backslash c)/b \otimes b \rightarrow c$ is the following.

(6.7)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{a \otimes a\backslash c \rightarrow a \otimes a\backslash c}}{a \otimes a\backslash c \rightarrow c} \qquad b \rightarrow b}{(a \otimes a\backslash c)/b \rightarrow c/b} \qquad b \rightarrow b}{(a \otimes a\backslash c)/b \otimes b \rightarrow c/b \otimes b}}{(a \otimes a\backslash c)/b \otimes b \rightarrow c}$$

As we saw, the derivation list in example 6.6 corresponds to the following deduction.

(6.8)

$$\dfrac{\dfrac{\dfrac{\vdots}{(a \otimes a\backslash c)/b \otimes b \rightarrow (a \otimes a\backslash c)/b \otimes b}}{(a \otimes a\backslash c)/b \otimes b \rightarrow a \otimes a\backslash c}}{(a \otimes a\backslash c)/b \otimes b \rightarrow c}$$

Observe that this second derivation could be obtained by definition 82 if, instead of *one cycle* of pattern contraction for each formula in *mon* (see clauses b) and c) of definition 82), we computed the *closure* of the set *mon* under contraction operations. However, this is not necessary required for NL[2].

Proposition 26. The recognition procedure resulting from proposition 21 generates only non-redundant derivations.

Proof. As definition 82 follows the innermost reduction strategy and only one cycle of pattern contraction is applied, no redundant derivation may arise. □

6.3 Enumerating readings

In this section, we examine the problem of calculating how many readings, that is how many non-equivalent derivations, a sequent may have. [van Benthem, 1991] proves that the number of non-equivalent readings for sequents of the commutative Lambek calculus is *finite*. Obviously, this finiteness result applies NL as well (every sequent has only a finite number of normal derivations). Nonetheless, it appears that this number may soon become very big in relation, for instance, to the length of the input sequent. Consider the example of $s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s \to s$, discussed by [Hendriks, 1993] and [de Groote, 1999], among others. This sequent has the following two readings, presented as deductions in ER.

(6.9) Subject wide scope:

$$\dfrac{\dfrac{s/(n\backslash s) \to s/(n\backslash s) \qquad \dfrac{s \to s \qquad \dfrac{s/(n\backslash s) \to s/(n\backslash s)}{n \to s/(n\backslash s)}}{(s/(n\backslash s))\backslash s \to n\backslash s}}{s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s \to s/(n\backslash s) \otimes n\backslash s}}{s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s \to s}$$

[2] Compare this situation to the *associative Lambek calculus*. If we wish to build normal derivations for this system, we should, for example, capture among the pattern contractions, the expanding pattern $a/b \to (a/c)/(b/c)$. Thus, if a formula of the form $(a/c)/(b/c)$ were in the set *mon*, we would generate a/b, in the same way as we generate c, if $a/(c\backslash a)$ is in *mon* in definition 82. Observe, anyway, that in this case a/b is not a subformula of $(a/c)/(b/c)$, thus it may give raise to a further contraction, if, for eample, $a/b \equiv (a'/x)/(b'/x)$, and so on. Therefore, in this case, one cycle of pattern simplification would not be enough and we should compute the *closure* of the set *mon* under the appropriate contraction operations.

(6.10) Verb wide scope:

$$\frac{s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s \to s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s}{s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s \to s}$$

De Groote concludes what follows.

> Now it is easy to construct, from the above example, sequents with an exponential number of possible proofs.

Let us examine this problem in more detail and try to find an upper bound to such number. Consider once more the set AX^0 of the basic characteristic laws of *NL*.

$$\begin{array}{ll} c/a \otimes a \to c & a \otimes a\backslash c \to c \\ a \to (a \otimes c)/c & a \to c\backslash(c \otimes a) \\ a \to c/(a\backslash c) & a \to (c/a)\backslash c \end{array}$$

Let us define the following operations, together with their symmetric forms, eventually.

Definition 94. For a and b ranging over *atoms*,

- $\phi^x(a)(b)(0) = a$, for $x \in \{e, c, l\}$
- $\phi^e(a)(b)(i+1) = (\phi^e(a)(b)(i))/b \otimes b$
- $\phi^c(a)(b)(i+1) = ((\phi^c(a)(b)(i)) \otimes b)/b$
- $\phi^l(a)(b)(i+1) = b/((\phi^l(a)(b)(i))\backslash b)$

Each iteration of these operations introduces a pair of connectives. As a and b are arbitrary atoms, and constant in the iteration, we shall write $\phi^x(n)$, $x \in \{e, c, l\}$, for $\phi^x(a)(b)(n)$. We may observe the following.

Remark 4. $\phi^x(n)$, $x \in \{c, l\}$ is a *closure operation*, for $n \geqslant 0$ as each of the following sequents is provable.

$$\begin{array}{lll} \phi^x(n) & \to & \phi^x(n+1) \\ \phi^x(n+2) & \to & \phi^x(n+1) \\ \phi^x(a)(b)(n) & \to & \phi^x(a')(b)(n), \text{ if } a \to a' \end{array}$$

In order to make this construction completely uniform, we may introduce the following abstraction.

Definition 95. Let $n \geqslant 0$ and $m > 0$. We write $n \xrightarrow{\phi} m$ for

$\phi(a)(b)(n) \rightarrow \phi(a)(b)(m)$, if $\phi \in \{\phi^c, \phi^l\}$

$\phi(a)(b)(m) \rightarrow \phi(a)(b)(n)$, if $\phi \in \{\phi^e\}$.

We call ϕ-sequents such kinds of sequents. For example,

(6.11) $2 \xrightarrow{\phi^c} 3 = ((a \otimes b)/b \otimes b)/b \rightarrow (((a \otimes b)/b \otimes b)/b \otimes b)/b$

(6.12) $2 \xrightarrow{\phi^e} 3 = ((a/b \otimes b)/b \otimes b)/b \otimes b \rightarrow (a/b \otimes b)/b \otimes b$

As we said before, the number of non-equivalent proofs for a sequent of the *commutative* Lambek is *finite* as proved in [van Benthem, 1991] and [Tiede, 1999]. [de Groote, 1999] claims that such number may be exponential on the length of the sequent even in the case of *NL*. In general, it seems to us that the only way to find out the actual number of readings of a sequent is to count the number of proofs that can be constructed for it. Consequently, we can know the degree of ambiguity of a sequent only *a posteriori*. In certain special cases, however, it is possible to know the number of readings of a sequent by just looking at its shape. This is indeed the case for ϕ-sequents, as we will see. Furthermore, we are going to show that ϕ-sequents of length n provide an upper bound for the number of readings of any sequent of length n. This gives us the possibility to provide an *a priori* upper bound for the degree of ambiguity of any sequent.

Let us define the following count.

Definition 96. Let $n \geqslant 0$ and $m \geqslant 1$.

$$\begin{array}{lcl} \rho(n, m) & = & 1, \text{ if } n = 0 \text{ or } m = 1 \\ \rho(n+1, m+1) & = & \rho(n, m+1) + \rho(n+1, m) \end{array}$$

We now prove that the function ρ counts exactly the number of readings of ϕ-sequents. We write $|a \rightarrow c|^\rho$ for the number of different proofs of $a \rightarrow c$.

Proposition 27. Let $n \geqslant 0$ and $m \geqslant 1$.

$$|n \xrightarrow{\phi} m|^\rho = \rho(n, m)$$

Proof. In the proof, we examine the case of ϕ^c, the other cases being similar.

Case $n = 0$ and $m > 1$. By induction hypothesis, $|0 \xrightarrow{\phi} m - 1|^{\rho} = 1$. There is only the following way to obtain $a \xrightarrow{\phi} m$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{0 \xrightarrow{\phi^c} m-1}{a \to \phi^c(m-1) \quad b \to b}}{a \otimes b \to \phi^c(m-1) \otimes b \quad b \to b}}{(a \otimes b)/b \to (\phi^c(m-1) \otimes b)/b}}{a \to (\phi^c(m-1) \otimes b)/b}}{0 \xrightarrow{\phi^c} m}$$

Case $m = 1$ and $n > 0$. Then, by induction hypothesis, $|n - 1 \xrightarrow{\phi} 1|^{\rho} = 1$. The only way of proving $n \xrightarrow{\phi} 1$ is the following.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{n-1 \xrightarrow{\phi^c} 1}{\phi^c(n-1) \to (a \otimes b)/b \quad b \to b}}{\phi^c(n-1) \otimes b \to (a \otimes b)/b \otimes b}}{\phi^c(n-1) \otimes b \to a \otimes b \qquad b \to b}}{(\phi^c(n-1) \otimes b)/b \to (a \otimes b)/b}}{n \xrightarrow{\phi^c} 1}$$

Case $n > 0$ and $m > 1$. There are two possibilities of obtaining $n \xrightarrow{\phi^c} m$.

1.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{n \xrightarrow{\phi^c} m-1}{\phi^c(n) \to \phi^c(m-1) \quad b \to b}}{\phi^c(n) \otimes b \to \phi^c(m-1) \otimes b \quad b \to b}}{(\phi^c(n) \otimes b)/b \to (\phi^c(m-1) \otimes b)/b}}{\phi^c(n) \to (\phi^c(m-1) \otimes b)/b}}{n \xrightarrow{\phi^c} m}$$

2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{n-1 \xrightarrow{\phi^c} m}{\phi^c(n-1) \to \phi^c(m)}}{\phi^c(n-1) \to (\phi^c(m-1) \otimes b)/b \quad b \to b}}{\phi^c(n-1) \otimes b \to (\phi^c(m-1) \otimes b)/b \otimes b}}{\phi^c(n-1) \otimes b \to \phi^c(m-1) \otimes b \qquad b \to b}}{(\phi^c(n-1) \otimes b)/b \to (\phi^c(m-1) \otimes b)/b}}{n \xrightarrow{\phi^c} m}$$

Therefore, $|n \xrightarrow{\phi} m|^{\rho} = |n \xrightarrow{\phi} m-1|^{\rho} + |n-1 \xrightarrow{\phi} m|^{\rho}$. By induction hypothesis, $|n \xrightarrow{\phi} m-1|^{\rho} = \rho(n, m-1)$ and $|n-1 \xrightarrow{\phi} m|^{\rho} = \rho(n-1, m)$. Hence, $|n \xrightarrow{\phi} m|^{\rho} = \rho(n-1, m) + \rho(n, m-1)$. □

Theorem 27 gives rise to the table in figure 6.1, where we enumerate readings for ϕ-sequents. For $n \xrightarrow{\phi} m$, with $0 \leqslant n \leqslant 7$ and $0 < m \leqslant 7$, we write n in the leftmost vertical column and m in the topmost horizontal row. Notice that this is the well known Pascal's triangle.

$\xrightarrow{\phi}$	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1
1	1	2	3	4	5	6	7
2	1	3	6	10	15	21	28
3	1	4	10	20	35	56	84
4	1	5	15	35	70	126	210
5	1	6	21	56	126	252	462
6	1	7	28	84	210	462	924
7	1	8	36	120	330	792	1716

Figure 6.1: Readings of ϕ-sequents.

Remark 5. The count $\rho(n, m)$ can be formulated in terms of the *binomial coefficient*.

$$\rho(n, m) = \binom{n+m-1}{m-1} = \frac{(n+m-1)!}{n!(m-1)!}$$

Going back to the issue of finding an upper bound for the number of reading of a *NL* sequent, let us introduce the following notation. We denote Φ_n the set of ϕ-sequents of length n. Moreover, let $\lceil \Phi_n \rceil$ be the integer m such that $s \in \Phi_n$, $|s|^{\rho} = m$ and for all $s' \in \Phi_n$, $m \geqslant |s'|^{\rho}$. We can state the following.

Proposition 28. Let $a \rightarrow c$ be given. If $|a \rightarrow c| = n$, then $|a \rightarrow c|^{\rho} \leqslant \lceil \Phi_n \rceil$.

Proof. The shape of ϕ-sequents allows to match literals and connectives inside them, introducing an high degree of ambiguity. Consider, as an example, the ϕ-sequent $1 \xrightarrow{\phi^c} 2$. This has two readings, namely

$$\frac{((a \otimes b)/b \otimes b)/b \rightarrow ((a \otimes b)/b \otimes b)/b}{(a \otimes b)/b \rightarrow ((a \otimes b)/b \otimes b)/b}$$

and

$$\cfrac{\cfrac{\cfrac{(a\otimes b)/b \to (a\otimes b)/b}{a \to (a\otimes b)/b} \qquad b \to b}{a\otimes b \to (a\otimes b)/b\otimes b} \qquad b \to b}{(a\otimes b)/b \to ((a\otimes b)/b\otimes b)/b}$$

Any sequent resulting from a ϕ-sequent by changing atoms, for example $(a\otimes b)/b \to ((a\otimes b)/b\otimes c)/c$, has a smaller number of readings. If we replace a formula for an atom, the length of the sequent increases. Changing connectives destroys the shape of ϕ-sequents. □

In conclusion, we have the following result for the number of readings of arbitrary sequents.

Proposition 29. Let $a \to c$ be given. Let $|a \to c| = n$ and $m = \frac{n}{2} - 1$. Then,

- if $m = 0$, then
$$|a \to c|^{\rho} \leqslant 1$$

- otherwise,
$$|a \to c|^{\rho} \leqslant max\ \{\ \rho(i,j) \mid 0 \leqslant i\ \&\ 0 < j\ \&\ i+j = m\ \}$$

Proof. The case $m = 0$ accounts for the case in which a and c are atoms. The other case follows immediately from propositions 27 and28. □

6.4 Conclusion

In this chapter, we have refined Kandulski's notion of normal derivation by eliminating what we called redundant derivations. We proved that the procedure in definition 82 does indeed generate only non-redundant derivations. Thus, definition 82 provides a stronger notion of normal derivation: one which is not affected by spurious ambiguity.

In exploring the properties of the theorem prover arising from proposition 21, we observed the elegant progression of figure 6.1. I wish to thank Michael Moortgat for telling me that that was the famous Pascal triangle. The link between number of readings of *NL* sequents and binomial coefficient is an important result, that strengthen the previous results of [van Benthem, 1991] and [Tiede, 1999]. This result clarifies also [de Groote, 1999] claims about the exponential number of readings of *NL* sequents.

Chapter 7

A look at cut elimination

In chapter 2, we have presented labeled deductive systems and shown how to associate lambda terms to derivations. Although it is possible to assign lambda terms to categorial derivations, it is not possible to associate categorial derivations to lambda terms: due to the lack of directionality of the lambda notation several derivations may correspond to the same lambda term. This means that the relation between Lambek derivations and lambda terms is not an *isomorphism*, as it is the relation between intuitionistic derivations and lambda terms, but only an *homomorphism*: several derivations may correspond to a single term.

In a series of papers, Lambek presented his calculi with the term annotation proper to *category theory* and formulated rules of conversion for terms encoding proofs. [Lambek and Scott, 1987] is probably the most thorough work on this approach, although it is dedicated to intuitionistic logic. Instead, *NL* appears as a fragment of the logic in [Lambek, 1993], where a term language isomorphic to sequent calculus deductions is formulated. In this paper, Lambek defines conversion rules for proof terms on the basis of the equations arising from application of the cut-elimination algorithm. From this perspective, what we called before a sequent becomes the *type* of the term. The advantage of such a formalism for proofs is that equality of proofs become syntactic congruence of terms in normal form.

On the other hand, the sequent formulation adopted in [Lambek, 1993] renders sometimes ambiguous the formulation of terms, as Lambek himself admits.

> The notation $f\langle g\rangle$ for the resulting sequent of a cut is of course ambiguous, as it does not show where f has been substituted into g, but we shall use it nonetheless in order to avoid making

the notation too heavy.

In this chapter, we look at the cut-elimination algorithm within a combinatorial formulation of *NL* which will allow us to define explicitly the equations arising from the cut elimination algorithm.

7.1 Admissibility of Cut

Let us formulate the system $\mathcal{G}^\star$, which is like $\mathcal{G}$ except in that it lacks the lifting rule,

$$\frac{g\colon b \to a'/c \quad f\colon a' \to a}{\mathscr{L}^{\backslash}_{(f,g)}\colon c \to b\backslash a}$$

and its symmetric. These rules are in fact derivable in $\mathcal{G}^\star$, thus we omit them from the analysis.

$$\dfrac{1_b\colon b \to b \qquad \dfrac{\dfrac{g\colon b \to a'/c \quad 1_c\colon c \to c}{g^{/}(1_c)\colon b \otimes c \to a'} \qquad f\colon a' \to a}{f \cdot g^{/}(1_c)\colon b \otimes c \to a}}{C^{\backslash}_{(f \cdot g^{/}(1_c),1_b)}\colon c \to b\backslash a}$$

Hence, we have

$$\begin{array}{rcl} \mathscr{L}^{\backslash}_{(f,g)} & = & C^{\backslash}_{(f \cdot g^{/}(1_c)1_b)} \\ \mathscr{L}^{/}_{(f,g)} & = & C^{/}_{(f \cdot g^{\backslash}(1_c)1_b)} \end{array}$$

For ease of exposition, we give here the system $\mathcal{G}^\star$.

- **Identities**:

Axioms

$$1_a\colon a \to a$$

Cut

$$\frac{f\colon a \to b \quad g\colon b \to c}{g \cdot f\colon a \to c}$$

- **Binary Rules**:

$$\frac{f\colon a \to a' \quad g\colon b \to b'}{f \otimes g\colon a \otimes b \to a' \otimes b'}$$

$$\frac{g\colon a \to a' \quad f\colon b \to a'\backslash c}{f^{\backslash}(g)\colon a \otimes b \to c} \qquad\qquad \frac{f\colon a \to c/b' \quad g\colon b \to b'}{f^{/}(g)\colon a \otimes b \to c}$$

$$\frac{f\colon a' \to a \quad g\colon b \to b'}{\backslash\!\!\backslash_{(f,g)}\colon b'\backslash a' \to b\backslash a} \qquad\qquad \frac{f\colon a' \to a \quad g\colon b \to b'}{/\!\!/_{(f,g)}\colon a'/b' \to a/b}$$

$$\frac{g\colon b \to b' \quad f\colon b' \otimes c \to a}{\mathscr{C}^{\backslash}_{(f,g)}\colon c \to b\backslash a} \qquad\qquad \frac{g\colon b \to b' \quad f\colon c \otimes b' \to a}{\mathscr{C}^{/}_{(f,g)}\colon c \to a/b}$$

The following proof of cut elimination for G provides us with the term equations which allow normalization of proof terms. The proof proceeds as the ususal proof of cut elimination for sequent calculus, with the main cases, listed immediately below and the permutation cases in the next section. One assumes, without loss of generality, that the premises $f\colon a \to b$ and $g\colon b \to c$, from which $g \cdot f\colon a \to c$ is derived, have been constructed without cut. The *degree of the cut* is defined as usual as the number of occurrences of each connective in the sequents involved in the cut. The *main cases* properly lower the degree of the cut, while the *permutation cases* percolate the cut upwards. Each instance, provides us with a rewriting rule for proof terms. We call redex the proof on the left and contractum the proof on the right. The term in the root arrow of the redex is rewritten as the term in the root arrow of the contractum. We have finally instance of what we call *cycles* which give rise to further reduction. These arise from cyclic application of the rules and we will see how to eliminate them.

The first case we examine is the trivial cut occurring when one of the premises is an identity axiom.

(7.1) Cut with identities:

$$\begin{array}{ccc}
\textit{Redex:} & & \textit{Contractum:} \\[1ex]
\dfrac{1_a\colon a \to a \quad f\colon a \to b}{f \cdot 1_a\colon a \to b} & \rightsquigarrow & f\colon a \to b \\[3ex]
\dfrac{f\colon a \to b \quad 1_b\colon b \to b}{1_b \cdot f\colon a \to b} & \rightsquigarrow & f\colon a \to b
\end{array}$$

7.1.1 Main cuts

We examine the main cases of cut. These properly reduce the degree of the cut.

(7.2) The cut formula is $a^* \otimes b^*$ and is introduced on both premisses by product introduction rule:

Redex:

$$\dfrac{\dfrac{f\colon a \to a^* \quad g\colon b \to b^*}{f \otimes g\colon a \otimes b \to a^* \otimes b^*} \quad \dfrac{f'\colon a^* \to a' \quad g'\colon b^* \to b'}{f' \otimes g'\colon a^* \otimes b^* \to a' \otimes b'}}{(f' \otimes g') \cdot (f \otimes g)\colon a \otimes b \to a' \otimes b'}$$

Contractum:

$$\dfrac{\dfrac{f\colon a \to a^* \quad f'\colon a^* \to a'}{f' \cdot f\colon a \to a'} \quad \dfrac{g\colon b \to b^* \quad g'\colon b^* \to b'}{g' \cdot g\colon b \to b'}}{(f' \cdot f) \otimes (g' \cdot g)\colon a \otimes b \to a' \otimes b'}$$

(7.3) The cut formula is $a^* \otimes b^*$ and is introduced on the left premise by product introduction rule and on the right by application rule (we omit the symmetric case):

Redex:

$$\dfrac{\dfrac{f\colon a \to a^* \quad g\colon b \to b^*}{f \otimes g\colon a \otimes b \to a^* \otimes b^*} \quad \dfrac{f'\colon a^* \to c/b' \quad g'\colon b^* \to b'}{f'^{/}(g')\colon a^* \otimes b^* \to c}}{f'^{/}(g') \cdot f \otimes g\colon a \otimes b \to c}$$

Contractum:

$$\dfrac{\dfrac{f\colon a \to a^* \quad f'\colon a^* \to c/b'}{f' \cdot f\colon a \to c/b'} \quad \dfrac{g\colon b \to b^* \quad g'\colon b^* \to b'}{g' \cdot g\colon b \to b'}}{(f' \cdot f)^{/}(g' \cdot g)\colon a \otimes b \to c}$$

(7.4) The cut formula is a^*/b^* and is introduced on both premisses by monotonicity of the slash (we omit the symmetric case):

Redex:

$$\dfrac{\dfrac{f : a \to a^* \quad g : b^* \to b}{/\!\!/_{(f,g)} : a/b \to a^*/b^*} \qquad \dfrac{f' : a^* \to a' \quad g' : b' \to b^*}{/\!\!/_{(f',g')} : a^*/b^* \to a'/b'}}{/\!\!/_{(f',g')} \cdot /\!\!/_{(f,g)} : a/b \to a'/b'}$$

Contractum:

$$\dfrac{\dfrac{f : a \to a^* \quad f' : a^* \to a'}{f' \cdot f : a \to a'} \qquad \dfrac{g' : b' \to b^* \quad g : b^* \to b}{g \cdot g' : b' \to b}}{/\!\!/_{(f' \cdot f, g \cdot g')} : a/b \to a'/b'}$$

(7.5) The cut formula is a/b and is introduced on the left premise by the coapplication rule and on the right premise by monotonicity of the slash (we omit the symmetric case):

Redex:

$$\dfrac{\dfrac{g : b \to b^* \quad f : c \otimes b^* \to a}{\mathscr{C}^{/}_{(f,g)} : c \to a/b} \qquad \dfrac{f' : a \to a' \quad g' : b' \to b}{/\!\!/_{(f',g')} : a/b \to a'/b'}}{/\!\!/_{(f',g')} \cdot \mathscr{C}^{/}_{(f,g)} : c \to a'/b'}$$

Contractum:

$$\dfrac{\dfrac{f : c \otimes b^* \to a \quad f' : a \to a'}{f' \cdot f : c \otimes b^* \to a'} \qquad \dfrac{g' : b' \to b \quad g : b \to b^*}{g \cdot g' : b' \to b^*}}{\mathscr{C}^{/}_{(f' \cdot f, g \cdot g')} : c \to a'/b'}$$

As the term notation allows abstraction over the slash orientation, let us use $\|$ as a variable over $\{/\!\!/, \backslash\!\!\backslash\}$. We can capture the above instances in the folowing schematic rewriting rules:

Definition 97. Main reductions.

$$\begin{aligned}
\sigma_{(f',g')} \cdot \delta_{(f,g)} &\rightsquigarrow \sigma_{(f' \cdot f, g' \cdot g)}, \\
&\quad \text{if } \sigma_{(f',g')} \in \{f' \otimes g',\ f'(g')\} \text{ and } \delta_{(f,g)} \in \{f \otimes g\}. \\
\sigma_{(f',g')} \cdot \delta_{(f,g)} &\rightsquigarrow \sigma_{(f' \cdot f, g \cdot g')}, \\
&\quad \text{if } \sigma_{(f',g')} \in \{\|_{(f',g')}\} \text{ and } \delta_{(f,g)} \in \{\|_{(f,g)}, \mathscr{C}_{(f,g)}\}
\end{aligned}$$

7.1.2 Permutations

The cases that we analyze below are distinct from the previous ones in that they do not directly reduce the degree of the cut, but permute the cut upwards, making new cuts visible. In every case, the cut formula is the simpler formula of a simplifying rule. For each case we distinguish various subcases, depending on the rule from which the main premise $f\colon x \to y$ of the main premise of the cut is derived.

(7.6) Case:

$$\dfrac{h\colon c \to c' \qquad \dfrac{g\colon b \to b' \quad f\colon c' \otimes b' \to a}{\mathscr{C}^{/}_{(f,g)}\colon c' \to a/b}}{\mathscr{C}^{/}_{(f,g)} \cdot h\colon c' \to a/b}$$

1. Subcase: $f\colon c' \otimes b' \to a$ derives from product introduction rule.

   Redex:

$$\dfrac{h\colon c \to c' \qquad \dfrac{g\colon b \to b' \quad \dfrac{f^*\colon c' \to c^* \quad f'\colon b' \to a'}{f^* \otimes f'\colon c' \otimes b' \to c^* \otimes a'}}{\mathscr{C}^{/}_{(f^* \otimes f',g)}\colon c' \to (c^* \otimes a')/b}}{\mathscr{C}^{/}_{(f^* \otimes f',g)} \cdot h\colon c \to (c^* \otimes a')/b}$$

   Contractum:

$$\dfrac{1_b\colon b \to b \qquad \dfrac{\dfrac{h\colon c \to c' \quad f^*\colon c' \to c^*}{f^* \cdot h\colon c \to c^*} \quad \dfrac{g\colon b \to b' \quad f'\colon b' \to a'}{f' \cdot g\colon b \to a'}}{(f^* \cdot h) \otimes (f' \cdot g)\colon c \otimes b \to c^* \otimes a'}}{\mathscr{C}^{/}_{((f^* \cdot h)\otimes(f' \cdot g),1_b)}\colon c \to (c^* \otimes a')/b}$$

2. Subcase: $f\colon c'\otimes b'\to a$ derives from the rule of right application.

   Redex:

$$\dfrac{h\colon c\to c' \qquad \dfrac{g\colon b\to b' \qquad \dfrac{f^*\colon c'\to a/a' \quad f'\colon b'\to a'}{f^{*/}(f')\colon c'\otimes b'\to a}}{\mathscr{C}^{/}_{(f^{*/}(f'),g)}\colon c'\to a/b}}{\mathscr{C}^{/}_{(f^{*/}(f'),g)}\cdot h\colon c\to a/b}$$

   Contractum:

$$\dfrac{1_b\colon b\to b \qquad \dfrac{\dfrac{h\colon c\to c' \quad f^*\colon c'\to a/a'}{f^*\cdot h\colon c\to a/a'} \quad \dfrac{g\colon b\to b' \quad f'\colon b'\to a'}{f'\cdot g\colon b\to a'}}{f^*\cdot h^{/}(f'\cdot g)\colon c\otimes b\to a}}{\mathscr{C}^{/}_{(f^*\cdot h^{/}(f'\cdot g),1_b)}\colon c\to a/b}$$

3. Subcase: $f\colon c'\otimes b'\to a$ derives from the rule of left application.

   Redex:

$$\dfrac{h\colon c\to c' \qquad \dfrac{g\colon b\to b' \qquad \dfrac{f^*\colon c'\to a' \quad f'\colon b'\to a'\backslash a}{f'^{\backslash}(f^*)\colon c'\otimes b'\to a}}{\mathscr{C}^{/}_{(f'^{\backslash}(f^*),g)}\colon c'\to a/b}}{\mathscr{C}^{/}_{(f'^{\backslash}(f^*),g)}\cdot h\colon c\to a/b}$$

   Contractum:

$$\dfrac{1_b\colon b\to b \qquad \dfrac{\dfrac{h\colon c\to c' \quad f^*\colon c'\to a'}{f^*\cdot h\colon c\to a'} \quad \dfrac{g\colon b\to b' \quad f'\colon b'\to a'\backslash a}{f'\cdot g\colon b\to a'\backslash a}}{(f'\cdot g)^{\backslash}(f^*\cdot h)\colon c\otimes b\to a}}{\mathscr{C}^{/}_{((f'\cdot g)^{\backslash}(f^*\cdot h),1_b)}\colon c\to a/b}$$

4. Subcase: $f\colon c'\otimes b'\to a$ derives from the rule of coapplication. Then $f\equiv C_{(f',g')}$. Then one applies one of the previous cases to f'.

(7.7) Case:

$$\dfrac{\dfrac{f\colon a\to c'/b' \quad g\colon b\to b'}{f^{/}(g)\colon a\otimes b\to c'} \qquad h\colon c'\to c}{h\cdot(f^{/}(g))\colon a\otimes b\to c}$$

1. Subcase: $f\colon a\to c'/b'$ derives from the rule of monotonicity of slash.

   Redex:

$$\dfrac{\dfrac{\dfrac{f'\colon a'\to c' \quad f^*\colon b'\to a^*}{\sslash_{(f',f^*)}\colon a'/a^*\to c'/b'} \qquad g\colon b\to b'}{(\sslash_{(f',f^*)})^{/}(g)\colon a'/a^*\otimes b\to c'} \qquad h\colon c'\to c}{h\cdot((\sslash_{(f',f^*)})^{/}(g))\colon a'/a^*\otimes b\to c}$$

   Contractum:

$$\dfrac{\dfrac{\dfrac{f'\colon a'\to c' \quad h\colon c'\to c}{h\cdot f'\colon a'\to c} \quad \dfrac{g\colon b\to b' \quad f^*\colon b'\to a^*}{f^*\cdot g\colon b\to a^*}}{\sslash_{(h\cdot f',f^*\cdot g)}\colon a'/a^*\to c/b} \qquad 1_b\colon b\to b}{(\sslash_{(h\cdot f',f^*\cdot g)})^{/}(1_b)\colon a'/a^*\otimes b\to c}$$

2. Subcase: $f\colon a\to c'/b'$ derives from the rule of coapplication.

   Redex:

$$\dfrac{\dfrac{\dfrac{f^*\colon b'\to b^* \quad f'\colon a\otimes b^*\to c'}{\mathscr{C}^{/}_{(f',f^*)}\colon a\to c'/b'} \qquad g\colon b\to b'}{(\mathscr{C}^{/}_{(f',f^*)})^{/}(g)\colon a\otimes b\to c'} \qquad h\colon c'\to c}{h\cdot((\mathscr{C}^{/}_{(f',f^*)})^{/}(g))\colon a\otimes b\to c}$$

   Contractum:

$$\dfrac{\dfrac{\dfrac{g\colon b\to b' \quad f^*\colon b'\to b^*}{f^*\cdot g\colon b\to b^*} \quad \dfrac{f'\colon a\otimes b^*\to c' \quad h\colon c'\to c}{h\cdot f'\colon a\otimes b^*\to c}}{\mathscr{C}^{/}_{(h\cdot f',f^*\cdot g)}\colon a\to c/b} \qquad 1_b\colon b\to b}{(\mathscr{C}_{(h\cdot f',f^*\cdot g)})^{/}(1_b)\colon a\otimes b\to c}$$

3. Subcase: $f\colon a\to c'/b'$ derives from the rule of application. Then $f\equiv f'(g')$. Then one applies one of the previous cases to f'.

We can capture the above instances in the folowing schematic rewriting rules:

Definition 98. Permutations.

If $\sigma_{(f,g)} \in \{\mathscr{C}^{\#}_{(f,g)}\}$, then if $\delta_{(g,h)} \in \{g \otimes h, g^{\#}(h)\}$, then $\sigma_{(\delta_{(f^*,f')},g)} \cdot h \rightsquigarrow \sigma_{(\delta_{(f^* \cdot h, f' \cdot g)},1)}$.

Else, if $\delta_{(g,h)} \in \{g \otimes h, g^{\widetilde{\#}}(h)\}$, then $\sigma_{(\delta_{(f^*,f')},g)} \cdot h \rightsquigarrow \sigma_{(\delta_{(f^* \cdot g, f' \cdot h)},1)}$, where $\widetilde{\#}$ is the symmetric connective of #.

$h \cdot \sigma_{(\delta_{(f',f^*)},g)} \rightsquigarrow \sigma_{(\delta_{(h \cdot f', f^* \cdot g)},1)}$, if $\sigma_{(f,g)} \in \{f(g)\}$ and $\delta_{(f,g)} \in \{||_{(f,g)}, \mathscr{C}_{(f,g)}\}$.

This concludes the analysis of the various instances of cut.

7.1.3 Cycles

The following cases do not deal with cut. However, the cases in the previous section may give rise to further reductions. We call such instances *cycles* and we show how to reduce and eliminate them.

1. Instance:

   Redex:

$$\dfrac{\dfrac{g\colon b^* \to b' \quad f\colon c \otimes b' \to a}{\mathscr{C}^{/}_{(f,g)}\colon c \to a/b^*} \qquad h\colon b \to b^*}{(\mathscr{C}^{/}_{(f,g)})^{/}(h)\colon c \otimes b \to a}$$

   Contractum:

$$\dfrac{\dfrac{\dfrac{h\colon b \to b^* \quad g\colon b^* \to b'}{g \cdot h\colon b \to b'} \qquad f\colon c \otimes b' \to a}{\mathscr{C}^{/}_{(f,g \cdot h)}\colon c \to a/b} \qquad 1_b\colon b \to b}{(\mathscr{C}^{/}_{(f,g \cdot h)})^{/}(1_b)\colon c \otimes b \to a}$$

   Termination:

$$\dfrac{\dfrac{1_b\colon b \to b \quad f\colon c \otimes b \to a}{\mathscr{C}^{/}_{(f,1_b)}\colon c \to a/b} \qquad 1_b\colon b \to b}{(\mathscr{C}^{/}_{(f,1_b)})^{/}(1_b)\colon c \otimes b \to a} \quad \rightsquigarrow \quad f\colon c \otimes b \to a$$

2. Instance:

Redex:

$$\frac{h\colon b \to b^* \qquad \dfrac{f\colon c \to a/b' \quad g\colon b^* \to b'}{f^{/}(g)\colon c \otimes b^* \to a}}{\mathscr{C}^{/}_{(f^{/}(g),h)}\colon c \to a/b}$$

Contractum:

$$\frac{1_b\colon b \to b \qquad \dfrac{f\colon c \to a/b' \qquad \dfrac{h\colon b \to b^* \quad g\colon b^* \to b'}{g \cdot h\colon b \to b'}}{f^{/}(g \cdot h)\colon c \otimes b \to a}}{\mathscr{C}^{/}_{(f^{/}(g \cdot h),1_b)}\colon c \to a/b}$$

Termination:

$$\frac{1_b\colon b \to b \qquad \dfrac{f\colon c \to a/b \quad 1_b\colon b \to b}{f^{/}(1_b)\colon c \otimes b \to a}}{\mathscr{C}^{/}_{(f^{/}(1_b),1_b)}\colon c \to a/b} \qquad \rightsquigarrow \quad f\colon c \to a/b$$

7.2 Conclusion

The combinatorial analysis of the cut elimination algorithm proposed in the previous sections represents the basis for further work which may enlighten the properties of an appropriate term language for encoding normal *NL* proofs. The task is quite an ambitious one, as it aims at expressing equivalence of proofs for the non-associative Lambek calculus. As we saw in the various case analyses, the proof term notation becomes soon rather complex and probably the choice of a simpler notation, if possible, would help. We discussed, on the other hand, the fact that [Lambek, 1993] notation, though slightly simpler, is ambiguous. In this respect, we provided an explicit notation for proof terms isomorphic to *NL* proofs. The equation arising in this analysis represent a further step towards the semantics of *NL* proofs.

Chapter 8

Conclusion

This book presented a thorough study of categorial grammars. We investigated logical, linguistic and computational properties of the non-associative Lambek calculus.

Concerning the logical properties, in chapter 5 we improved on the method of [Kandulski, 1988] for the construction of normal derivations for *NL*, by defining a *finite* procedure for building normal derivations. Furthermore, our procedure is free from spurious ambiguity as we showed in chapter 6. As a result, we calculated the number of readings of a special class of sequents, that we called ϕ-sequents. The connection with the Pascal triangle is a pleasant theoretical result, illustrating the great generative power of *NL*.

With respect to linguistics, the analysis that we presented in chapter 4 should mainly be taken as an exemplification of how *NL* may be applied to natural language analysis. The problems we have dealt with are complex and articulated, as acknowledged by the entire linguistic community. They involve aspect of morphology and prosodic phonology as well as of semantics and pragmatics of discourse which were beyond the scope of our treatment. Thus, we hope at least that our examples were clear and that our approach may lay the ground for further research.

Finally, we studied the computational properties of *NL*. In chapter 3, we designed the CYK parser for Ajdukiewicz Bar-Hillel grammars with product. As the normalization procedure given in chapter 5 converts an *NL* into an equivalent $AB^{\otimes}$ grammar, the converted *NL* grammar can also be parsed in cubic time. In chapter 3, we showed also that one can get rid of the product introduction rule in the parsing process, through a transformation that converts an $AB^{\otimes}$ grammar into an equivalent AB grammar (without product), and still be able to recover the deduction in the original $AB^{\otimes}$ via a simple term normalization procedure, whose complexity is linear. We also applied the technique of shared forest to the parse table returned by the CYK recognition algorithm for $AB^{\otimes}$. Since one is usually interested in retrieving *all* readings of a categorial sequent, this technique represents an efficient solution of this problem.

The term equations arisen in the application of the cut-elimination algorithm in chapter 7 are an interesting aspect of proof theoretic investigation of *NL* which deserve further study. As we explained, lambda terms do not offer an appropriate semantics of *NL* proofs and only a few works on term languages isomorphic to *NL* proofs have been proposed. The proof terms that we have developed and the reductions we identified by applying the cut elimination algorithm are a further step towards an explicit semantics of proofs for *NL*.

Bibliography

S. Adams. Functional pearls: Efficient sets—a balancing act. *Journal of Functional Programming*, 3(4):553–561, October 1993.

A. Aho and J. Ullman. *The Theory of Parsing, Translation and Compiling*, volume 1: Parsing. Prentice-Hall, INC., 1972.

A. V. Aho. Indexed grammars - An extension of context free grammars. In *FOCS*, pages 21–31. IEEE, 1967.

K. Ajdukiewicz. Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27, 1935.

Y-M. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.

Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.

Y. Bar-Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information. Selected Essays on their Theory and Application*, pages 99–115. Addison-Wesley, Reading, MA, 1964a.

Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, pages 116–150. Addison-Wesley Publishing Co., 1964b.

R. Bernardi. *Reasoning with polarity in categorial type logic*. PhD thesis, UiL-OTS, Utrecht, 2002.

S. Billot and B. Lang. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th annual meeting on Association for Computational*

Linguistics, pages 143–151, Morristown, NJ, USA, 1989. Association for Computational Linguistics.

P. Blackburn and J. Bos. Computational semantics for natural language. Course notes for NASSLLI 2003, Indiana University, 2003.

M. Buliśka. P-TIME decidability of NL1 with assumptions. In *FG2006: The 11th Conference on Formal Grammar*, pages 29–38, 2006. URL `http://cs.haifa.ac.il/~shuly/fg06/FG.pdf`.

W. Buszkowski. Lambek calculus with nonlogical axioms. In C. Casadio, P. Scott, and R. Seely, editors, *Language and Grammar: Studies in Mathematical Linguistics and Natural Language*, pages 77–93. CSLI Lecture Notes 168, Stanford, 2005.

W. Buszkowski. Generative capacity of the nonassociative Lambek calculus. *Bulletin of the Polish Academy of Sciences, Mathematics*, 34:507–516, 1986.

W. Buszkowski. Gaifman's theorem on categorial grammars revisited. *Studia Logica*, 47:23–33, 1988.

W. Buszkowski. Mathematical linguistics and proof theory. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 683–736. Elsevier, Amsterdam, 1997.

B. Carpenter. The turing-completeness of multimodal categorial grammars, 1996. URL `citeseer.ist.psu.edu/194254.html`.

C. Casadio. *Logic for Grammar. Developments in Linear Logic and Formal Linguistics*. Bulzoni, Roma, 2002.

C. Casadio. Semantic categories and the development of categorial grammars. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammar and Natural Language Structures*. Reidel, Dodrecht, 1988.

C. Casadio and J. Lambek. An algebraic analysis of clitic pronouns in italian. In P. de Groote, G. Morrill, and C. Retoré, editors, *Logical Aspects of Computational Linguistics*, Lecture Notes in Computer Science, pages 110–124. Springer, 2001.

Claudio Casadio and Joachim Lambek. A tale of four grammars. *Studia Logica*, 71(3):315–329, 2002.

N. Chomsky. *Syntactic Structures*. Mouton and Co., The Hague, 1957.

N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

N. Chomsky. Formal properties of grammars. In *Handbook of Mathematical Psychology*, volume 2, pages 323–418. J. Wiley and Sons, New York, 1963.

N. Chomsky. *Aspects of the theory of syntax*. The MIT Press, 1965.

N. Chomsky. *The Minimalist Program*. The MIT Press, Cambridge, Massachusetts, 1995.

T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

H. B Curry and R. Feys. *Combinatory Logic I*. North-Holland, Amsterdam, 1958.

L. Damas and R. Milner. Principal type-schemes for functional programs. In *POPL*, pages 207–212, 1982.

F. de Groote. The non-associative Lambek calculus with product in polynomial time. In N. V. Murray, editor, *Lecture Notes in Artificial Intelligence*, volume 1617. Springer-Verlag, 1999.

Philippe de Groote and François Lamarche. Classical non-associative lambek calculus. *Studia Logica*, 71(3):355–388, 2002.

S. Degeilh and A. Preller. Efficiency of pregroups and the French noun phrase. *Journal of Logic, Language, and Information*, 14(4):423–444, 2005.

K. Doets and J. van Eijck. *The Haskell Road to Logic, Maths and Programming*. King's College Publications, 2004.

K. Došen. A brief survey of frames for the Lambek calculus. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 38:179–187, 1992.

D. R. Dowty, R. E. Wall, and S. Peters. *Introduction to Montague Semantics*. Reidel, Dordrecht, 1981.

J. Earley. *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1968.

J. Earley. An efficient context-free parsing algorithm. *Comm. ACM*, 13: 94–102, 1970.

A. Finkel and I. Tellier. A polynomial algorithm for the membership problem with categorial grammar. *Theoretical Computer Science*, 164:207–221, 1996.

G. Gazdar. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel, Dordrecht, 1988.

G. Gazdar, E. Klein, G. Pullum, and I. Sag. *Generalized Phrase Structure Grammar*. Basil Blackwell, 1985.

J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

J.-Y. Girard. Linear logic: Its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, Workshop on Linear Logic, 1993, pages 1–42. Cambridge Univ. Press, 1995.

J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*. Cambridge University Press, 1989.

C. Hankin. *An Introduction to Lambda Calculi for Computer Scientist*. King's College Publications, 2004.

M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachussets, 1978.

I. Heim. File change semantics and the familiarity theory of definiteness. In Rainer Bäuerle, Christoph Schwarze, and Arnim von Stechow, editors, *Meaning, Use, and Interpretation of Language*, pages 164–189. Walter de Gruyter, Berlin, 1983.

H. Hendriks. *Studied Flexibility: Categories and Types in Syntax and Semantics*. PhD thesis, ILLC, Amsterdam, 1993.

H. Hendriks. The logic of tune. A proof-theoretic analysis of intonation. In A. Lecomte, editor, *Logical Aspects of Computational Linguistics*, New York, 1999. Springer.

H. Hendriks and P. Dekker. Links without locations. In P. Dekker and M. Stokhof, editors, *Proceedings of the Tenth Amsterdam Colloquium*, 1998.

M. Hepple. Chart parsing Lambek grammars: Modal extensions and incrementality. In *Proceedings of COLING-92*, pages 134–140, 1992. URL `http://acl.ldc.upenn.edu/C/C92/C92-1024.pdf`.

M. Hepple. A compilation-chart method for linear categorial deduction. In *Proceedings of COLING-96*, pages 537–542, Copenhagen, 1996.

M. Hepple. An earley-style predictive chart parsing method for Lambek grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pages 465–472, Maryland, June 1999.

D. Heylen. *Types and Sorts. Resource logic for feature checking*. PhD thesis, UiL-OTS, Utrecht, 1999.

J. R. Hindley. *Basic Simple Type Theory*. Cambridge University Press, 1997.

G. Jäger. Anaphora and quantification in categorial grammar. In M. Moortgat, editor, *Logical Aspects of Computational Linguistics*, volume 2014, Lecture Notes in Computer Science. Springer-Verlag, 2001.

S. Peyton Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

A. K. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions – tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing – Theoretical, Computational and Psychological Perspectives*, pages 206–250. Cambridge University Press, 1985.

A. K. Joshi, L. S. Levy, and M. Takahashi. Tree adjunct grammars. *JCSS: Journal of Computer and System Sciences*, 10, 1975.

M. Kandulski. The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:41–52, 1988.

T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Res. Lab., Bedford Mass.,, 1965.

E. König. A hypothetical reasoning algorithm for linguistic analysis. *Journal of Logic and Computation*, 4(1):1–19, February 1994.

E. Kraak. French object clitics: a multimodal analysis. In G. Morrill and R. T. Oehrle, editors, *Formal Grammar*, 1995.

G.-J. M. Kruijff and J. Baldridge. Multi-modal combinatory categorial grammar. In *EACL*, pages 211–218, 2003. URL `http://acl.ldc.upenn.edu/E/E03/E03-1036.pdf`.

N. Kurtonina. *Frames and labels. A modal analysis of categorial inference*. PhD thesis, UiL-OTS, Utrecht, 1995.

N. Kurtonina and M. Moortgat. Structural control. In P. Blackburn and M. de Rijke, editors, *Specifying Syntactic Structures*, pages 75–113. CSLI, Stanford, 1997.

J. Lambek. Type grammars as pregroups. *Grammars*, 4(1):21–39, 2001.

J. Lambek. The mathematic of sentence structure. *American Mathematical Monthly*, 65(3):154–170, 1958.

J. Lambek. On the calculus of syntactic types. In R. Jacobson, editor, *Proceedings of the Twelfth Symposium in Applied Mathematics*, volume XII, pages 166–178, 1961.

J. Lambek. Categorial and categorical grammars. In R. T. Oehrle et al., editor, *Categorial Grammars and Natural Language Structures*, pages 297–317. Reidel, Dodrecht, 1988.

J. Lambek. Logic without structural rules (another look at cut elimination). In P. Schroeder-Heister and K. Dosen, editors, *Substructural Logic*, pages 179–206. Claredon Press, Oxford, 1993.

J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Cambridge University Press, 1987.

Y. Le Nir. From NL grammars to AB grammars. In M. Moortgat and V. Prince, editors, *CG2004 Proceedings*, Montpellier-France, 2004.

Y. Le Nir. *Structures des analyses syntaxiques catégorielles. Application à l'inférence grammaticale*. PhD thesis, Université de Rennes 1, Rennes, 2003.

P. Monachesi. *A lexical approach to Italian cliticization.* CSLI publications, Stanford, 1999.

R. Montague. Universal grammar. *Theoria*, 36:373–398, 1970a. Reprinted in [Montague, 1974].

R. Montague. English as a formal language. In *Linguaggi nella Società e nella Tecnica*, pages 189–224. Edizioni di Comunità, Milan, 1970b. Reprinted in [Montague, 1974].

R. Montague. *Formal Philosophy: Selected Papers of Richard Montague*. Yale, 1974. Edited and with an introduction by R. Thomason.

M. Moortgat. Generalized quantification and disconinuous type constructors. In H. Bunt and A. van Horck, editors, *Proceedings of the Tilburg Symposium on Discontinuous Dependencies*, Berlin, 1997a. Mouton de Gruyter.

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

M. Moortgat. Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5(3/4):349–385, 1996.

M. Moortgat. Categorial type logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam, 1997b.

M. Moortgat and G. Morrill. Heads and phrases. Type calculus for dependency and constituent structure. OTS Research Papers, 1991.

M. Moortgat and R.T. Oehrle. Proof nets for the grammatical base logic. In V.M. Abrusci and C. Casadio, editors, *Proceedings of the IV Roma Workshop*, Roma, 1997. Bulzoni Editore.

R. Moot. *Proof Nets for Linguistic Analysis*. PhD thesis, UiL-OTS, Utrecht, 2002.

G. Morrill. *Type Logical Grammar: Categorial Logic of Signs*. Kluwer, Dordrecht, 1994.

G. Morrill. Memoisation of categorial proof nets: Parallelism in categorial processing. In V. M. Abrusci and C. Casadio, editors, *Proofs and Linguistic Categories,* Proceedings 1996 Roma Workshop, pages 157–169, Bologna, 1996. Cooperativa Libraria Universitaria Editrice.

M.-J. Nederhof and G. Satta. Tabular parsing. In C. Martin-Vide, V. Mitrana, and G. Paun, editors, *Formal Languages and Applications, Studies in Fuzziness and Soft Computing 148*, pages 529–549. Springer, 2004.

R. T. Oehrle. Multi-dimensional compositional functions as a basis for grammatical analysis. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammar and Natural Language Structures*. Reidel, Dodrecht, 1988.

C. Okasaki. *Purely Functional Data Structures*. Cambridge University Press, Cambridge, England, 1998.

C. Okasaki. Red-black trees in a functional setting. *Journal of Functional Programming*, 9(4):471–477, 1999.

B. H. Partee, A. ter Meulen, and R. E. Wall. *Mathematical Methods in Linguistics*. Kluwer Academic Publishers, 1990.

M. Pentus. Lambek calculus is NP-complete. CUNY Ph.D. Program in Computer Science Technical Report TR–2003005, CUNY Graduate Center, New York, May 2003. http://www.cs.gc.cuny.edu/tr/techreport.php?id=79.

M. Pentus. Lambek calculus is np-complete. *Theoretical Computer Science*, 357(1-3):186–201, 2006.

M. Pentus. Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Los Alamitos, California, 1993. IEEE Computer Society Press.

F. C. N. Pereira and D. H. D. Warren. Parsing as deduction. In *Proceedings of 21st Annual Meeting of the Association for Computational Linguistics*. MIT, June 1983.

J. Pierrehumbert and G. Hirshberg. The meaning of intonational contours in the interpretation of discourse. In J. Morgan P. Cohen and M. Pollack, editors, *Intentions in Communication*. The MIT Press, Cambridge, 1990.

C. Pollard and I. A. Sag. *Head-driven Phrase Structure Grammar*. University of Chicago Press, Chicago, IL, 1994.

T. Reinhart. Pragmatics and linguistics: an analysis of sentence topics. *Philosophica*, 27:53–116, 1982.

A. Sanfilippo. Thematic accessibility in discontinuous dependencies. In E. Engdahl and M. Reape, editors, *Parametric Varition in Germanic and Romance: Preliminary Investigations*, pages 87–99. ESPRIT Basic Research Project 3175, Dynamic Interpretation of Natural Language, DYANA Deliverable R1.1.A, ECCS, University of Edinburg, 1990.

Y. Savateev. The derivability problem for lambek calculus with one division, 2006. URL `http://www.phil.uu.nl/preprints/ckipreprints/PREPRINTS/preprint056.pdf`.

S. M. Shieber, Y. Schabes, and F. C. N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.

K. Sikkel. *Parsing schemata*. PhD thesis, Dept. of Computer Science, University of Twente, Enschede, NL, 1993.

K. Sikkel. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 199, 1998.

Edward Stabler. Derivational minimalism. In Christian Retoré, editor, *Logical Aspects of Computational Linguistics*, pages 68–95, Berlin, 1997. Springer. LNAI 1328.

M. Steedman. Information structure and the syntax-phonology interface. *Linguistic Inquiry*, 31(4):649–689, 2000a.

M. Steedman. *The Syntactic Process*. The MIT Press, 2000b.

H.-J. Tiede. Lambek calculus proofs and tree automata. In Michael Moortgat, editor, *LACL*, volume 2014 of *Lecture Notes in Computer Science*, pages 251–265. Springer, 1998. ISBN 3-540-42251-X. URL `http://link.springer.de/link/service/series/0558/bibs/2014/20140251.htm`.

H-J. Tiede. Counting the number of proofs in the commutative Lambek calculus. In J. Gerbrandy, M. Marx, M. de Rijke, and Y. Venema, editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Amsterdam University Press, Amsterdam, 1999.

E. Vallduvi. *The Informational Component*. PhD thesis, University of Pennsylvania, Philadelphia, 1990.

J. van Benthem. *Language in Action: Categories, Lambdas and Dynamic Logic*. The MIT Press, 1991.

J. van Eijck. Computational semantics and type theory, 2003. URL `http://homepages.cwi.nl/~jve/cs/`.

J. van Eijck. Deductive parsing in haskell, 2004. URL `http://homepages.cwi.nl/~jve/papers/04/parsing/DP.pdf`.

J. van Eijck. Deductive parsing with sequentially indexed grammars, 2005. URL `http://homepages.cwi.nl/~jve/papers/05/sig/DPS.pdf`.

W. Vermaat. *The logic of variation. A cross-linguistic account of wh-question formation*. PhD thesis, UiL-OTS, Utrecht, 2005.

K. Vijay-Shanker. *A study of tree adjoining grammars*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1987.

K. Vijay-Shanker and D. J. Weir. Polynomial time parsing of combinatory categorial grammars. In *ACL*, pages 1–8, 1990.

P. Wadler. The essence of functional programming. In R. Sethi, editor, *Proceedings of the 19th Annual Symposium on Principles of Programming Languages*, pages 1–14, Albuquerque, January 1992. ACM Press. ISBN 0-89791-453-8 /0-89791-453-8.

D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10:189–208, 1967.

W. Zielonka. Axiomatizability of Ajdukiewicz-Lambek calculus by means of cancellation schemes. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:215–224, 1981.