Parsing with Structure-Preserving Categorial Grammars

Published by
LOT
Janskerkhof 13
3512 BL Utrecht
The Netherlands

Phone: +31 30 253 6006
Fax: +31 30 253 6406
e-mail: lot@let.uu.nl
http://www.lotschool.nl/

Cover illustration: Melancholia I, A. Dürer 1514, detail.

ISBN 978-90-78328-33-9
NUR 616

This dissertation is typeset using LaTeX.

Parsing with Structure-Preserving Categorial Grammars

Zinsontleding met Structuurbehoudende Categoriale Grammatica's

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit Utrecht
op gezag van de rector magnificus, prof. dr. W. H. Gispen,
ingevolge het besluit van het college voor promoties
in het openbaar te verdedigen
op 9 juli 2007
des middags te 4.15 uur

door

Matteo Capelletti

geboren op 4 januari 1977 te Cesena, Italië

Promotores: Prof.dr. M.J. Moortgat
Prof.dr. D.J.N. van Eijck

To my sister Beatrice.

Acknowledgments

EXACTLY four years ago, I came to the Netherlands "to study categorial grammar with Professor Moortgat". During this time the expectations of those first days have never been disappointed nor has the excitement decreased. I had the chance to study most interesting topics in a very pleasant environment and with greatly talented people. I wish here to express my gratitude for the opportunity that I have been given.

First of all, I wish to thank warmly my supervisors Michael Moortgat and Jan van Eijck for their help and guidance. Michael followed my studies with great care, trust and enthusiasm since the beginning. Jan guided me through the study of functional programming, logic and parsing, with kindness and patience. It was a privilege and a great pleasure to work with them.

Secondly, I wish to thank the members of the thesis committee, Maciej Kandulski, Mati Pentus, Christian Retoré, Doaitse Swierstra and Albert Visser for taking the time to read this dissertation with care.

Special thanks go to Gianluca Giorgolo, for a careful reading of the manuscript and for helpful comments and discussions; to Richard Moot for suggestions and clarifications on several points, and to Øystein Nilsen for many precious hints and advice.

I also owe much to Raffaella Bernardi and Claudia Casadio who encouraged and helped me to come to Utrecht; to Herman Hendriks who taught fascinating courses on Montague grammar, and to Willemijn Vermaat, my colleague at OTS working on type-logical grammar.

I also wish to thank the various office mates across the years for creating a pleasant and stimulating working environment: Bert Le Bruyn, Michiel Hildebrand, Heleen Hoekstra, Olga Khomitsevich, Marjo van Koppen, Øystein Nilsen and Yoad Winter.

During these years, I have spent some fruitful research periods in Bordeaux. Thereby I wish to thank the members of the team Signes: Maxime Amblard, Roberto Bonato, Alain Lecomte, Yannick Le Nir, Renaud Marlet, Richard Moot, Christian Retoré and the rest of the team for receiving me and exchanging valuable ideas.

Also many other people have helped me with my work, sometimes by making useful remarks and criticisms, sometimes just by showing interest in my

ideas and encouraging me. My thanks go to Wojciech Buszkowski Philippe de Groote, Henriette de Swart, Maurizio Ferriani, Nissim Francez, Maciej Kandulski, Joachim Lambek, Glyn Morrill, Reinhard Muskens, Dick Oehrle, Aarne Ranta, Greg Restall, Giorgio Sandri, Ed Stabler, Mark Steedman, Lutz Straßburger and Walter Tega.

I also wish to thank the following friends and colleagues: Anna Asbury, Fabian Battaglini, Natalie Boll, Ivana Brasileiro, Christophe Costa-Florencio, Joke Delange, Alexis Dimitriadis, Jakub Dotlacil, Paola Escudero, Martin Everaert, Mario Fadda, Berit Gehrke, Nicole Gregoire, Nino Grillo, Christina Hunger, Judith Kamalski, Annemarie Kerkhoff, Cem Keskin, Esther Kraak, Huib Kranendonk, Shakuntala Mahanta, Marijana Marelj, Anna Mlynarczyk, Paola Monachesi, Iris Mulders, Rick Nouwen, Christina Paoletti, Dimitra Papangeli, György Rákosi, Marco Prandoni, Oren Sadeh-Leicht, Kakhi Sakhltkhutsishvili, Maaike Schoorlemmer, Anca Sevcenco, Natalia Slioussar, Giorgos Spathas, Fabio Tamburini, Roberta Tedeschi, Sharon Unsworth, Daphne van Weijen, Mario van der Visser, Nada Vasic, Nina Versteeg, Nadja Vinokurova and Evangelia Vlachou.

I thank Jet and Rien for their hospitality and generosity, and for making me feel at home.

My parents Gabriele and Francesca I wish to thank for being close to me even from far away. To Stefano Pasini I am grateful for his always encouraging words. To the rest of my family in Italy, nonni, nonne, zii, cugine, and other relatives and friends I owe a great deal for surrounding me with laughter, care and food every time I go back home.

I thank Saara for her constant presence even from afar, for her support and encouragement.

This book is dedicated to my beloved sister Beatrice (1980-2005) who always believed in me.

Contents

Chapter 1

Introduction

The term *categorial grammar* includes various linguistic formalisms which share the assumption that expressions may belong to either *complete* or *incomplete* syntactic categories. Such theoretical assumption is reflected in the functional notation adopted for encoding categories. Apart from this common feature, the methods and the tools of the various frameworks may be substantially different both for theoretical and computational properties.

On one side, the so called *combinatory categorial grammars* of Steedman [2000b] are rooted in the combinatorial tradition of Ajdukiewicz [1935] and Bar-Hillel [1953]. These systems adopt a small set of schematic composition rules, called *combinators*, encoding different forms of functor-argument application.

On the other, there are the *logical grammars* of Morrill [1994] and Moortgat [1997] among others, stemming from the work of Lambek [1958, 1961]. These systems interpret the language generation process as a full-fledged *logical* deduction.

The rules of combinatory categorial grammar, CCG for short, only compose simpler structures into larger ones[1], according to generalized oriented variants of the rule of *modus ponens*:

$$\frac{\mathsf{g}\colon \mathsf{p} \quad \mathsf{f}\colon \mathsf{p} \Rightarrow \mathsf{q}}{(\mathsf{f}\ \mathsf{g})\colon \mathsf{q}}$$

Type-logical grammars (TLG) have, together with the composition rules, rules that *decompose* complex structures into simpler ones: variants of the introduction rule for the implication:

$$\frac{\begin{array}{c}[\mathsf{x}\colon \mathsf{p}]\\ \vdots \\ \mathsf{g}\colon \mathsf{q}\end{array}}{\lambda \mathsf{x}.\mathsf{g}\colon \mathsf{p} \Rightarrow \mathsf{q}}$$

[1] We are in fact oversimplifying, since also CCG has forms of type raising rules. We will largely discuss this kind of rules in the third part of this book.

This difference has rather drastic consequences for the theoretical and computational properties of the two systems.

With regard to the theoretical foundation of the grammar system, Lambek style categorial grammars are grammars based on *logical systems* which can be proved to be *complete* with respect to the appropriate models, as shown in [Kurtonina, 1995], [Kurtonina and Moortgat, 1997], [Pentus, 1995] and [Buszkowski, 1997].

As for the computational properties, the rule based CCG framework can easily make use of the efficient parsing methods which have been developed for context-free grammars, as shown, for instance, in [Vijay-Shanker and Weir, 1990]. Instead, for TLG even the task of recognizing a sequent whose structure is given, a task which has hardly any sense for context-free or combinatory categorial grammar, can be computationally costly. In fact, even a sequent consisting of two formulas (the antecedent and the succedent) may have a number of non-equivalent proofs which is exponential on its length. Thus the size of the search space in automatic recognition may easily grow beyond a tractable size.

In addition to functors, the structure of syntactic categories can be extended to include complex categories which result from 'merging' simpler categories without function-argument application. Thus, for instance, we may say that if the expression w_1 is of category a and the expression w_2 is of category b, then the expression $w_1 w_2$ is of category $a \otimes b$. Such product categories can be a valuable tool for enforcing specific constituent structures and therefore for the structural adequacy of the linguistic description. On the other hand, an expression $w_0 \dots w_n$, such that each w_i is assigned the category a_i, can be assigned C_n product categories, where C_n is the Catalan number of n. Such combinatorial explosion is a problem that may affect both CCG[2] and TLG *with product*.

In this book, I will give a solution to these computational problems for the most basic kind of categorial *logic*: the pure logic of residuation, also known as non-associative Lambek calculus, since it was formulated for the first time by Lambek [1961]. From the type-logical perspective, any other logic can be obtained from the pure logic of residuation by addition of *structural* rules, that is to say, rules which *change* the structure of the categories. Hence the expression *structure-preserving* in the title of this book.

The main results contained in this book are the followings:

1. A function calculating the number of possible readings of a sequent on the basis of its length.

2. An efficient method for handling product categories in the parsing process.

[2]Although CCG are customarily presented as product-free system, the extension to the product would be rather natural.

3. A redundancy-free proof construction method.

4. A redundancy-free polynomial chart-parser for grammars based on the non-associative Lambek calculus.

1.1 Overview

This book is organized as follows.

In Chapter 2, I introduce the formalisms of categorial grammar and of context-free grammar, and most of the basic notions that will be used throughout the book.

In Chapter 3, I define parsers for context-free and Ajdukiewicz–Bar-Hillel categorial grammars with product. Parsers are formulated here as deductive systems in the style of the deductive parsers of [Shieber et al., 1995] and [Sikkel, 1993]. The implementation and complexity of these *parsing systems* are studied in Chapter 4.

Chapter 5 presents a simple method for constructing normal derivations in the non-associative Lambek calculus. *Normal*, here, means *almost* the same as in [Buszkowski, 1986] and [Kandulski, 1988]: a proof of a sequent $\mathfrak{a} \to \mathfrak{c}$ consists in a series of reducing-complexity transitions followed by a series of expanding-complexity transitions. However, it is *not* exactly the same. In the sense of these authors, *normal* does not imply uniqueness. We will see in Chapter 6 that there can be several equivalent normal derivations, in Buszkowski and Kandulski's sense. Instead, the procedure that I formulate in Definition 5.9, in Chapter 5, generates only non-equivalent derivations as I prove in Chapter 6.

The topic of Chapter 6 is that of ambiguity, both spurious and non. I address also the question of the number of readings of a sequent in the non-associative Lambek calculus. In this respect, I establish the connection between the number of readings of a sequent and the binomial coefficient in Section 6.2.

Chapter 7 is concerned with the complexity of the non-associative Lambek calculus. It is well know, from [de Groote, 1999] and [Buszkowski, 2005], that this system can be parsed in polynomial time. My contribution is a detailed description of a polynomial recognition procedure which satisfies some of the desiderata of [de Groote, 1999]. Besides I lower the previous polynomial bounds by adopting a stronger notion of subformula.

Chapter 2

Background

In this chapter, I introduce the basic notions of formal language theory that will accompany us throughout the book. Context-free grammars and categorial grammars are formulated as *deductive systems*, following the *parsing-as-deduction* methodology of [Pereira and Warren, 1983]. I will present associative and non-associative Ajdukiewicz–Bar-Hillel grammars, the associative and non-associative Lambek calculus and the framework of multi-modal type-logical grammars.

2.1 Languages

Languages are defined from their terminal elements, often called *words* or more in general *symbols*.

Definition 2.1. 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 $\{\mathsf{a}\}$. Let us give some example of vocabularies.

Example 2.1. Vocabularies:

- $\mathsf{V}_0 = \{0, 1\}$, where $0, 1$ are of type $\mathtt{Int}$(eger).
- $\mathsf{V}_1 = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, where $\mathsf{a}, \mathsf{b}, \mathsf{c}$ are of type $\mathtt{Char}$(acter).
- $\mathsf{V}_2 = \{$ *John*, *Mary*, *every*, *a*, *man*, *woman*, *loves*, *ismissing* $\}$, where each element of V_2 is a *list* of characters.

We saw that in the vocabulary V_2, symbols are structured objects: lists of objects of type $\mathtt{Char}$. Let us introduce precisely the notion of list.

Definition 2.2. A *list* of objects of type a, denoted $\mathtt{List}\,\mathsf{a}$, is defined as

$$\mathtt{List}\,\mathsf{a} := \epsilon \mid \mathsf{H}\,\mathsf{a}\,(\mathtt{List}\,\mathsf{a})$$

This definition states that a list of objects of type $\mathtt{a}$ is either ϵ (the empty list) or the result of the application of the *constructor* H to an object of type $\mathtt{a}$ and to an object of type $\mathtt{List\,a}$ (a list of objects of type $\mathtt{a}$). Following the Haskell syntax, [Jones, 2003], instead of $\mathtt{List\,a}$, we use the notation $[\mathtt{a}]$ 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 $\mathtt{f}$ can be made infix by writing $\mathtt{`f`}$. While an infix function $\mathtt{g}$ is made prefix by writing $(\mathtt{g})$. Thus, $\mathsf{H} = (:)$ and $: = \mathtt{`H`}$. The *type* of this function is written $(:) :: \mathtt{a} \to [\mathtt{a}] \to [\mathtt{a}]$, which means that $(:)$ is a function that takes in input an argument $\mathtt{x}$ of type $\mathtt{a}$, and returns a function that takes in input a list $[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ of objects of type $\mathtt{a}$ and returns as output a list $[\mathtt{x}, \mathtt{x}_1, \ldots, \mathtt{x}_n]$ of objects of type $\mathtt{a}$. Thus, for example, we have the following equivalences.

$$\begin{array}{lcl} [\,] & = & \epsilon \\ \mathtt{a}:\mathtt{as} & = & \mathsf{H}\,\mathtt{a}\,\mathtt{as} \\ [\mathtt{a},\mathtt{b},\mathtt{c}] & = & \mathtt{a}:\mathtt{b}:\mathtt{c}:[\,] \;=\; \mathsf{H}\,\mathtt{a}\,(\mathsf{H}\,\mathtt{b}\,(\mathsf{H}\,\mathtt{c}\,\epsilon)) \end{array}$$

As usual, we use $\_$ for the *wildcard*. We define the length of a list $\mathtt{xs}$, denoted $|\mathtt{xs}|$:

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

Definition 2.3. The *concatenation*, $(+\!\!+) :: [\mathtt{a}] \to [\mathtt{a}] \to [\mathtt{a}]$, of two lists $\mathtt{xs}$ and $\mathtt{ys}$ is defined as follows.

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

List concatenation has the following properties.

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

Definition 2.4.

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

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

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

Example 2.2. Languages:

- $\mathsf{V}_0^* = \{[], [0], [1], [0,1], [0,0], [1,1], [0,0,0], \ldots\}$,
- $\mathsf{V}_0^+ = \{[0], [1], [0,1], [0,0], [1,1], [0,0,0], \ldots\}$,

- $\mathsf{L}_1 = \{[0], [1], [0, 1, 1, 1]\}$ is a (finite) language over V_0.

By *string*, we mean a list of symbols of some type. In order to simplify the notation, we write a list of `Char` type objects, $[\mathtt{a}, \mathtt{b}, \mathtt{c}]$, as `abc` and the concatenation of two lists `abc` and `de` simply as `abcde`. This convention is 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 lists of characters [*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 sets of lists of objects.

Definition 2.5. If L_1 and L_2 are two languages of type $\{[\mathtt{a}]\}$, we write $\mathsf{L}_1\mathsf{L}_2$ the language consisting of all the strings $\mathtt{xy}$ such that $\mathtt{x} \in \mathsf{L}1$ and $\mathtt{y} \in \mathsf{L}_2$.
$(\overline{+\!+}) :: \{[\mathtt{a}]\} \to \{[\mathtt{a}]\} \to \{[\mathtt{a}]\}$

$$\mathsf{L}_1 \overline{+\!+} \mathsf{L}_2 = \{\mathtt{x} +\!+ \mathtt{y} \mid \mathtt{x} \in \mathsf{L}_1,\ \mathtt{y} \in \mathsf{L}_2\}$$

2.2 Grammars

According to Chomsky [1957, 1959], 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. A grammar can be seen as a kind of deductive system, subject to specific constraints. I will define deductive systems below. Then, we will examine context-free grammars and categorial grammars as instances of deductive systems.

Deductive systems

We start this section by introducing syntactic categories.

Definition 2.6. *Syntactic categories.*
Let a set of *atoms*, A be defined as

$$\mathsf{A} := \mathsf{S} \mid \mathsf{NP} \mid \mathsf{VP} \mid \mathsf{N} \mid \ldots \mid \mathtt{a}_1 \mid \mathtt{a}_2 \mid \ldots$$

Furthermore, let

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

Later we will extend the data type $\mathcal{F}$ of formulas with other type constructors for categorial formulas. Formulas, or categories, are the basic objects that the deductive systems that we are going to define will manipulate. We denote $\Gamma[\Delta]$ an element of $\mathcal{F}^*$ with a distinguished occurrence of $\Delta \in \mathcal{F}^*$. The result of replacing Δ for Λ in $\Gamma[\Lambda]$ is $\Gamma[\Delta]$.

Definition 2.7. A *deductive system* D is a triple $\langle \mathcal{F}, \mathsf{AX}, \mathsf{R} \rangle$, where

- AX is a set of pairs $\langle \Gamma, \Delta \rangle$ such that $\Gamma \in \mathcal{F}^*$ and $\Delta \in \mathcal{F}^*$, the set of axioms. We write each $\langle \Gamma, \Delta \rangle \in \mathsf{AX}$ as $\Gamma \to \Delta$.
- R is the set of the inference rules proper to the system. Such rules are conditionals 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*. We remark that what appears in the rules are *variables* over sequents.

Definition 2.8. A *tree* of objects of type $\mathtt{a}$, denoted $\mathtt{Tree\ a}$, is defined as follows[1]

$$\mathtt{Tree\ a} := \mathtt{Branch\ a\ [Tree\ a]}$$

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

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

However, we will usually work with unary and binary trees.

Definition 2.9. Let a deductive system $\mathsf{D} = \langle \mathcal{F}, \mathsf{AX}, \mathsf{R} \rangle$ be given. We recursively define a *deduction* in D as a tree of sequents such that:

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

We introduce also the notions of generation and of language generated by a deductive system.

[1] Such representation of a tree is also called a *rose*.

Definition 2.10.

A deductive system D *generates* a sequent $\Gamma \to \Delta$, denoted $\vdash_{\mathsf{D}} \Gamma \to \Delta$, if $\Gamma \to \Delta$ is the root of a deduction in D.

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

In the context of phrase structure grammars, the notion of *derivation* is used more often than the one of deduction. Let us introduce also the following definitions.

Definition 2.11.

A *rewriting system* $\mathcal{R}$ is a pair $\langle \mathcal{F}, \mathsf{AX} \rangle$ such that AX is as in Definition 2.7.

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

$$\Gamma[\Lambda] \Rightarrow \Gamma[\Delta] \quad \text{if} \quad \Lambda \to \Delta \in \mathsf{AX}.$$

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 $\vdash_{\mathcal{R}} \Gamma \to \Delta$ if and only if $\Gamma \Rightarrow^* \Delta$.

An immediate consequence of Definition 2.11 is the following.

Proposition 2.1. 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 $\Lambda \Rightarrow \Delta \equiv \Delta'[\Omega'] \Rightarrow \Delta'[\Omega]$ and $\Omega' \to \Omega \in \mathsf{AX}$, 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 2.2. Let $\mathcal{R} = \langle \mathcal{F}, \mathsf{AX} \rangle$ be a rewriting system and $\mathsf{D} = \langle \mathcal{F}, \mathsf{AX}, \{\mathsf{Cut}\} \rangle$ a deductive system such that Cut is the following rule

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

Then

$$\vdash_{\mathcal{R}} \Gamma \to \Delta \quad \text{iff} \quad \vdash_{\mathsf{D}} \Gamma \to \Delta$$

Proof. Clearly, Cut is the same inference rule as the one in Proposition 2.1. □

The rule used in Proposition 2.2, 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 the first place, it is simple and easy to use for designing grammars. Context-free grammars are theoretically well understood and have pleasant computational properties as we will see in Chapters 3 and 4. Secondly, every natural language is to a large extent (though not entirely) context-free (CF for short), in the sense that it can be analyzed with the formalism of 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 2.12. A *context-free grammar* G is a quadruple $\langle \mathsf{V_t}, \mathsf{S}, \mathsf{Lex}, \mathsf{D} \rangle$ where

- $\mathsf{D} = \langle \mathcal{F}, \mathsf{AX}, \mathsf{R} \rangle$ is a deductive system[2],
- $\mathcal{F}^* \times \mathcal{F} \subseteq \mathsf{AX}$ are the productions of the grammar, which we write $\Gamma \to \mathsf{A}$, with $\Gamma \in \mathcal{F}^*$and $\mathsf{A} \in \mathcal{F}$
- R consists of the *Cut rule*:

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

- S is a distinguished formula, the start symbol,
- $\mathsf{V_t}$ is the terminal vocabulary. We assume that $\mathsf{V_t} \cap \mathcal{F} = \emptyset$,
- $\mathsf{Lex} = \{ w \to \mathsf{A} \mid w \in (\mathsf{V_t} \cup \{\epsilon\}),\ \mathsf{A} \in \mathcal{F} \}$.

Remark 2.1.

The cut rule has a global status. This means that the elements of Lex can be plugged into the leaves of the deduction of D, see Definition 2.20 on page 16.

Although in the previous definition, we distinguished between lexical axioms in Lex and axioms of the deductive system in AX, these two sets are usually merged by adopting a set of axioms, called *productions*, of the form

$$\Gamma \to \mathsf{A}, \text{ with } \Gamma \in (\mathcal{F} \cup \mathsf{V_t})^* \text{ and } \mathsf{A} \in \mathcal{F}.$$

Thus for simplicity, from now on we will write a CF grammar as quadruple $\langle \mathsf{V_t}, \mathsf{S}, \mathcal{F}, \mathcal{P} \rangle$, where $\mathcal{P} \subseteq (\mathcal{F} \cup \mathsf{V_t})^* \times \mathcal{F}$ is the set of the productions of the grammar.

Moreover, we are writing as $\Gamma \to \mathsf{A}$, what is usually written $\mathsf{A} \to \Gamma$ in the literature on phrase structure grammars. We preferred this notation because it allows a more uniform treatment of categorial and context-free grammars.

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

In the following examples of CF grammars, we write only the set of productions $\mathcal{P}$. Moreover, whenever several rewriting options $\Gamma_0 \to A, \ldots, \Gamma_n \to A$ appear in $\mathcal{P}$, we write them as $\Gamma_0 \mid \ldots \mid \Gamma_n \to A$ as it is customary.

Example 2.3. CF grammars:

- $G_0 = \{\, 0S1 \mid \epsilon \to S \,\}$.
- $G_1 = \{\, (S)S \mid \epsilon \to S \,\}$.
- $G_2 = \{\, 1S \mid 0 \to S \,\}$.
- $G_3 =$

$$\begin{array}{lcl}
\{\ \mathsf{NP}\ \mathsf{VP} & \to & \mathsf{S}, \\
\mathsf{Np} \mid \mathsf{Det}\ \mathsf{N} & \to & \mathsf{NP}, \\
\mathsf{IV} \mid \mathsf{TV}\ \mathsf{NP} & \to & \mathsf{VP}, \\
\textit{whistles} & \to & \mathsf{IV}, \\
\textit{loves} & \to & \mathsf{TV}, \\
\textit{John} \mid \textit{Mary} & \to & \mathsf{Np}, \\
\textit{every} \mid \textit{a} & \to & \mathsf{Det}, \\
\textit{man} \mid \textit{woman} & \to & \mathsf{N}\ \}.
\end{array}$$

We write $\vdash_G \Gamma \to A$, if the grammar G generates the sequent $\Gamma \to A$. We introduce some further notions below.

Definition 2.13.

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 $\vdash_G \Gamma \to S$.

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 take the terminal language generated to be the set

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

the grammar being understood.

Example 2.4. 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.

Let us give a deduction of the string (())() in grammar G_1.

Example 2.5. A deduction of the terminal string (())() in G_1.

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

Observe that several other deductions are available for the same terminal string. However, all of them are in a sense equivalent. To clearify in what sense, we can make use of the notion of *structural description*. A structural description is a tree whose internal nodes are labeled by non-terminals of the grammar and whose leaves are labeled by terminals. Such a tree indicates:

1. the hierarchical grouping of the parts of the expression into constituents,
2. the grammatical category of each constituent and
3. the left-to-right order of the constituents.

Let us define here a recursive procedure for mapping a deduction into a structural description.

Definition 2.14. Let a CF grammar $G = \langle V_t, S, \mathcal{F}, \mathcal{P} \rangle$ 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 $\Gamma \to C \equiv \delta_1 \dots \delta_m \to C \in \mathcal{P}$, then T is the tree

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

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

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

Assume that the tree

$$\dfrac{\overline{\delta}_1 \quad \dots \quad \overline{\delta}_m}{B}$$

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

$$\dfrac{\overline{\gamma}_1 \quad \dots \quad \overline{\gamma}_i \quad B \quad \overline{\gamma}_{i+1} \quad \dots \quad \overline{\gamma}_n}{C}$$

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

$$\dfrac{\overline{\gamma}_1 \quad \dots \quad \overline{\gamma}_i \qquad \dfrac{\overline{\delta}_1 \quad \dots \quad \overline{\delta}_m}{B} \qquad \overline{\gamma}_{i+1} \quad \dots \quad \overline{\gamma}_n}{C}$$

One can easily see that all the alternative deductions of the string (())() in grammar G_1 are mapped to the same structural description. Hence they are *equivalent*.

In fact, 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. Clearly, 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 keep these, because, for example, they express different *meanings* of a sequent. Take for instance the following grammar:

$$\begin{aligned} S \wedge S &\to S \\ \neg S &\to S \\ p_i &\to S, \quad i \leqslant 0 \end{aligned}$$

In this grammar, the expression $\neg p_1 \wedge p_2$ can be derived in different ways which are not all equivalent since they map to two different structural descriptions.

In Chapter 3, we will see that there are elegant and powerful methods to solve these problems for CF grammars. We will also see in Chapter 6 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 2.15. A grammar G is in Chomsky normal form (CNF), if it contains only productions of the form:

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

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

$\epsilon \to S$.

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 makes sense to maintain the partition of the axioms into *lexical* axioms, namely productions of the form $w \to A$, and *non-lexical* axioms, namely productions of the form $A\,B \to C$. Thus, we will write a Chomsky normal form CF grammar as $\langle V_t, S, \mathcal{F}, \mathsf{Lex}, AX \rangle$. All productions of CNF grammars are binary, and we will see in Chapters 3 and 4 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, we should remark also 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 formulation of linguistic categories as *basic* and *functor* categories appeared long before the advent of generative grammar. Ajdukiewicz [1935] was the first to apply the distinction between complete and incomplete expressions to the analysis of natural language. He was formalizing and applying to natural and artificial languages concepts from Leśniewski's mereology and Husserl's notion of semantic category, with mathematical tools coming from Russel's theory of types and of Frege's functional notation. A functor category was presented as an object of the form

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

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

2.4.1 AB grammars

Ajdukiewicz's notation was meant to provide a characterization of *semantic* well-formedness. In fact, he was aiming at the formalization of a notion of *semantic* category capable of encoding the function-argument structure of some formal and natural languages[3]. This notation was refined by Joshua Bar-Hillel, who distinguished categories of the form $\mathtt{a/b}$ and categories of the form $\mathtt{b\backslash a}$ in [Bar-Hillel, 1953]. The meaning of such a notation was the following.

- An expression of category $\mathtt{a/b}$ combines with an expression of category $\mathtt{b}$ to its right to give an expression of category $\mathtt{a}$.
- An expression of category $\mathtt{b\backslash a}$ combines with an expression of category $\mathtt{b}$ to its left to give an expression of category $\mathtt{a}$.

While this notation for linguistic categories preserves the functor-argument structure of Ajdukiewicz's original one, it also enables to distinguish between left-looking functors and right-looking functors, that is between functors selecting their arguments to the left and functors that select their arguments

[3]Such project was later developed into what is known as *model theoretic semantics*. For instance, [Montague, 1970] adopts a variant of Ajdukiewicz's notation for defining semantic well-formedness of natural language expressions.

to the right, respectively. Observe also that this definition of linguistic category merges the two fundamental domains of syntax and semantics: the way in which the expressions are composed and the way in which meanings are composed. We will see in more detail how this correspondence, also known as *compositionality* principle, is realized in categorial grammar in Section 2.6.

The calculus resulting by adopting the rules above, which we call *cancellation schemes*, is nowadays called AB calculus. More formally, we extend the formula type constructor as follows.

Definition 2.16. *Formulas*, or *categories*, are defined from the set of atoms A as

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

In the context of categorial grammar, CG hereafter, formulas are also called categories. Formulas of CF 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 CF grammar will always begin with capital letters. In showing complex formulas, we assume that the slashes have higher precedence over the product. Thus for example we write $\mathsf{a} \otimes \mathsf{b}\backslash\mathsf{c}$ for $\mathsf{a} \otimes (\mathsf{b}\backslash\mathsf{c})$. Finally, we assume that / is left associative and \ is right associative. So $(\mathsf{a}/\mathsf{b})/\mathsf{c}$ may be written as $\mathsf{a}/\mathsf{b}/\mathsf{c}$, and $\mathsf{c}\backslash(\mathsf{b}\backslash\mathsf{a})$ as $\mathsf{c}\backslash\mathsf{b}\backslash\mathsf{a}$.

Definition 2.17.

The *length* of a formula a, denoted $|\mathsf{a}|$, is the number of its atom occurrences.

The *order* of a formula a, denoted $\mathsf{o}(\mathsf{a})$ is defined as

$$\begin{array}{lcl} \mathsf{o}(\mathsf{a}) & = & 0, \text{ if } \mathsf{a} \text{ is an atom} \\ \mathsf{o}(\mathsf{a}/\mathsf{b}) & = & \mathsf{max}(\mathsf{o}(\mathsf{a}), \mathsf{o}(\mathsf{b}) + 1) \\ \mathsf{o}(\mathsf{b}\backslash\mathsf{a}) & = & \mathsf{max}(\mathsf{o}(\mathsf{a}), \mathsf{o}(\mathsf{b}) + 1) \\ \mathsf{o}(\mathsf{b} \otimes \mathsf{a}) & = & \mathsf{max}(\mathsf{o}(\mathsf{a}), \mathsf{o}(\mathsf{b})) \end{array}$$

Product formulas, that is formulas of the form $\mathsf{a}\otimes\mathsf{b}$, appear for the first time in [Lambek, 1958]. Still nowadays, many categorial linguists work with *product-free* systems. In some case, this may be a legitimate limitation. 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 in which complex expressions are derived according to the inference rules.

Definition 2.18. A *categorial grammar* based on a deductive system D is a quadruple $\langle \mathsf{V_t}, \mathsf{s}, \mathsf{Lex}, \mathsf{D} \rangle$ where

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

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

Definition 2.19. The AB deductive system is a triple $\langle \mathcal{F}, AX, R\rangle$ such that

- $\mathcal{F} \times \mathcal{F} \subseteq AX$ is a set of *identity* axioms, which we write $a \to a$.
- R consists of the following rules which we call *basic cancellation rules*.

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

We call AB grammar a categorial grammar based on the deductive system AB. As an example, consider the following AB grammar.

Example 2.6.

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

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 connection, 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 2.20. A categorial grammar $G = \langle V_t, s, Lex, D\rangle$ *generates* a string $w_0 \dots w_m \in V_t^*$ if and only if $\vdash_D a_0 \dots a_n \to s$, $m \leqslant n$ and $w_0 \dots w_m \Rightarrow^+ a_0 \dots a_n$ on the basis of the axioms in Lex.

In this definition, we used $\Rightarrow^+$ for the transitive closure of $\Rightarrow$. As an example, we prove that grammar A_0 generates the language $a^n b^n$.

Example 2.7. Grammar A_0 generates the language $a^n b^n$.

Suppose that $\vdash_{A_0} (s/c/s)^n\, s\, (c)^n \to s$ for $n \geqslant 0$, then

$$\frac{\dfrac{s/c/s \to s/c/s \quad (s/c/s)^n\, s\, (c)^n \to s}{(s/c/s)^{n+1}\, s\, (c)^n \to s/c} \qquad c \to c}{(s/c/s)^{n+1}\, s\, (c)^{n+1} \to s}$$

Then, on the basis of the axioms in Lex,

$$a^n b^n \Rightarrow^+ (s/c/s)^n\, (c)^n \Rightarrow (s/c/s)^n\, s\, (c)^n$$

In the original formulations of AB grammars, a second kind of cancellation rules 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 prefer to keep distinct the two systems and discuss them separately.

Definition 2.21. We call AAB ('associative' AB calculus) the deductive system $\langle \mathcal{F}, AX, R \rangle$ where $\mathcal{F}$ and AX are as in Definition 2.19, and R consists of the basic cancellation rules of the AB calculus and of the following inference rules, which we call *associative cancellation rules*[4].

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

An AAB grammar is a categorial grammar that has AAB as deductive engine. Consider the following associative Ajdukiewicz–Bar-Hillel categorial grammar. For simplicity, we simply list the lexical assignments.

Example 2.8.

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

John	$\to$	n
Mary	$\to$	n
someone	$\to$	$s/(n\backslash s)$
everyone	$\to$	$(s/n)\backslash s$
everyone	$\to$	$((n\backslash s)/n)\backslash(n\backslash s)$
loves	$\to$	$(n\backslash s)/n$
ismissing	$\to$	$n\backslash s$
ismissing	$\to$	$(s/(n\backslash s))\backslash s$

As formulas may become soon rather long, we introduce the following macros. In using these macros, we will try to find a compromise between clarity and shortness.

Definition 2.22. *Macros.*

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

We present some deductions in the associative Ajdukiewicz–Bar-Hillel calculus.

[4] In fact, one can generalize the cancellation rules for the associative setting in the following way. Let $0 \leqslant k$ (if $k \equiv 0$, then $b/_1 \ldots /_k c \equiv b$), then one rule is the following

$$\frac{\Gamma \to a/b \quad \Delta \to b/_1 \ldots /_k c}{\Gamma\Delta \to a/_1 \ldots /_k c}$$

the other is the symmetric dual.

Example 2.9. Deductions of *Someone loves everyone* in A_1.

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

2.
$$\dfrac{s/iv \to s/iv \qquad \dfrac{tv \to tv \quad tv\backslash iv \to tv\backslash iv}{tv,\ tv\backslash iv \to iv}}{s/iv,\ tv,\ tv\backslash iv \to s}$$

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, one of which is left branching and the other right branching) and there are several reasons to be interested in both of them. In Section 2.6, 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 2.9 express different scope relations between the subject and the object noun phrase[5].

2.4.2 Lambek style categorial grammars

The rule component of an $(\mathsf{A})\mathsf{AB}$ deductive systems consists of rules for building larger structures from simpler ones. Thus, for instance, 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 CF 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 axiom $A\,B \to C$ is among the productions of the grammar.

An important advancement in the field of categorial grammar was represented by the work of Joachim Lambek. In [Lambek, 1958], new rules were added to the composition rules of AB. 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.

[5] At the syntactic and prosodic levels, the two deductions in Example 2.9 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 scholars, see for example [Steedman, 2000b,a; Baldridge, 2002], emphasize the connection between the structures arising from (combinatory) categorial deductions and prosodic phrasing.

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

The product-free associative Lambek calculus, L for short, results from the AB calculus in Definition 2.19 by adding the slash introduction rules.

Definition 2.23. The product-free associative Lambek calculus, L, is a triple $\langle \mathcal{F}, AX, R \rangle$ where $\mathcal{F}$ and AX are as for the categorial grammars that we saw before and R consists of the following rules:

- Elimination rules:

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

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

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

Observe that in the introduction rules of L, Γ is assumed to be non-empty as in the original formulation of [Lambek, 1958] see also [Retoré, 2005]. We notice also that while AB was non-associative, L is *associative*. In fact, the absence of brackets in the antecedent of the sequents together with the slash introduction rules amounts to the same as having a fully associative regime.

An actual simplification, with respect to the system presented in [Lambek, 1958] is the absence of *product* rules, that is of rules dealing with formulas of the form $a \otimes b$. These rules will be introduced in the next section.

A product-free Lambek grammar based on L is a categorial grammar based on L. We show now the system L at work with a simple example.

Example 2.10.

Let A_2 be a L grammar whose lexicon consists of the following assignments:

John	$\to$	n
Mary	$\to$	n
someone	$\to$	$s/(n\backslash s)$
everyone	$\to$	$(s/n)\backslash s$
loves	$\to$	$(n\backslash s)/n$
book	$\to$	c
that	$\to$	$(c\backslash c)/(s/n)$
ismissing	$\to$	$(s/(n\backslash s))\backslash s$

We have the following deduction (among others). For reasons of space we use the macros in Definition 2.22 and we use commas in the antecedents for readability.

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

By means of lexical axioms we have:

$$\textit{book that someone loves} \Rightarrow^{+} c,\ (c\backslash c)/(s/n),\ s/iv,\ iv/n$$

The following laws can be proved in L.

Example 2.11. Characteristic theorems of L: for all formulas a, b and c,

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

Moreover, different kinds of *derived inference rules* can be adopted, and could replace those we used in defining L, as it is done in [Lambek, 1958] and [Zielonka, 1981] among others. As an example, we deduce one of the associative cancellation schemes of AAB.

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

2.4.3 Product categories

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], these calculi also included product formulas and rules to deal with them. $(A)AB$ calculi and CCG, instead, are customarily presented as product-free systems. The only formulation we know of an AB calculus with product is the one given in [Kandulski, 1988], whose rules we present below.

Definition 2.24. Inference rules of the $\mathsf{AB}^{\otimes}$ deductive system:

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

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

The system $\mathsf{AB}^{\otimes}$ consists of the basic cancellation rules of the system AB and of the *product rule*. We will make extensive use the system $\mathsf{AB}^{\otimes}$ and of its variants in Chapters 3 and 4. Indeed, we can see $\mathsf{AB}^{\otimes}$ as the link between the non-associative Lambek calculus which we are going to see and context-free grammars.

I introduce now the calculus of [Lambek, 1961]: the non-associative Lambek calculus with product, NL for short. This logic was introduced by Lambek to handle *bracketed strings of formulas*, that is to say to generate *trees* of formulas rather than *lists* of formulas[6]. In fact, a property of grammars based on the *syntactic calculus* of [Lambek, 1958] or on its product free variant is *structural completeness*: for any string in the language generated by such grammars all possible tree structures living on this string[7] are derivable, see [Buszkowski, 1997]. On the other hand, the assignment of structural descriptions to the expressions of a language is a primary concern of generative linguistics. In this respect, NL, equipped with product categories, has all the tools for generating appropriate structural descriptions for grammatical expressions.

We present the non-associative Lambek calculus.

Definition 2.25. *Pure logic of residuation*, NL.

- Identities:

Axioms

$$a \to a$$

Cut

$$\frac{a \to b \quad b \to c}{a \to c}$$

- Residuation rules:

$$\frac{a \otimes b \to c}{a \to c/b} \qquad \frac{a \to c/b}{a \otimes b \to c}$$

$$\frac{a \otimes b \to c}{b \to a\backslash c} \qquad \frac{b \to a\backslash c}{a \otimes b \to c}$$

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

[7] For a bracketed string Γ to *live on* a string $a_0 \dots a_n$ means that $a_0 \dots a_n$ is the result of eliminating all brackets from Γ.

Again a NL grammar is a categorial grammar based on NL. Let us define the notion of generation for such systems.

Definition 2.26. A NL grammar G generates a string $w_0 \dots w_m \in V_t^*$ if and only if NL generates $c \to s$, and

1. for some $n \geqslant m$, $a_0 \dots a_n \Rightarrow^* c$ by application of transitions of the form
$$a\, b \to a \otimes b$$
2. $w_0 \dots w_m \Rightarrow^+ a_0 \dots a_n$ by application of lexical axioms
$$w \to a_i \in Lex$$

We give here an example of a deduction in NL involving hypothetical reasoning.

Example 2.12. Consider the lexicon A_2 in Example 2.10.
We prove $n \otimes (s/(n\backslash s))\backslash s \to s$.

$$\dfrac{\dfrac{\dfrac{n\backslash s \to n\backslash s}{n \otimes n\backslash s \to s}}{n \to s/(n\backslash s)} \quad \dfrac{\dfrac{(s/(n\backslash s))\backslash s \to (s/(n\backslash s))\backslash s}{s/(n\backslash s) \otimes (s/(n\backslash s))\backslash s \to s}}{s/(n\backslash s) \to s/((s/(n\backslash s))\backslash s)}}{\dfrac{n \to s/((s/(n\backslash s))\backslash s)}{n \otimes (s/(n\backslash s))\backslash s \to s}}$$

Then, by means of one transition of the form $a\, b \to a \otimes b$ we have:

$$n\ (s/(n\backslash s))\backslash s \Rightarrow n \otimes (s/(n\backslash s))\backslash s$$

Finally, by means of lexical axioms

$$\textit{John ismissing} \Rightarrow^+ n\ (s/(n\backslash s))\backslash s$$

Although the previous example does not illustrate this aspect, we remark that in NL the product is not 'ornamental'[8]. It allows to distinguish the syntactic structures projected by heads of the form in A from those of the form in B below, which are instead (row by row) interderivable in the calculus of [Lambek, 1958].

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

For instance, one may prefer to assign to a multi-complement verb, say the ditransitive *gives*, a category like $(n\backslash s)/(n \otimes n)$, that projects a right branching structure, rather than the left branching category $(n\backslash s)/n/n$.

[8] As it is in the syntactic calculus due to structural completeness.

Algebraic semantics

NL is also called the *pure logic of residuation*. Let us explain why. If M is a non-empty set and $\circ$ is a binary operation on M, then the structure $(M, \circ)$ is called a *groupoid*. Furthermore, if $\leqslant$ is a partial ordering on M (that is a reflexive and transitive relation) and $\triangleleft$ and $\triangleright$ are binary operations on M fulfilling the equivalences

$$a \leqslant c \triangleleft b \quad \text{iff} \quad a \circ b \leqslant c \quad \text{iff} \quad b \leqslant a \triangleright c$$

then the structure $(M, \circ, \triangleleft, \triangleright, \leqslant)$ is called a *residuated* groupoid. A *model* is a pair $(\mathcal{M}, \nu)$, of a residuated groupoid $\mathcal{M}$ (with universe M) and an *interpretation* function ν which maps atomic formulas into elements of M. The function ν is extended to formulas by the following clauses:

$$\begin{aligned} \nu(a \otimes b) &= \nu(a) \circ \nu(b) \\ \nu(a/b) &= \nu(a) \triangleleft \nu(b) \\ \nu(b\backslash a) &= \nu(b) \triangleright \nu(a) \end{aligned}$$

A sequent $a \to c$ is said to be true in a model $(\mathcal{M}, \nu)$ if $\nu(a) \leqslant \nu(c)$. Sequents derivable in NL are precisely those sequents which are true in all residuated groupoid models as proved in [Szczerba, 1997], see also [Buszkowski, 1997, 2005].

Frame semantics

Formulas of NL can also be interpreted in *modal frames*. A modal frame F, is a pair $(W, \{R^3\})$, where W is a set possible worlds and R^3 is a ternary *accessibility* relation. We refer to [Blackburn et al., 2001] for an introduction to modal logic and to [Kurtonina, 1995; Moortgat, 1997; Buszkowski, 1997] for the application of modal logic to Lambek calculi. A *model* is a pair (F, ν), of a modal frame F and an *interpretation* function ν 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 $\Rightarrow$ as shorthand for *implies*).

Definition 2.27. Interpretation of formulas:

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

Completeness of NL with respect to modal frames is proved in [Došen, 1992].

Proposition 2.3. [Došen, 1992]

$\vdash_{NL} a \to c$ iff $\nu(a) \subseteq \nu(c)$ for every evaluation ν on every ternary frame.

[Kurtonina, 1995] and to [Kurtonina and Moortgat, 1997] generalize this completeness result to other logics resulting from NL by addition of some set of structural postulates (see section 2.8).

In the next sections, we will see another aspect of categorial semantics, one which is common to both CCG and Lambek style categorial grammars. We introduce the lambda calculus and examine the so called formula-as-type correspondence, also known has Curry-Howard correspondence, see [Howard, 1980].

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 2.28.

$$\begin{array}{lcl} Var & := & x_1 \mid x_2 \mid \ldots \\ Con & := & c_1 \mid c_2 \mid \ldots \mid c_n \\ Lam & := & Var \mid Con \mid \lambda Var.\ Lam \mid (Lam\ Lam) \\ & & \mid \langle Lam,\ Lam \rangle \mid \pi^1(Lam) \mid \pi^2(Lam) \end{array}$$

We show variables in Var as x, y or z, in which case different letters denote different variables[9]. A term $\lambda x.t$ is called an *abstract*, $(t_1\ t_2)$ is called *application*, $\langle t_1, t_2 \rangle$ *pair* and $\pi^1(t)$ and $\pi^2(t)$ 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, as far as it creates no ambiguity, we write $\pi^i(x)$ as $\pi^i x$, $i = 1$ or $i = 2$.

Definition 2.29.

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 it 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". Roughly, a term of the form $\lambda x.t$ is a *functor* that applied to an argument term v gives as result the term t with each occurrence of x replaced by the term v. For instance, $(\lambda x.(+\ x\ 3)\ 2) = (+\ 2\ 3)$.

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.

[9] Later, we will use variables x, y or z also for formulas. However it should always be clear what we are talking about.

Definition 2.30. *Substitution*: $t[x := t']$ is the result of substituting t' for x in t. Formally,

$$
\begin{array}{lcll}
x[x := v] & = & v & \\
x[y := v] & = & x & \\
c[y := v] & = & c & \text{if } c \in \mathsf{Con} \\
\langle t_1,\, t_2\rangle[y := v] & = & \langle t_1, [y := v],\ t_2[y := v]\rangle & \\
\pi^i(t)[y := v] & = & \pi^i(t[y := v]) & i = 1 \text{ or } i = 2 \\
(t_1\ t_2)[y := v] & = & (t_1[y := v]\ t_2[y := v]) & \\
(\lambda\, x.t)[x := v] & = & \lambda\, x.t & \\
(\lambda\, x.t)[y := v] & = & \lambda\, x.t & \text{if } y \notin \mathsf{FV}(t) \\
(\lambda\, x.t)[y := v] & = & \lambda\, x.(t[y := v]) & \text{if } y \in \mathsf{FV}(t) \text{ and } x \notin \mathsf{FV}(v) \\
(\lambda\, x.t)[y := v] & = & \lambda\, z.(t[x := z][y := v]) & \text{if } y \in \mathsf{FV}(t) \text{ and } x \in \mathsf{FV}(v)
\end{array}
$$

We 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 logic.

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 into 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 2.31.

Contraction: term equations for lambda terms

β-contraction		η-contraction
$\pi^i\langle t_1, t_2\rangle$	$= t_i,\ i = 1 \text{ or } i = 2$	$\langle \pi^1 t, \pi^2 t\rangle = t$
$((\lambda x.t)\ v)$	$= t[x := v]$	$\lambda x.(t\ x) = t$, if $x \notin \mathsf{FV}(t)$

A *reduction* is a series of contractions.

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

2.6 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. If a typed term t is functional, its type determines

the domain of its possible argument terms as well as the domain of the result of applying t to its arguments. We underline here informally the parallel with the categorial formalism: an expression of functional category, say $\frac{a}{b}$, combines with an expression of category b to give an expression of category a. In other words, syntactic categories are equivalence classes of linguistic expressions in the same way in which types are equivalence classes of lambda terms. We will see below that there is indeed a close correspondence between formulas of categorial grammar, types and typed terms that provides a strong ground for model theoretic semantics. This aspect of categorial logic has been deeply investigated in [van Benthem, 1991] and [Hendriks, 1993].

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

$$\mathsf{Type} := e \mid t \mid \mathsf{Type} \to \mathsf{Type} \mid \mathsf{Type} \times \mathsf{Type}$$

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 an object of type $p \to q$ as a function from objects of type p to objects of type q, while objects of type $p \times q$ are the Cartesian product of objects of type p and objects of type q.

Syntactic categories can be mapped into types in a straightforward way. Indeed, the type results from the category by ignoring the orientation of the slashes.

Definition 2.32. Let ty be a function from syntactic categories to types, $\mathsf{ty} :: \mathcal{F} \to \mathsf{Type}$. Let $\mathsf{ty}(x)$ be given for all the $x \in A$, for instance $\mathsf{ty}(n) = e$ and $\mathsf{ty}(s) = t$. Compound formulas are mapped to compound types as follows:

$$\begin{aligned} \mathsf{ty}(a/b) &= \mathsf{ty}(b) \to \mathsf{ty}(a) \\ \mathsf{ty}(b\backslash a) &= \mathsf{ty}(b) \to \mathsf{ty}(a) \\ \mathsf{ty}(b \otimes a) &= \mathsf{ty}(b) \times \mathsf{ty}(a) \end{aligned}$$

In [Moortgat, 1997] and [Moot, 2002], one can find a so called Church style definition of typed lambda terms in which each term constructor is assigned to a type. Instead, we are adopting 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 which can be assigned a type.

Below we present the basic rules for assigning types to lambda terms, when this is possible. The rules for typing lambda terms are deductive rules which operate on *type assignments*, objects of the form $t{:}p$, where t is a term and p is a type (variable), whose meaning is '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 $x_i{:}p_i$, $1 \leqslant i \leqslant n$, then t is a well formed term of type p'. In such judgemets, all free variables of t must be contained in $\{x_1, \ldots, x_n\}$.

Furthermore, we assume, without stating it explicitly in the rules in Definition 2.33, that each type judgement is *consistent*, that is no variable is assigned more than one type. The lack of consistency leads to failure of the type assignments. Here we use $\Gamma, \Gamma_i,\ 0 \leqslant i$ as variables over sets of type assigments.

Definition 2.33. Typing rules for typed lambda calculus:

$$\Gamma \cup \{x\colon p\} \vdash x\colon p$$

$$\frac{\Gamma_1 \vdash t_1\colon q \to p \quad \Gamma_2 \vdash t_2\colon q}{\Gamma_1 \cup \Gamma_2 \vdash (t_1\ t_2)\colon p} \qquad \frac{\Gamma \cup \{x\colon q\} \vdash t\colon p}{\Gamma \vdash \lambda x.t\colon q \to p}$$

$$\frac{\Gamma \vdash t\colon p \times q}{\Gamma \vdash \pi^1 t\colon p} \qquad \frac{\Gamma \vdash t\colon p \times q}{\Gamma \vdash \pi^2 t\colon q}$$

$$\frac{\Gamma_1 \vdash t_1\colon p \quad \Gamma_2 \vdash t_2\colon q}{\Gamma_1 \cup \Gamma_2 \vdash \langle t_1, t_2\rangle\colon p \times q}$$

To be precise, the typing rules operate on *type variables* p, q .... Thus the result of a type deduction is a *type schema* for the input lambda term. A more detailed 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]. In the first of the examples below, we directly unify the type of the term with the type resulting for the syntactic category.

Example 2.13. Typing terms.

1. The word $\mathtt{himself}$ is assigned the category $((n\backslash s)/n)\backslash n\backslash s$ and the term $\lambda x\lambda y.(x\ y\ y)$. We have $ty(((n\backslash s)/n)\backslash n\backslash s) = (e \to (e \to t)) \to (e \to t)$ and the following typing deduction.

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

2. We type the term $\lambda x\lambda y.x$:

$$\dfrac{\dfrac{\{x\colon p,\ y\colon q\} \vdash x\colon p}{\{x\colon p\} \vdash \lambda y.x\colon q \to p}}{\emptyset \vdash \lambda x\lambda y.x\colon p \to q \to p}$$

3. The term $\lambda x.(x\ x)$ cannot be typed. Assume that it were. Then it is a function $p \to r$, where p is the type of x and r is the type of $(x\ x)$. Then x should be of type $p \to r$, which gives an inconsistent type judgement.

2.7 Extended categorial grammars

By *extended deductive systems* we mean a deductive system extended with proof-encoding terms of some sort. If we are interested in compositional semantics, the term language will be the lambda term language. However, depending on the purposes, different term languages may be useful. For example, [Lambek, 1993] uses a term language isomorphic to deductions in the non-associative Lambek calculus and defines proof normalization via the term equations arising from the cut elimination algorithm. [Moortgat and Oehrle, 1997] adopt a similar term language.

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

Definition 2.34. An *extended deductive system* is a quadruple $\langle \mathcal{F}, \mathcal{T}, AX, R \rangle$ such that

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

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

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

The functional character of lambda calculus and categorial logics allows to interpret each inference rule as an operation on lambda terms. We use uppercase greek letters for *bracketed* strings of categories. 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 term is always an object of the form

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

such that 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. Thus, for instance, identity axioms are always of the form

$$\lambda x.x : a \to a$$

We define now the rules with semantic terms for the Ajdukiewicz–Bar-Hillel calculi that we saw before.

Definition 2.35. Rules with semantic annotation for Ajdukiewicz–Bar-Hillel calculi:

$$\frac{u\colon \Gamma \to a \quad v\colon \Delta \to b}{\lambda x.\langle (u\ \pi^1 x), (v\ \pi^2 x)\rangle\colon (\Gamma,\ \Delta) \to a \otimes b}$$

$$\frac{v\colon \Gamma \to a/b \quad u\colon \Delta \to b}{\lambda x.(v\ \pi^1 x\ (u\ \pi^2 x))\colon (\Gamma,\ \Delta) \to a}$$

$$\frac{u\colon \Gamma \to b \quad v\colon \Delta \to b\backslash a}{\lambda x.(v\ \pi^2 x\ (u\ \pi^1 x))\colon (\Gamma,\ \Delta) \to a}$$

$$\frac{v\colon \Gamma \to a/b \quad u\colon \Delta \to b/c}{\lambda x\lambda y.(v\ \pi^1 x\ (u\ \pi^2 x\ y))\colon (\Gamma,\ \Delta) \to a/c}$$

$$\frac{u\colon \Gamma \to c\backslash b \quad v\colon \Delta \to b\backslash a}{\lambda x\lambda y.(v\ \pi^2 x\ (u\ \pi^1 x\ y))\colon (\Gamma,\ \Delta) \to c\backslash a}$$

Observe that we added brackets to the antecedent of the conclusion of each rule. Their role will be explained in Definition 2.38 on page 31.

Let us define define the class of categorial lambda terms. These are the terms which can be constructed in categorial deductions. We say that $\pi^i x$ is free in t if it is non in the scope of λx.

Definition 2.36. *Categorial lambda terms.*

We say that $x \in Var$ is *linear in* the term t if and only if

1. x has exactly one free occurrence in t, or
2. x is of type $p \times q$ and there is exactly one free occurrence of $\pi^1 x$ and exactly one free occurrence of $\pi^2 x$ in t. Then, let u and v be two variables of type p and of type q, respectively, which do not occur in t. Let t' be the result of substituting u for the free occurrence of $\pi^1 x$ and v for the free occurrence of $\pi^2 x$ in t. Then x is *linear* in t, if u and v are linear in t'.

A lambda term t is *categorial* if and only if t contains no free variables and for every subterm $\lambda x.t'$ of t, x is linear in t'.

In fact, the constraints on categorial lambda terms apply only to the terms built in the categorial deductions. 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 2.36 is the assignment of the reflexive pronoun, *himself*, whose category is $((n\backslash s)/n)\backslash(n\backslash s)$ and whose term assignment is $\lambda x\lambda y.(x\ y\ y)$.

Figure 2.2 on page 33, contains several examples of lexical assignments that violate the constraints on categorial lambda terms.

We define extended categorial grammars.

Definition 2.37. An extended categorial grammar is a categorial grammar based on an extended deductive system D and whose lexicon Lex contains triples $\langle t, w, c\rangle$, which we write as $t : w \to c$, where $w \in (V_t \cup \{\epsilon\})$, $c \in \mathcal{F}$ 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 that for each lexical arrow the type of the lambda term is the same as that of the category to which it is associated. Thus, j and m are constants of type e, $\mathtt{loves}'$ is of type $e \to (e \to t)$, $\mathtt{ismissing}'$ is of type $e \to t$ and $\mathtt{ismissing}^*$ is of type $((e \to t) \to t) \to t$. $\exists$ and $\forall$ are constants of type $(e \to t) \to t$. An expression $\exists x\, \nu$ is shorthand for $(\exists\ \lambda x.\, \nu)$, which is the term resulting from application of the constant $\exists$ to the term $\lambda x.\, \nu$ of type $e \to t$[10]. The same holds for $\forall x\, \nu$. 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 2.14. Figure 2.1 presents lexicon A_1 with semantic annotation:

j :	*John*	$\to$	n
m :	*Mary*	$\to$	n
$\lambda x.\exists y\,(x\ y)$:	*someone*	$\to$	$s/(n\backslash s)$
$\lambda x.\forall y\,(x\ y)$:	*everyone*	$\to$	$(s/n)\backslash s$
$\lambda x\lambda y.\forall z\,((x\,z)\ y)$:	*everyone*	$\to$	$((n\backslash s)/n)\backslash(n\backslash s)$
$\mathtt{loves}'$:	*loves*	$\to$	$(n\backslash s)/n$
$\mathtt{ismissing}'$:	*ismissing*	$\to$	$n\backslash s$
$\mathtt{ismissing}^*$:	*ismissing*	$\to$	$(s/(n\backslash s))\backslash s$

Figure 2.1: A categorial lexicon with semantic annotation.

The notion of generation is extended to the systems enhanced with lambda terms in order to assign a semantic representation to the strings generated.

[10] 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.

Definition 2.38. An Ajdukiewicz–Bar-Hillel categorial grammar with semantic terms $\mathsf{G} = \langle \mathsf{V_t}, \mathsf{s}, \mathsf{Lex}, \mathsf{D} \rangle$ generates a string $w_0 \dots w_m \in \mathsf{V_t^*}$ and assigns it the semantic representation t if and only if D generates $\mathsf{f} \colon \Gamma \to \mathsf{s}$ and $\mathsf{t} = (\mathsf{f}\ \mathsf{t}')$, where

1. Γ is a structure living on $a_0 \dots a_n$, $m \leqslant n$,
2. $w_0 \dots w_m \Rightarrow^+ a_0 \dots a_n$ by means of lexical axioms
$$\mathsf{t}_i \colon w \to a_i \in \mathsf{Lex}$$
3. t' is obtained by replacing in Γ each a_i with t_i and each (with $\langle$ and each) with $\rangle$.

In this definition, the structural information encoded by Γ is exploited in 3 to build a lambda term t', which is a pair structure living on the terms $\mathsf{t}_0 \dots \mathsf{t}_n$ assigned, respectively, to $a_0 \dots a_n$ in the lexicon. The term t' encodes the *lexical semantics* of $w_0 \dots w_m$ according to G. The *derivational semantics* instead is encoded in the term f resulting from the deduction. The semantic representation t of $w_0 \dots w_m$ in G is given by the application of the derivational term f to the lexical term t', that is $\mathsf{t} = (\mathsf{f}\ \mathsf{t}')$.

The deductions given in Example 2.9 on page 18 are now proposed once more with lambda term decorations. For reasons of space we add the following macros to those in Definition 2.22.

Definition 2.39. *Macros.*

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

In the deductions below, we show the terms in normal form.

Example 2.15. Deductions with semantic annotation.

1. Deduction 1 of Example 2.9.
$$\dfrac{\dfrac{\lambda x.x \colon qs \to s/iv \quad \lambda y.y \colon tv \to iv/n}{\lambda x \lambda y.(\pi^1 x\ (\pi^2 x\ y)) \colon qs,\ tv \to s/n} \quad \lambda z.z \colon qo \to qo}{\lambda x.(\pi^2 x\ \lambda k.(\pi^1 \pi^1 x\ (\pi^2 \pi^1 x\ k))) \colon (qs,\ tv),\ qo \to s}$$
Lexical semantics: from the bracketed string of categories $(qs,\ tv)$, qo and A_1, we obtain the term $\langle \langle \lambda\, x.\exists y\, (x\ y), loves' \rangle, \lambda\, x.\forall y\, (x\ y) \rangle$. Thus we shall reduce the term
$$(\lambda x.(\pi^2 x\ \lambda k.(\pi^1 \pi^1 x\ (\pi^2 \pi^1 x\ k)))\ \langle \langle \lambda\, x.\exists y\, (x\ y), loves' \rangle, \lambda\, x.\forall y\, (x\ y) \rangle)$$
which gives the following semantic representation for *someone loves everyone*:
$$\forall z\, \exists y\, (loves'\ z\ y)$$

2. Deduction 2 of Example 2.9, instead, assigns the lambda term

$$\exists y\, \forall z\, (\mathsf{loves}'\, z\, y)$$

to the input string, as the reader can easily check.

We conclude this section by presenting NL with semantic annotation.

Definition 2.40. *Pure logic of residuation*, NL.

- Identities:

$$\text{Axioms} \qquad \lambda x.x\colon a \to a \qquad\qquad \text{Cut} \qquad \frac{\nu\colon a \to b \quad u\colon b \to c}{\lambda x.(u\ (\nu\ x))\colon a \to c}$$

- Residuation rules:

$$\frac{\nu\colon a \otimes b \to c}{\lambda x \lambda y.(\nu\ \langle x, y\rangle)\colon a \to c/b} \qquad \frac{\nu\colon a \to c/b}{\lambda x.(\nu\ \pi^1 x\ \pi^2 x)\colon a \otimes b \to c}$$

$$\frac{\nu\colon a \otimes b \to c}{\lambda x \lambda y.(\nu\ \langle y, x\rangle)\colon b \to a\backslash c} \qquad \frac{\nu\colon b \to a\backslash c}{\lambda x.(\nu\ \pi^2 x\ \pi^1 x)\colon a \otimes b \to c}$$

Figure 2.2 on the next page presents some examples of semantically annotated lexical entries for a NL lexicon. We added the atomic types i, for infinitive, and b, for base form. With such assignments we can generate sentences like *John seems to leave* and *it seems that John leaves* as truth conditionally equivalent (observe the vacuous abstraction in the sentential complement *seem*). Both reduce to the term

$$(\mathsf{seem}'\ (\mathsf{leave}'\ j))$$

We have also examples of multiple occurrences of the same projection terms and of the same variable in the object controlled verb *persuaded* and in the subject controlled *promised* and *wants*, respectively. This gives, for instance

$$\begin{aligned} &\textit{John persuaded Mary to leave} \quad : \quad (\mathsf{persuade}'\ m\ (\mathsf{leave}'\ m)\ j)\\ &\textit{John promised Mary to leave} \quad : \quad (\mathsf{promise}'\ m\ (\mathsf{leave}'\ j)\ j) \end{aligned}$$

Consider the following two examples, from which we omit the derivational component, as easily recoverable from the given sequents. We also simplify the notation for the terms in the following way: we write $\pi^i_n x$ for the term $\pi^i_n \ldots \pi^i_1 x$.

Example 2.16.

John gave Mary a book.

j :	*John*	$\rightarrow$	n
m :	*Mary*	$\rightarrow$	n
$\lambda x.(x\ y)$:	*it*	$\rightarrow$	$\mathsf{s}/(\mathsf{n}\backslash\mathsf{s})$
give' :	*gave*	$\rightarrow$	$(\mathsf{n}\backslash\mathsf{s})/(\mathsf{n}\otimes\mathsf{n})$
$\lambda x\lambda y.(\mathsf{seem}'\ x)$:	*seems*	$\rightarrow$	$(\mathsf{n}\backslash\mathsf{s})/\mathsf{s}$
seem' :	*seems*	$\rightarrow$	$(\mathsf{n}\backslash\mathsf{s})/\mathsf{i}$
$\lambda x.(\mathsf{persuade}'\ \pi^1 x\ (\pi^2 x\ \pi^1 x))$:	*persuaded*	$\rightarrow$	$(\mathsf{n}\backslash\mathsf{s})/(\mathsf{n}\otimes\mathsf{i})$
$\lambda x\lambda y.(\mathsf{promise}'\ \pi^1 x\ (\pi^2 x\ y)\ y)$:	*promised*	$\rightarrow$	$(\mathsf{n}\backslash\mathsf{s})/(\mathsf{n}\otimes\mathsf{i})$
$\lambda x\lambda y.(\mathsf{want}'\ (x\ y)\ y)$:	*wants*	$\rightarrow$	$(\mathsf{n}\backslash\mathsf{s})/\mathsf{i}$
praise' :	*praises*	$\rightarrow$	$(\mathsf{n}\backslash\mathsf{s})/\mathsf{n}$
leave' :	*leave*	$\rightarrow$	b
leave' :	*leaves*	$\rightarrow$	$\mathsf{n}\backslash\mathsf{s}$
$\mathsf{ismissing}'$:	*ismissing*	$\rightarrow$	$(\mathsf{s}/(\mathsf{n}\backslash\mathsf{s}))\backslash\mathsf{s}$
a' :	*a*	$\rightarrow$	n/c
book' :	*book*	$\rightarrow$	c
$\lambda x.x$:	*that*	$\rightarrow$	s/s
$\lambda x.x$:	*to*	$\rightarrow$	i/b
$\lambda x\lambda y.\langle y, x\rangle$:	*to*	$\rightarrow$	$(\mathsf{n}\backslash(\mathsf{n}\otimes\mathsf{n}))/\mathsf{n}$
$\lambda x\lambda y.(x\ y\ y)$:	*himself*	$\rightarrow$	$((\mathsf{n}\backslash\mathsf{s})/\mathsf{n})\backslash(\mathsf{n}\backslash\mathsf{s})$

Figure 2.2: A categorial lexicon with semantic annotation.

1. $\mathsf{n}\otimes(\mathsf{iv}/(\mathsf{n}\otimes\mathsf{n})\otimes(\mathsf{n}\otimes(\mathsf{n}/\mathsf{c}\otimes\mathsf{c})))\rightarrow\mathsf{s}$
2. Derivational semantics:
$$\lambda x.(\pi^1\pi^2 x\ \langle\pi^1\pi_2^2 x, (\pi^1\pi_3^2 x\ \pi_4^2 x)\rangle\ \pi^1 x)$$
3. Lexical semantics:
$$\langle A, \langle \mathsf{give}', \langle R, \langle \mathsf{a}', \mathsf{book}'\rangle\rangle\rangle\rangle$$
4. Meaning representation:
$$((\mathsf{give}'\ \langle \mathsf{m}, (\mathsf{a}'\ \mathsf{book}')\rangle)\ \mathsf{j})$$

John gave a book to Mary.

1. $\mathfrak{n} \otimes (\mathfrak{iv}/(\mathfrak{n} \otimes \mathfrak{n}) \otimes ((\mathfrak{n}/\mathfrak{c} \otimes \mathfrak{c}) \otimes ((\mathfrak{n}\backslash(\mathfrak{n} \otimes \mathfrak{n}))/\mathfrak{n} \otimes \mathfrak{n}))) \to \mathfrak{s}$
2. Derivational semantics:
$$\lambda x.(\pi^1\pi^2 x\ (\pi^1\pi^2_3 x\ \pi^2_4 x\ (\pi^1_2\pi^2_2 x\ \pi^2\pi^1\pi^2_2 x))\ \pi^1 x)$$
3. Lexical semantics:
$$\langle \mathfrak{j}, \langle \mathfrak{give}', \langle\langle \mathfrak{a}', \mathfrak{book}'\rangle, \langle \lambda x\lambda y.\langle y, x\rangle, \mathfrak{m}\rangle\rangle\rangle\rangle$$
4. Meaning representation:
$$((\mathfrak{give}'\ \langle \mathfrak{m}, (\mathfrak{a}'\ \mathfrak{book}')\rangle)\ \mathfrak{j})$$

We observe that both syntactic structures are right branching, in accordance with the intuitions. Furthermore, both constructions obtain the same semantic representation. 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. We remark that while syntactically inadequate, the assignment $(\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n}/\mathfrak{n}$ for the ditransitive verb should be associated to two different semantic translations to achieve the same effect.

From NL, we obtain the calculus of [Lambek, 1958], the so called syntactic calculus, by adding the labeled variant of the structural rules of associativity:

$$\lambda x.\langle\langle \pi^1 x, \pi^1\pi^2 x\rangle, \pi^2\pi^2 x\rangle : \mathfrak{a} \otimes (\mathfrak{b} \otimes \mathfrak{c}) \to (\mathfrak{a} \otimes \mathfrak{b}) \otimes \mathfrak{c}$$
$$\lambda x.\langle \pi^1\pi^1 x, \langle \pi^2\pi^1 x, \pi^2 x\rangle\rangle : (\mathfrak{a} \otimes \mathfrak{b}) \otimes \mathfrak{c} \to \mathfrak{a} \otimes (\mathfrak{b} \otimes \mathfrak{c})$$

However, the extension of a logical system by means of a set of structural rules is a general process in the type-logical setting and we dedicate the next section to this aspect of categorial logics.

2.8 Multi-modal type-logical grammars

Multi-modal type-logical grammars are a generalization of categorial grammars. The rules of *associativity* of the *syntactic calculus* of [Lambek, 1958], which we assumed implicitly in our formulation of L, have a somewhat special status: they are *structural rules*. As we mentioned before, the syntactic calculus results from NL by adding the structural rules of associativity. More in general, the framework of multi-modal type-logical grammar assumes that NL is the basic deductive engine (the *pure logic of residuation*) and that other logics are defined by adding to NL a package of structural rules. Thus, a type-logical system is a pair $\langle \mathsf{Q}, \mathsf{NL}\rangle$, where Q is a set of structural postulates. Let us give some examples of structural postulates and of the logics that we can define. We refer the reader to [Došen, 1988] for a larger set of postulates (though without term labeling).

Example 2.17.

Associativity, A:

$$A_1 \quad \lambda x.\langle\langle \pi^1 x, \pi^1\pi^2 x\rangle, \pi^2\pi^2 x\rangle : a \otimes (b \otimes c) \to (a \otimes b) \otimes c$$

$$A_2 \quad \lambda x.\langle \pi^1\pi^1 x, \langle \pi^2\pi^1 x, \pi^2 x\rangle\rangle : (a \otimes b) \otimes c \to a \otimes (b \otimes c)$$

Permutation, P:

$$\lambda x.\langle \pi^2 x, \pi^1 x\rangle : b \otimes a \to a \otimes b$$

Contraction, C:

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

Weakening, W:

$$\begin{array}{cc} W_1 & W_2 \\ \lambda x.\pi^1 x : a \otimes b \to a & \lambda x.\pi^2 x : b \otimes a \to a \end{array}$$

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

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

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

Lambek *syntactic calculus* $\mathsf{S} = \langle\{A\}, \mathsf{NL}\rangle$, formulated in [Lambek, 1958].

Taken in isolation, 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[11]. The natural candidates for natural language grammar seem to lay in between LP and NL. However, while the Lambek-van Benthem calculus is too permissive, as it admits any permutation of the input string, NL is too restrictive as it generates only context-free languages as proved in [Kandulski, 1988].

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, S 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.

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

The multi-modal setting

The *multi-modal* setting of Moortgat [1996, 1997] solves these problems in the following way. The notion of syntactic category is generalized to n-ary type constructors ($n \geqslant 1$) and distinguished *composition modes*.

Definition 2.41. Multi-modal formulas:

$$\mathcal{F} := A \mid /_i^n(\mathcal{F}_1, \ldots, \mathcal{F}_n) \mid \backslash_i^n(\mathcal{F}_1, \ldots, \mathcal{F}_n) \mid \otimes_i^n(\mathcal{F}_1, \ldots, \mathcal{F}_n)$$

In this definition, n is a positive integer expressing the arity of the connective and i an index, the so called *composition mode* distinguishing for example $\otimes_j^n$ from $\otimes_k^n$. In previous systems, $n = 2$ and there was only one composition mode. In [Buszkowski, 2005], one can find a Gentzen style sequent calculus formulation of the generalized Lambek calculus with n-ary type forming operators. In linguistic applications, one usually works with $n \leqslant 2$. Every distinguished connective obeys the laws of the pure logic of residuation. However, by specifying the *modes* appearing in the structural rules, one specifies also the syntactic configurations which 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 grammar 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 2.18. Mixed postulates:

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

We 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. Vermaat uses the postulates of [Moortgat, 1999], which in turn are variants of the mixed postulates in Example 2.18. The main difference consists in the use of *unary operators* to mark the substructure subject to displacement.

The unary operators have a special role in the multi-modal setting. They are the diamond $\Diamond_i = \otimes_i^1$ and its residual, the box $\Box_i$.

Definition 2.42. Unary residuation rules:

$$\frac{\Diamond_i a \to c}{a \to \Box_i c} \qquad \frac{a \to \Box_i c}{\Diamond_i a \to c}$$

The interpretation of unary formulas is given by extending the modal frame F which we described just before Definition 2.27 on page 23, to a pair $(W, \{\mathsf{R}^2, \mathsf{R}^3\})$, where R^2 a binary relation. The model-theoretic interpretation ν is extended to unary formulas in the straightforward way:

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

These operators have found a wide range of applications 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 2.19. Displacement postulates, from [Moortgat, 1999][12]:

$$\begin{aligned} \Diamond \mathfrak{a} \otimes (\mathfrak{b} \otimes \mathfrak{c}) &\to (\Diamond \mathfrak{a} \otimes \mathfrak{b}) \otimes \mathfrak{c} \\ (\mathfrak{a} \otimes \mathfrak{b}) \otimes \Diamond \mathfrak{c} &\to \mathfrak{a} \otimes (\mathfrak{b} \otimes \Diamond \mathfrak{c}) \\ \Diamond \mathfrak{a} \otimes (\mathfrak{b} \otimes \mathfrak{c}) &\to \mathfrak{b} \otimes (\Diamond \mathfrak{a} \otimes \mathfrak{c}) \\ (\mathfrak{a} \otimes \mathfrak{b}) \otimes \Diamond \mathfrak{c} &\to (\mathfrak{a} \otimes \Diamond \mathfrak{c}) \otimes \mathfrak{b} \end{aligned}$$

We observe that only the specific ternary configurations in which the diamond marker appears are subject to these forms of restructuring. The occurrence of diamond decorations in a deduction, in turn, is ultimately triggered by some lexical item. Hence *restructuring* is always *local* and lexically driven and controlled. We should also mention 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 in the multi-modal setting. [Heylen, 1999] develops a feature theoretic syntax relying on 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 employed 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: then unary distribution postulates enforce clitic climbing and 'attachment' of the clitic to the verb. Finally, in [Hendriks, 1999], the diamond is used as a prosodic selector.

We remark that the technique of multi-modal control has been imported also in CCG. For instance, in [Baldridge, 2002; Kruijff and Baldridge, 2003], different forms of multi-modal control have been adopted and successfully applied to the analysis of several syntactic and prosodic phenomena.

[12] Unary operators are assumed to have higher precedence over binary operators. Thus $\Diamond \mathfrak{a} \otimes \mathfrak{b}$ is interpreted as $(\Diamond \mathfrak{a}) \otimes \mathfrak{b}$.

Extending the logical vocabulary

The type-logical approach is currently being developed in the direction of the further enrichment of the logical vocabulary, see [Bernardi and Moortgat, 2007] and [Moortgat, 2007]. Categorial formulas are defined as follows.

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

The connectives $\oslash$ and $\oslash$ just introduced are called right and left co-slashes, respectively, and $\oplus$ co-product. The deductive system operating on these formulas is a refinement of a system originally presented in [Grishin, 1983]. To the rules of NL, we add the following rules:

$$\frac{a \oslash c \to b}{c \to a \oplus b} \qquad \frac{c \oslash b \to a}{c \to a \oplus b}$$

$$\frac{c \to a \oplus b}{a \oslash c \to b} \qquad \frac{c \to a \oplus b}{c \oslash b \to a}$$

We call the resulting system NG. These rules are symmetric to those of NL with respect to the arrow symbol. Thus, for instance, while in NL we has lifting

$$a \to c/(a\backslash c)$$

in NG one can prove also its symmetric

$$c \oslash (a \oslash c) \to a$$

Indeed, for every theorem of NL, also the symmetric dual holds in NG. More interesting are however the theorems arising by adding to NG the following *interaction* postulates.

Definition 2.43. Interaction principles:

$$\textit{mal} \quad (a \oslash b) \otimes c \to a \oslash (b \otimes c)$$

$$\textit{mcl} \quad a \otimes (b \oslash c) \to b \oslash (a \otimes c)$$

$$\textit{mar} \quad a \otimes (b \oslash c) \to (a \otimes b) \oslash c$$

$$\textit{mcr} \quad (a \oslash b) \otimes c \to (a \otimes c) \oslash b$$

These postulates are variants of the displacement postulates which we saw before.

2.9 Generative power of categorial grammars

I conclude this chapter by recalling some results about the generative power of categorial grammars. For a thorough discussion of this topic, I refer the reader

to [Buszkowski, 1997] and to the works he draws upon.

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

The rules of the Ajdukiewicz–Bar-Hillel systems that we saw in this chapter (apart from the product rule) are a subset of the rule component of *combinatory categorial grammar*, see [Ades and Steedman, 1982; Steedman, 2000b]. The generative power of combinatory categorial grammar depends on the set of rules which is adopted. While the Ajdukiewicz–Bar-Hillel grammars that we saw are all context-free, the addition of other rules, as the so called mixed Geach rules

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

increases the generative power to *mildly context-sensitive*. We refer to [Weir and Joshi, 1988; Vijay-Shanker and Weir, 1994] for a proof and to [Steedman, 2000b] for discussions and linguistic applications of combinatory categorial grammar.

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 grammars based on the *syntactic calculus*, which has been an open problem for more than thirty years, was given in [Pentus, 1993].

With regard to the strong generative power of Lambek style categorial grammars, we refer to [Tiede, 1998, 1999b] for an alternative notion of generated structure, see also [Le Nir, 2003a]. In these works, the structures generated are assumed to be the proof trees themselves. It turns out that the proof trees in the product-free non-associative Lambek calculus are regular trees, hence context-free, while in the case of the product-free associative Lambek calculus they are non-regular, hence richer than context-free.

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 if the structural rules duplicate or erase formulas, multi-modal type-logical grammars are Turing-complete. In fact, [Moot, 2002] shows in Section 9.3 that even with *linear* structural rules for the binary operators (that is no contraction nor weakening), the multi-modal system may be undecidable since the structural rules for unary operators may be used to "encode the recognition problem for type 0 languages". However, Moot proves also that if all its postulates are *non-expanding* every multi-modal grammar is equivalent to some context-sensitive grammar.

An interesting subject for research is that of the generative power of a type-logical grammar adopting a more restricted set of structural postulates as, for

instance, the mixed postulates of [Moortgat, 1996, 1999] which we saw before. Clearly, the mixed postulates can derive modalized variants of the mixed Geach rules, as shown in [Moortgat, 1997], [Vermaat, 2005] and [Baldridge, 2002]. However, a number of restrictions can be imposed to the postulates and to their application. For instance, one may enforce some forms of asymmetries, as unidirectional application of structural postulates, or the constraint that for each application of a structural postulate, there should be a 'mirror' application of the symmetric postulate. Such restrictions may also reduce the computational complexity of multi-modal type-logical grammars (an issue that I will discuss at the end of Chapter 3).

2.10 Conclusion

In this chapter, I have introduced some of the notions and of the formal systems that we will use throughout this book. We will continue to study Ajdukiewicz–Bar-Hillel categorial grammars in Chapters 3 and 4, and I will show how these systems can be regimented for automatic theorem proving. The second part of the book is instead about NL and we will see how we can use NL grammars for normal form parsing.

Part I

Automated Reasoning

Chapter 3

Deductive Parsers

This chapter presents the CF and $\mathsf{AB}^{\otimes}$ systems as *parsing systems*, that is to say as deductive systems in which the construction of a deduction takes advantage of the linear order of the syntactic categories involved. My contribution consists in the application of the well known CYK and Earley parsing systems to $\mathsf{AB}^{\otimes}$ grammars and in the formulation of an original parsing system specific to $\mathsf{AB}^{\otimes}$ grammar.

As $\mathsf{AB}^{\otimes}$ grammars are equivalent to CF grammars, it is not too surprising that the standard parsing methods for CF grammars can be applied also to $\mathsf{AB}^{\otimes}$ grammars. In fact, there is a straightforward way of translating an $\mathsf{AB}^{\otimes}$ grammar into an equivalent CF grammar which allows to apply CF parsing methods to the resulting grammar. This method is used for instance in [Finkel and Tellier, 1996]. However, the result of such a translation is a CF grammar which is much bigger than the original $\mathsf{AB}^{\otimes}$ grammar as we show in Example 3.3 on page 46. Indeed, this approach fails to take advantage of the *abstract* character of the inference rules of the $\mathsf{AB}^{\otimes}$ system and of their independence from any non-logical stipulation. As we saw in Chapter 2, in the case of CF grammars each inference step is determined by the productions of the grammar. In the case of $\mathsf{AB}^{\otimes}$ grammars, instead, the lexicon determines which axioms take part in the deduction, while the construction of the deduction relies only upon the abstract rules of the deductive system and is entirely independent of the grammar. As the size of the input grammar is relevant for the efficiency of these parsing methods, it is worthwhile to explicitly formulate parsing systems for $\mathsf{AB}^{\otimes}$ grammars which make direct use of the abstract inference schemes proper to the categorial system.

A formulation of the CYK deductive parser for CCG grammars can be found also in [Shieber et al., 1995]. However, the CYK deductive parser that I will present is designed for AB grammars *with* product. The Earley style parsing system which I design for $\mathsf{AB}^{\otimes}$ is entirely new[1]. Furthermore, I propose a new

[1] In fact, [Hepple, 1999] presents an Earley style parser for Lambek grammars. On the other hand, this parser uses a number of extra-logical methods, as index sharing and first-

parsing system for $\mathsf{AB}^{\otimes}$ grammars, which I call $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ as it results from the 'fusion' of CYK and Earley, incorporating all the best features of the CYK and Earley system for $\mathsf{AB}^{\otimes}$ grammars.

In the first part of the chapter, we will see that the CYK and Earley systems for $\mathsf{AB}^{\otimes}$ exhibit *complementary* features. The bottom-up approach of the CYK system allows a straightforward implementation of the cancellation rules. In this approach, however, the product rule should be constrained in order to generate only the products which are required for the success of the computation. With regard to the Earley approach, top-down *prediction* offers a simple and natural way of encoding the product rule. On the other hand, it renders the classical *reduction* schemes

$$\mathfrak{a}\ \mathfrak{a}\backslash\mathfrak{b} \to \mathfrak{b} \qquad \text{and} \qquad \mathfrak{b}/\mathfrak{a}\ \mathfrak{a} \to \mathfrak{b}$$

a sort of *expanding* patterns (see also Chapter 5): the succedent, which is shorter, is given in the premise and the antecedent, which is longer, is returned in the conclusion. Thus for the Earley system, we shall constrain the application of the slash prediction.

The complementarity of these features suggests the possibility of integrating the two approaches into a new one. This is, in fact, what I do in the final part of the chapter. Earley and CYK for $\mathsf{AB}^{\otimes}$ will be 'merged' into a new parsing system incorporating all and only their pleasant properties, namely two oriented variants of completion for the slashes, as in the CYK system, and a top-down prediction for the product, as in the Earley system.

Chapter 4 is dedicated to the implementation and complexity analysis of these parsing systems.

3.1 Problems

In 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 in respect to Example 2.5 on page 12 that several other deductions were available. Let us consider another example.

According to the purely declarative notion of derivation given in Definition 2.11 on page 9, there are six possible ways of obtaining the sentence *John whistles* in grammar G_3 from Example 2.3.

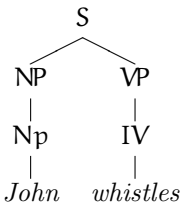
Example 3.1. Derivations of *John whistles* in grammar G_3

1. [*John whistles*, Np *whistles*, NP *whistles*, NP IV, NP VP, S]
2. [*John whistles*, Np *whistles*, Np IV, NP IV, NP VP, S]
3. [*John whistles*, *John* IV, Np IV, NP IV, NP VP, S]

order compilation, which render it rather complicated. Secondly, it is designed for a product-free system.

4. [*John whistles*, Np *whistles*, Np IV, Np VP, NP VP, S]
5. [*John whistles*, *John* IV, Np IV, Np VP, NP VP, S]
6. [*John whistles*, *John* IV, *John* 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 used the term spurious ambiguity for the situation arising when distinct derivations encode the same structural description. Clearly, if six different derivations are available for such a simple sentence, spurious ambiguity is a serious problem which may affect drastically the size of the search space.

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 will examine more sophisticated methods in the rest of the chapter.

One may observe that the redundancy in Example 3.1 is determined also by the shape of the productions of the grammar. We could, for instance, rephrase grammar G_3 into the equivalent grammar G_3' below.

Example 3.2. Grammar G_3':

$$\begin{aligned} \mathsf{NP\ VP} &\rightarrow \mathsf{S} \\ \mathit{John} \mid \mathit{Mary} \mid \mathsf{Det\ N} &\rightarrow \mathsf{NP} \\ \mathit{whistles} \mid \mathsf{TV\ NP} &\rightarrow \mathsf{VP} \\ \mathit{loves} &\rightarrow \mathsf{TV} \\ \mathit{every} \mid \mathit{a} &\rightarrow \mathsf{Det} \\ \mathit{man} \mid \mathit{woman} &\rightarrow \mathsf{N} \end{aligned}$$

Then, grammar G_3' only admits two equivalent derivations for the sentence *John whistles*.

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

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

A second observation has to do with the fact that in building a derivation for a sentence we can chose a *strategy* for applying the productions of the grammar. For example, for each rule $X_1 \dots X_n \to X$ we may chose to always expand first either the leftmost symbol X_1 or the rightmost symbol X_n. Such choice has the advantage of further reducing the number of possible derivations. For instance, continuing with the previous example, we have the following two cases.

Rightmost derivation:

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

Leftmost derivation:

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

While the choice of one of these recognition strategies may solve part of the problem of multiple redundant derivations, we can easily see that other problems may arise. For instance, a leftmost (resp. rightmost) reduction strategy may enter an infinite loop if the input grammar contains left (resp. right) recursive productions, of the form $Y X_1 \dots X_n \to Y$ (resp. $X_1 \dots X_n Y \to Y$). An example of right recursive rule in English is the following:

$$A\ N \to N$$

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 very elegant and simple ways for context-free grammars and that these solutions can be extended to $AB^{\otimes}$ grammars.

As we said at the beginning of the chapter, one could chose to translate an $AB^{\otimes}$ grammar into an equivalent CF grammar according to the procedure in [Bar-Hillel et al., 1964] (and extending it to the product case), and to apply the known algorithms to the resulting grammar, as [Finkel and Tellier, 1996] do. Consider the following example.

Example 3.3. Let G be a *first-order* $AB^{\otimes}$ grammar. The generation of the CF grammar G′ such that $L_t(G) = L_t(G')$ involves replacing each subformula x of formulas in the lexicon of G with a new atomic formula of G′, which we denote N_x. Thus G′ contains the lexical productions

$$\{\, w \to N_x \mid w \to x \in Lex \,\}$$

and moreover all the productions of the form

$$N_a\ N_b \to N_{a \otimes b}$$
$$N_a\ N_{a \backslash b} \to N_b$$
$$N_{b/a}\ N_a \to N_b$$

such that $a \otimes b$, $a\backslash b$ and b/a are subformulas of formulas in the lexicon of G.

Clearly, the number of productions of the grammar G' depends on the number of assignments in the lexicon of G and on the length of the formulas in the lexicon. Since each formula of length n has $2n-1$ subformulas, we can have an idea of how bigger G' can be. Furthermore, if G is not a first-order grammar, first-order conversion will also increase the size of the grammar. On the other hand, we know that the complexity of the CF parsing algorithms is affected by the size of the grammar. Thus one may suspect that better results could be obtained by working directly with $AB^{\otimes}$ grammars. We will see that this is indeed the case[2].

3.2 Deductive parsers

In Chapter 2, we presented context-free and categorial grammars as deductive systems. *Parsers* can also be seen as deductive systems . This is the perspective 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* (the kinds of objects that replace the sequents of the deductive systems seen in Chapter 2) encode, at least, the portion of the input string that has been recognized and its syntactic category. The recognized portion of the input is identified by two position indices. For instance, consider a context-free grammar G and a production of G of the form

$$A\ B \rightarrow C$$

Such a production can be read as asserting that if $w_{i+1} \dots w_k \Rightarrow^* A$ and $w_{k+1} \dots w_j \Rightarrow^* B$, then $w_{i+1} \dots w_k w_{k+1} \dots w_j \Rightarrow^* C$. In the deductive parsing formalism (or in the definite clause grammar formalism, see [Pereira and Shieber, 1987]) this can be expressed as

$$\forall i, j\, \exists k, i < k < j \text{ such that } (i, A, k)\ (k, B, j) \rightarrow (i, C, j)$$

A grammatical derivation succeeds if the entire input is analyzed as being a sentence, that is if $w_1 \dots w_n \Rightarrow^* S$. In turn, a parsing deduction succeeds if the item (0,S,n) can be generated by the parsing system.

The deductive perspective on parsing plays an important role also in the *parsing schemata* theory of 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.

We introduce here the basic notions which we will develop in the following sections.

[2] In fact we will see in the Chapter 4, that while for CF grammars we have a complexity of $O(|G|n^3)$, where $|G|$ is the size of the grammar, for $AB^{\otimes}$ grammars we have $O(n^3)$. As G is often much bigger than n, this is an important improvement.

Definition 3.1.

A *parsing system* $\mathcal{D}$ is a triple $\langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{I}$ is the domain of *items*, $\mathcal{A}$ is the set of axioms of $\mathcal{D}$ and $\mathcal{R}$ is a set of inference rules whose premises and conclusion are 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 of η in $\mathcal{D}$.

In the following sections, we will see the most famous and efficient general parsing systems for context-free grammars and extend them to $\mathsf{AB}^{\otimes}$ grammars. As we said before, these are the CYK parser and the Earley 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 CF grammars. Later [Younger, 1967] showed that this algorithm recognizes a string of length n in $O(n^3)$ operations. A similar algorithm had been proposed independently in [Kasami, 1965].

Let us consider the deductive formulation of the CYK parser for CF grammars. The system $\mathcal{CF}_{\mathsf{CYK}}$, in its simplest form, works with Chomsky normal form CF grammars without ϵ-productions. We represent such a grammar as $G = \langle V_t, S, \mathcal{F}, \mathsf{Lex}, \mathcal{P} \rangle$, where $\mathsf{Lex} \subseteq V_t \times \mathcal{F}$ is the lexicon of the grammar and $\mathcal{P} \subseteq (\mathcal{F}\mathcal{F}) \times \mathcal{F}$ is the set of its binary productions. The system can easily be generalized to grammars which are not in CNF.

Items are triples (i, A, j) such that $A \in \mathcal{F}$ and $0 \leqslant i < j \leqslant n$, where n is the length of the input string. Such an item has to be interpreted as asserting that $w_{i+1} \dots w_j \Rightarrow^* A$ in the reference grammar.

Definition 3.2. Let a CNF CF grammar $G = \langle V_t, S, \mathcal{F}, \mathsf{Lex}, \mathcal{P} \rangle$ and a string $w_1 \dots w_n$ be given. The parsing system $\mathcal{CF}_{\mathsf{CYK}} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ is defined in Figure 3.1 on the next page.

As we said, the CYK system works *bottom-up*, that is it builds deductions 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 3.4. Decoding of the inference rule of $\mathcal{CF}_{\mathsf{CYK}}$.

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

Let us consider now an example deduction.

$$
\begin{array}{rcl}
\mathcal{I} & = & \{\, (i, A, j) \mid A \in \mathcal{F},\ 0 \leqslant i < j \leqslant n \,\} \\
\mathcal{A} & = & \{\, (i-1, A, i) \mid w_i \to A \in \mathsf{Lex} \,\} \\
\mathcal{R} & = & \left\{ \dfrac{(i, A, k) \quad (k, B, j)}{(i, C, j)} \text{ if } A\,B \to C \in \mathcal{P} \right.
\end{array}
$$

Figure 3.1: The parsing system $\mathcal{CF}_{\mathsf{CYK}}$.

Example 3.5. Deduction in $\mathcal{CF}_{\mathsf{CYK}}$.

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

$$
\dfrac{\dfrac{\dfrac{\textit{John}}{(0, \mathsf{NP}, 1)} \quad \dfrac{\dfrac{\textit{loves}}{(1, \mathsf{TV}, 2)} \quad \dfrac{\dfrac{\textit{a}}{(2, \mathsf{Det}, 3)} \quad \dfrac{\textit{woman}}{(3, \mathsf{N}, 4)}}{(2, \mathsf{NP}, 4)}}{(1, \mathsf{VP}, 4)}}{(0, \mathsf{S}, 4)}
$$

Correctness of the system can be stated as follows.

Proposition 3.1. Let G be a CF grammar in CNF and without ϵ-productions. Then, the system $\mathcal{CF}_{\mathsf{CYK}}$ for G recognizes items (i, C, j) such that $\vdash_{\mathsf{G}} w_{i+1} \ldots w_j \to C$.

In particular, a string $w_0 \ldots w_n$ is grammatical if $(0, S, n)$ is the conclusion of a deduction in $\mathcal{CF}_{\mathsf{CYK}}$. We omit the proof of this proposition because we will prove later correctness of the CYK system for $\mathsf{AB}^{\otimes}$ grammars and the two proofs are very similar.

3.3.1 AB grammars

From $\mathcal{CF}_{\mathsf{CYK}}$ to the CYK parsing system for AB grammars it is a short step.

We define the parsing system $\mathcal{AB}_{\mathsf{CYK}}$. As in the case of $\mathcal{CF}_{\mathsf{CYK}}$, the lexicon of the input grammar should be free from axioms of the form $\epsilon \to a$, which we call ϵ-assignments, but as the rules of the AB system are only binary, no other restriction is imposed.

Definition 3.3. Let an AB grammar $\mathsf{G} = \langle V_t, S, \mathsf{Lex}, \mathsf{AB} \rangle$ without ϵ-assignments and a string $w_1 \ldots w_n$ be given. The parsing system $\mathcal{AB}_{\mathsf{CYK}} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ is defined in Figure 3.2 on the following page.

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

Figure 3.2: The parsing system $\mathcal{AB}_{\mathsf{CYK}}$.

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

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

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

[Shieber et al., 1995] presents a parsing system for CCG, which is an extension of $\mathcal{AAB}_{\mathsf{CYK}}$.

3.3.2 Product rules

In the previous section, we considered *product-free* Ajdukiewicz–Bar-Hillel categorial system. However, it is possible to extend the CYK deductive systems to the AB calculus with product, $\mathsf{AB}^{\otimes}$. 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 $\mathcal{AB}_{\mathsf{CYK}}$ and produce a correct parsing system $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$ for grammars based on $\mathsf{AB}^{\otimes}$.

(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 and therefore it generates all possible product formulas having as immediate subformulas the formulas occurring in the premises. For example, given items (i, a, j), (j, b, k) and (k, c, l), we have both the following deductions

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

Thus if we adopt rule 3.1, the parsing system for $AB^{\otimes}$ grammars will generate an exponential number of items[3]. Clearly, only a small subset of all the product items that can be generated by rule 3.1 are in fact needed for the deduction. Hence, we shall 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* , 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 items are required: we are interested only in the product formulas that may have to be generated by rule 3.1. We define the following set of formulas.

Definition 3.4. We define two functions, $\delta^+,\ \delta^- :: \mathcal{F} \to \{\mathcal{F}\}$, returning the set of product subformulas required for the subformula test for the product parsing rule (we omit the symmetric cases).

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

We can now formulate the parsing system $\mathcal{AB}^{\otimes}_{CYK}$ for AB grammars with product.

Definition 3.5. Let an $AB^{\otimes}$ grammar $G = \langle V_t, s, Lex, AB^{\otimes} \rangle$ without ϵ-assignments and a string $w_1 \dots w_n$ be given.

[3]More precisely, for a string $w_1 \dots w_n$ such that each w_i is assigned the category a_i, and only a_i, in the lexicon, the string $w_1 \dots w_n$ can be assigned C_n categories by means of the product rule, where C_n is the Catalan number of n defined by the recurrence:

$$\begin{array}{lcl}
C_0 & = & 1 \\
C_{n+1} & = & C_0 C_n + C_1 C_{n-1} + \dots + C_{n-1} C_1 + C_n C_0
\end{array}$$

A set of formulas Σ is generated as follows:

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

In Figure 3.3, we define the parsing system $\mathcal{AB}^{\otimes}_{\mathsf{CYK}} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$.

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

Figure 3.3: The parsing system $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$.

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 before, without this restriction the $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$ parsing system would generate a number of items greater than the Catalan number of the length of the input string.

Let us look at an example of deduction in $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$.

Example 3.6. Deduction in $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$:

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

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

Input string: $\neg[p_1 \vee p_2]$.

We calculate $\Sigma = \{\mathtt{s} \otimes \mathtt{c}\}$.

$$\dfrac{\dfrac{\neg}{(0,\mathtt{s}/\mathtt{s},1)} \quad \dfrac{\dfrac{[}{(1,\mathtt{s}/(\mathtt{s}\otimes\mathtt{c}),2)} \quad \dfrac{\dfrac{\mathtt{p}_1}{(2,\mathtt{s},3)} \quad \dfrac{\dfrac{\vee}{(3,(\mathtt{s}\backslash\mathtt{s})/\mathtt{s},4)} \quad \dfrac{\mathtt{p}_2}{(4,\mathtt{s},5)}}{(3,\mathtt{s}\backslash\mathtt{s},5)}}{(2,\mathtt{s},5)} \quad \dfrac{]}{(5,\mathtt{c},6)}}{(2,\mathtt{s}\otimes\mathtt{c},6)}}{(1,\mathtt{s},6)}}{(0,\mathtt{s},6)}$$

Below, we prove the correctness of $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$.

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

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 $\mathcal{D}$ 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 which is made possible by the deductive perspective on parsing.

Let us introduce some formal definitions.

Definition 3.6. Let $\mathcal{D} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ be a parsing system.

The set of *valid* items $\mathsf{V}(\mathcal{D})$ is the set of items which can be deduced in any number of steps from hypotheses in $\mathcal{A}$ together with, possibly, some further items[4].

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

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

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

[4]For instance, the initialization items used in top-down predictive parsing as we will see in Section 3.4.2.

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

We will use lowercase Greek letters as variables over items. The proof of soundness amounts to the proof the following statement.

Proposition 3.2. 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 $\mathsf{V}(\mathcal{D}) \subseteq \mathcal{W}$.

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

Definition 3.8. For an $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$ system, for an $\mathsf{AB}^{\otimes}$ grammar G and a string $w_1 \dots w_n$, we define the set of *viable* items $\mathcal{W}$ as follows

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

We proceed to prove the soundness of $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$.

Proposition 3.3. *Soundess* of $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$.
$\mathsf{V}(\mathcal{AB}^{\otimes}_{\mathsf{CYK}}) \subseteq \mathcal{W}$

Proof.

If $\xi \in \mathcal{A}$, then $\xi = (i - 1, a, i)$ and $w_i \to a \in \mathsf{Lex}$. Thus $\xi \in \mathcal{W}$.

If ξ is deduced by cancellation from items $(i, a/b, k)$ and (k, b, j), then $\xi = (i, a, j)$. 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 $\xi = (i, a \otimes b, j)$. 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, we follow [Sikkel, 1998] in defining a *deduction length function*, *dfl* for short, on the set of viable items $\mathcal{W}$. The *dfl* is a measure of the length of the deduction of an item based on the length of the corresponding grammatical derivation. For every item η generated by the parsing system, its *dfl* should be bigger than that of its premises. This allows to prove completeness by induction on the *dfl*. A similar notion is the *rank* used in [Aho and Ullman, 1972].

Definition 3.9. Deduction length function, *dfl*.

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

1. $\mathtt{d}(\eta) = 0$, if $\eta \in \mathcal{A}$
2. for each $\xi \in \mathcal{W}$ there is an inference rule in $\mathcal{D}$

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

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

Proposition 3.4. Let $\mathcal{D}$ be a parsing system and $\mathcal{W} \subseteq \mathcal{I}$.
If a *dfl* $\mathtt{d}$ exists, then $\mathcal{W} \subseteq \mathsf{V}(\mathcal{D})$.

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

Proposition 3.5. *Completeness* of $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$.
$\mathcal{W} \subseteq \mathsf{V}(\mathcal{AB}^{\otimes}_{\mathsf{CYK}})$

Proof. One defines a function $\mathtt{d}$ such that

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

Clearly, $\mathtt{d}$ is a deduction length function. □

In Section 4.3, I will give the tabular implementation of the CYK algorithm for CF and $\mathsf{AB}^{\otimes}$ grammars.

3.4 Earley style parsing

In the previous section, we examined the CYK deductive system for 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 some rule 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 CF grammars, we have to assume that the input grammar is in *Chomsky normal form*[5]. Secondly, the input grammar should not contain ϵ-productions.

The first limitation does not affect Ajdukiewicz–Bar-Hillel categorial grammars, as these grammars have only two-premise rules. Instead, the availability

[5]Although, as we said, there are also *generalized* variants of the CYK system.

of the empty string may simplify notably the formulation of the grammar in certain cases, both in the CF and in the categorial setting.

In this section, we discuss another kind of deductive parser which is not affected by the previous limitations. It is called *Earley parser*, from the name of its inventor Jay Earley. It works with any CF grammar, and it is faster than the CYK algorithm, at least if the underlying grammar is not ambiguous. The parser 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 symbol towards the leaves.

While the CYK parser had already been applied to various kinds of categorial calculi, most notably to CCG, our formulation of the Earley parser for $\mathsf{AB}^{\otimes}$ grammars is new.

3.4.1 Earley system for 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 CF grammar and in some case it is more efficient than the CYK parser.

Earley items are triples $(\mathfrak{i}, \Delta \bullet \Lambda \to \mathsf{A}, \mathfrak{j})$ such that $\Delta\Lambda \to \mathsf{A}$ is a production of the grammar, $0 \leqslant \mathfrak{i} \leqslant \mathfrak{j} \leqslant \mathfrak{n}$, where $\mathfrak{n}$ is the length of the input string, and $\bullet$ is a special symbol not occurring in the grammar. Such items are interpreted as asserting that $w_{\mathfrak{i}+1} \dots w_{\mathfrak{j}} \Rightarrow^* \Delta$, and $w_1 \dots w_{\mathfrak{i}}\, \mathsf{A}\, \Gamma \Rightarrow^* \mathsf{S}$ for some $\Gamma \in (\mathsf{V_t} \cup \mathcal{F})^*$.

Definition 3.10. Earley's deductive parser $\mathcal{CF}_{\mathtt{Earley}}$.

Let a CF grammar $\mathsf{G} = \langle \mathsf{V_t}, \mathsf{S}, \mathcal{F}, \mathcal{P} \rangle$ and a string $w_1 \dots w_{\mathfrak{n}}$ be given.

The parsing system $\mathcal{CF}_{\mathtt{Earley}}$ is the triple $\langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ defined in Figure 3.4 on the next page.

The system $\mathcal{CF}_{\mathtt{Earley}}$ 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:

$$\mathcal{W} = \{\, (\mathfrak{i}, \Delta \bullet \Lambda \to \mathsf{A}, \mathfrak{j}) \mid w_{\mathfrak{i}+1} \dots w_{\mathfrak{j}} \Rightarrow^* \Delta,\ w_1 \dots w_{\mathfrak{i}}\, \mathsf{A}\, \Gamma \Rightarrow^* \mathsf{S} \text{ for some } \Gamma \in (\mathsf{V_t} \cup \mathcal{F})^* \,\}$$

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

Example 3.7. Deduction of [[]] in grammar $[\mathsf{S}]\mathsf{S} \mid \epsilon \to \mathsf{S}$.

Items generated:

1. $(0, \bullet[\mathsf{S}]\mathsf{S} \to \mathsf{S}, 0)$: *Init*
2. $(0, \bullet \to \mathsf{S}, 0)$: *Init*
3. $(0, [\bullet\mathsf{S}]\mathsf{S} \to \mathsf{S}, 1)$: *Scan* 1
4. $(1, \bullet[\mathsf{S}]\mathsf{S} \to \mathsf{S}, 1)$: *Predict* 3

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

Figure 3.4: The parsing system $\mathcal{CF}_{\texttt{Earley}}$.

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, if we assume that predicted items are leaves of the proof tree (the place of hypotheses), although they are in fact derived. We present part of the previous deduction in tree format:

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

Soundness of $\mathcal{CF}_{\mathtt{Earley}}$ is trivial. In order to prove *completeness*, one can define a deduction length function $\mathtt{d}$ as

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

One should check that $\mathtt{d}$ satisfies condition 2 of Definition 3.9 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 $\mathsf{AB}^{\otimes}$ grammars and prove its correctness.

3.4.2 The Earley parser for $\mathsf{AB}^{\otimes}$ grammars

Consider the following formulation of the AB calculus with product.

Definition 3.11. Let $\overline{\mathsf{AB}}^{\otimes}$ be the deductive system $\langle \mathcal{F}, \mathsf{AX}, \mathsf{R} \rangle$, where AX contains, for all $a, b \in \mathcal{F}$:

- Identity axioms:

$$a \to a$$

- Slash axioms:

$$a\ a\backslash b \to b \qquad b/a\ a \to b$$

- Product axioms:

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

R consists of the cut rule:

$$\dfrac{\Delta \to a \quad \Gamma[a] \to c}{\Gamma[\Delta] \to c}$$

An AB grammar with product can be constructed on top of $\overline{\mathsf{AB}}^{\otimes}$ in the usual way.

Proposition 3.6. [Kandulski, 1988]: The system $\mathsf{AB}^{\otimes}$ presented in Definition 2.24 on page 21 and $\overline{\mathsf{AB}}^{\otimes}$ are equivalent.

The slash and product axioms of $\overline{\mathsf{AB}}^{\otimes}$ will be applied in the Earley deductive system as different instances of the *prediction* rule. In the case of CF grammars, prediction expands a non-terminal X in accordance with the productions whose succedent is X. In case of $\overline{\mathsf{AB}}^{\otimes}$, the system will *expand* a formula $\mathfrak{a}$ as the list of formulas $\mathfrak{b}$ $\mathfrak{b}\backslash\mathfrak{a}$ (and/or $\mathfrak{a}/\mathfrak{b}$ $\mathfrak{b}$), such that $\mathfrak{b}\backslash\mathfrak{a}$ (and/or $\mathfrak{a}/\mathfrak{b}$) is a subformula of some lexical assumption. Hence for *slash prediction* we will have to keep note of the slash subformulas taking part in the computation.

Definition 3.12. We define a function, $\delta^* :: \mathcal{F} \to \{\mathcal{F}\}$, returning the set of slash subformulas needed for expanding the slash axioms in the parsing system.

$$\begin{aligned}\delta^*(\mathfrak{a}/\mathfrak{b}) &= \{\mathfrak{a}/\mathfrak{b}\} \cup \delta^*(\mathfrak{a})\\ \delta^*(\mathfrak{b}\backslash\mathfrak{a}) &= \{\mathfrak{b}\backslash\mathfrak{a}\} \cup \delta^*(\mathfrak{a})\\ \delta^*(\_) &= \emptyset\end{aligned}$$

We present the Earley deductive parser for $\mathsf{AB}^{\otimes}$ grammars.

Definition 3.13. The parsing system $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$.

Let an $\overline{\mathsf{AB}}^{\otimes}$ grammar $\mathsf{G} = \langle \mathsf{V_t}, \mathsf{s}, \mathsf{Lex}, \overline{\mathsf{AB}}^{\otimes}\rangle$ and a string $w_1 \dots w_n$ be given. Let S' be a symbol that does not occur in any lexical entry of G.

A set of formulas Σ is constructed as follows:

$$\begin{gathered}\Sigma = \{\, \mathfrak{b} \mid w_i \to \mathfrak{a} \in \mathsf{Lex},\ \mathfrak{b} \in \delta^*(\mathfrak{a}) \,\}\\ \cup\\ \{\, \mathfrak{b} \mid \epsilon \to \mathfrak{a} \in \mathsf{Lex},\ \mathfrak{b} \in \delta^*(\mathfrak{a}) \,\}\end{gathered}$$

Figure 3.5 on the next page presents the Earley deductive parser $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$.

Other formulations of the system $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ are possible. We chose this slightly verbose one as it parallels the Earley system for CF grammars and in Definition 3.16 we propose a simplified version of $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$. The *Init* rule has only an auxiliary status in that it allows us to avoid to state explicitly prediction for the root symbol s, saving us some space. It makes use of a new symbol S' that does not occur elsewhere in the grammar. The scanning rule could indeed be simplified by *contraction* with the lexical instance of prediction (see Remark 3.9 at the end of the section). However, it amounts to the same as the scanning rule of $\mathcal{CF}_{\mathtt{Earley}}$. The completion rule is exactly the same as the corresponding CF rule.

The *prediction* rule of $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ is new. In the case of $\mathcal{CF}_{\mathtt{Earley}}$, prediction is the following rule

$$\frac{(\mathsf{i},\ \Delta \bullet \mathsf{A}\,\Lambda \to \mathsf{C},\ \mathsf{j})}{(\mathsf{j},\ \bullet\,\Gamma \to \mathsf{A},\ \mathsf{j})}\ \Gamma \to \mathsf{A} \in \mathsf{AX}$$

$$
\begin{array}{rcll}
\mathcal{I} & = & \{\, (i, \Gamma \bullet \Delta \to c, j) \mid \Gamma\Delta \to c \in (AX \cup Lex),\ 0 \leqslant i \leqslant j \leqslant n \,\} & \\
\mathcal{A} & = & \{\, (i-1, w, i) \mid w \equiv w_i,\ 1 \leqslant i \leqslant n \,\} & \\
\mathcal{R} & = & \left\{ \begin{array}{ll}
(0, \bullet s \to S', 0) & \textit{Init} \\
\dfrac{(j, \bullet w \to c, j) \quad (j, w, j+1)}{(j, w\bullet \to c, j+1)} & \textit{Scanning} \\
\dfrac{(i,\ \Delta \bullet a\Lambda \to c,\ j)}{(j,\ \bullet w \to a,\ j)}\ w \to a \in Lex & \\
\dfrac{(i,\ \Delta \bullet b\Lambda \to c,\ j)}{(j,\ \bullet a\ a\backslash b \to b,\ j)}\ a\backslash b \in \Sigma & \textit{Prediction} \\
\dfrac{(i,\ \Delta \bullet b\Lambda \to c,\ j)}{(j,\ \bullet b/a\ a \to b,\ j)}\ b/a \in \Sigma & \\
\dfrac{(i,\ \Delta \bullet a \otimes b\, \Lambda \to c,\ j)}{(j,\ \bullet a\ b \to a \otimes b,\ j)} & \\
\dfrac{(k, \Lambda\bullet \to a, j) \quad (i, \Delta \bullet a\Gamma \to c, k)}{(i, \Delta\, a \bullet \Gamma \to c, j)} & \textit{Completion}
\end{array} \right.
\end{array}
$$

Figure 3.5: The system $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$.

In the case of $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$, we have three kinds of prediction, which we call lexical, slash and product prediction in accordance with the three kinds of axioms of the $\overline{\mathsf{AB}}^{\otimes}$ system which these rules rely upon.

- *Lexical* prediction,

$$\frac{(i,\ \Delta \bullet a\Lambda \to c,\ j)}{(j,\ \bullet w \to a,\ j)}\ w \to a \in Lex$$

  generates a new item (the conclusion), if the next symbol to be parsed is a lexical category. Observe that w, in the conclusion, can be the empty string.

- *Product* prediction

$$\frac{(\mathsf{i},\ \Delta \bullet \mathsf{a} \otimes \mathsf{b}\, \Lambda \rightarrow \mathsf{c},\ \mathsf{j})}{(\mathsf{j},\ \bullet\, \mathsf{a}\ \mathsf{b} \rightarrow \mathsf{a} \otimes \mathsf{b},\ \mathsf{j})}$$

  applies the product axiom

$$\mathsf{a}\ \mathsf{b} \rightarrow \mathsf{a} \otimes \mathsf{b}$$

  The rule states that if the next symbol to be parsed is of the form $\mathsf{a} \otimes \mathsf{b}$, then we can start a new subprocess which tries to recognize the two immediate subformulas a and b of the product.

- *Slash* prediction (we consider only one variant),

$$\frac{(\mathsf{i},\ \Delta \bullet \mathsf{b}\Lambda \rightarrow \mathsf{c},\ \mathsf{j})}{(\mathsf{j},\ \bullet\, \mathsf{a}\ \mathsf{a}\backslash\mathsf{b} \rightarrow \mathsf{b},\ \mathsf{j})}\ \mathsf{a}\backslash\mathsf{b} \in \Sigma$$

  is triggered by a slash subformula of the form $\mathsf{a}\backslash\mathsf{b}$ in the set Σ. If the 'numerator' of a formula in Σ matches the next symbol to be parsed in the input item, then a new process starts, trying to recognize the symbol as the output category of the slash axiom

$$\mathsf{a}\ \mathsf{a}\backslash\mathsf{b} \rightarrow \mathsf{b}$$

In comparison with $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$, one can observe that the product axioms is applied in $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ in a completely natural way. On the other hand, the *reduction* schemes $\mathsf{a}\ \mathsf{a}\backslash\mathsf{b} \rightarrow \mathsf{b}$ and $\mathsf{b}/\mathsf{a}\ \mathsf{a} \rightarrow \mathsf{b}$ of the AB calculus, become *expanding* from the top-down perspective: the output formula b is what is given, while the lists of formulas $\mathsf{a}\ \mathsf{a}\backslash\mathsf{b}$ or $\mathsf{b}/\mathsf{a}\ \mathsf{a}$ represent the new (though predictable) material.

We give now an example of application of the parsing system.

Example 3.8.

Input string: $\mathtt{aabb}$.

Input grammar:

$$\begin{array}{l}\mathsf{a} \rightarrow \mathsf{s}/(\mathsf{s} \otimes \mathsf{c})\\ \mathsf{b} \rightarrow \mathsf{c}\\ \epsilon \rightarrow \mathsf{s}\end{array}$$

$\Sigma = \{\mathsf{s}/(\mathsf{s} \otimes \mathsf{c})\}$

Items generated (we restrict the set to those needed for a successful deduction):

1. $(0, \bullet\mathsf{s} \rightarrow \mathsf{S}', 0)$: *Init*
2. $(0, \bullet\mathsf{s}/(\mathsf{s} \otimes \mathsf{c})\ \mathsf{s} \otimes \mathsf{c} \rightarrow \mathsf{s}, 0)$: *Predict* 1
3. $(0, \bullet\mathsf{a} \rightarrow \mathsf{s}/(\mathsf{s} \otimes \mathsf{c}), 0)$: *Predict* 2
4. $(0, \mathsf{a}\bullet \rightarrow \mathsf{s}/(\mathsf{s} \otimes \mathsf{c}), 1)$: *Scan* 3

5. $(0, \mathsf{s}/(\mathsf{s} \otimes \mathsf{c}) \bullet \mathsf{s} \otimes \mathsf{c} \to \mathsf{s}, 1)$: *Complete* 4-2
6. $(1, \bullet \mathsf{s}\ \mathsf{c} \to \mathsf{s} \otimes \mathsf{c}, 1)$: *Predict* 5
7. $(1, \bullet \mathsf{s}/(\mathsf{s} \otimes \mathsf{c})\ \mathsf{s} \otimes \mathsf{c} \to \mathsf{s}, 1)$: *Predict* 6
8. $(1, \bullet \mathfrak{a} \to \mathsf{s}/(\mathsf{s} \otimes \mathsf{c}), 1)$: *Predict* 7
9. $(1, \mathfrak{a} \bullet \to \mathsf{s}/(\mathsf{s} \otimes \mathsf{c}), 2)$: *Scan* 8
10. $(1, \mathsf{s}/(\mathsf{s} \otimes \mathsf{c}) \bullet \mathsf{s} \otimes \mathsf{c} \to \mathsf{s}, 2)$: *Complete* 7-9
11. $(2, \bullet \mathsf{s}\ \mathsf{c} \to \mathsf{s} \otimes \mathsf{c}, 2)$: *Predict* 10
12. $(2, \bullet \to \mathsf{s}, 2)$: *Predict* 11
13. $(2, \mathsf{s} \bullet \mathsf{c} \to \mathsf{s} \otimes \mathsf{c}, 2)$: *Complete* 11-12
14. $(2, \bullet \mathsf{b} \to \mathsf{c}, 2)$: *Predict* 13
15. $(2, \mathsf{b} \bullet \to \mathsf{c}, 3)$: *Scan* 14
16. $(2, \mathsf{s}\ \mathsf{c} \bullet \to \mathsf{s} \otimes \mathsf{c}, 3)$: *Complete* 15-13
17. $(1, \mathsf{s}/(\mathsf{s} \otimes \mathsf{c})\ \mathsf{s} \otimes \mathsf{c} \bullet \to \mathsf{s}, 3)$: *Complete* 16-10
18. $(1, \mathsf{s} \bullet \mathsf{c} \to \mathsf{s} \otimes \mathsf{c}, 3)$: *Complete* 17-6
19. $(3, \bullet \mathsf{b} \to \mathsf{c}, 3)$: *Predict* 18
20. $(3, \mathsf{b} \bullet \to \mathsf{c}, 4)$: *Scan* 19
21. $(1, \mathsf{s}\ \mathsf{c} \bullet \to \mathsf{s} \otimes \mathsf{c}, 4)$: *Complete* 20-18
22. $(0, \mathsf{s}/(\mathsf{s} \otimes \mathsf{c})\ \mathsf{s} \otimes \mathsf{c} \bullet \to \mathsf{s}, 4)$: *Complete* 21-5
23. $(0, \mathsf{s} \bullet \to \mathsf{S}', 4)$: *Complete* 22-1

Correctness of $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$

In this section, we state the correctness of $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$. However, we omit some of the details of the proofs and refer the reader to the proof of correctness of the system $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$, in the next section, that can be easily adapted to the present case.

Definition 3.14. Viable items of the system $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$, for an $\mathrm{AB}^{\otimes}$ grammar G and a string $w_1 \dots w_n$.

$$\mathcal{W} = \{\, (i, \Delta \bullet \Lambda \to \mathsf{c}, j) \mid w_{i+1} \dots w_j \Rightarrow^* \Delta, \\ w_1 \dots w_i\, \mathsf{c}\, \Gamma \Rightarrow^* \mathsf{s} \text{ for some } \Gamma \in (\mathsf{V_t} \cup \mathcal{F})^* \,\}$$

Proposition 3.7. *Soundness* of $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$.

$$\mathcal{V}(\mathcal{AB}^{\otimes}_{\mathtt{Earley}}) \subseteq \mathcal{W}$$

Proof. Induction on the $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ deduction. □

In order to prove completeness of $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ we define the following deduction length function.

Definition 3.15. We define a *dfl* $\mathtt{d} :: (\mathcal{A} \cup \mathcal{W}) \to \mathtt{Int}$ by

$$\begin{aligned}
\mathtt{d}(i-1, w, i) \quad &= 0 \\
\mathtt{d}(i, \Delta \bullet \Gamma \to a, j) \; &= \mathtt{min}\{\, \pi + 2\mu + 2\zeta + j \mid w_{i+1} \ldots w_j \Rightarrow^{\mu} \Delta, \\
&\qquad\qquad \Xi\, a\, \Lambda \Rightarrow^{\pi} s, \\
&\qquad\qquad w_1 \ldots w_i \Rightarrow^{\zeta} \Xi \,\}
\end{aligned}$$

Proposition 3.8. System $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ is *complete*.

$$\mathcal{W} \subseteq \mathcal{V}(\mathcal{AB}^{\otimes}_{\mathtt{Earley}})$$

Proof. As in the case of $\mathcal{CF}_{\mathtt{Earley}}$, one can easily check that $\mathtt{d}$ satisfies condition 2 of Definition 3.9 for every item in $\mathcal{W}$. We consider only a few cases, as the proof is very similar to the corresponding proof for $\mathcal{CF}_{\mathtt{Earley}}$ and to the one for $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ in the next section.

1. Consider $\xi = (i, \Delta\, b \bullet \Gamma \to a, j)$. Let $w_{i+1} \ldots w_m \Rightarrow^{\mu} \Delta$, $\Xi a \Lambda \Rightarrow^{\pi} s$ and $w_1 \ldots w_i \Rightarrow^{\zeta} \Xi$ with μ, π and ζ minimal. Then $\eta = (i, \Delta \bullet b\, \Gamma \to a, m) \in \mathcal{W}$ and

   $$\mathtt{d}(\eta) = \pi + 2\mu + 2\zeta + m$$

   Let $w_{m+1} \ldots w_j \Rightarrow^{\mu'} \Delta'$ with μ' minimal and $\Delta' \to b \in \mathcal{AX}$. Then $\theta = (m, \Delta' \to b, j) \in \mathcal{W}$. For θ, we have $\Xi\Delta\, b\, \Gamma\Lambda \Rightarrow \Xi a \Lambda \Rightarrow^{\pi} s$ and $w_1 \ldots w_m \Rightarrow^{\zeta+\mu} \Xi\Delta$. Thus,

   $$\mathtt{d}(\theta) = (\pi + 1) + 2\mu' + 2(\mu + \zeta) + j$$

   For ξ, we have $w_{i+1} \ldots w_j \Rightarrow^{\mu+\mu'} \Delta\Delta' \Rightarrow \Delta b$.

   $$\mathtt{d}(\xi) = \pi + 2(\mu + \mu' + 1) + 2\zeta + j$$

   Hence $\mathtt{d}(\xi) \geqslant \mathtt{d}(\theta) + 1$ and $\mathtt{d}(\xi) \geqslant \mathtt{d}(\eta) + (j - m + 2)$

2. Consider the case of prediction of $\xi = (j, \bullet a\, b \to a \otimes b, j)$. Suppose that $\Delta\, a \otimes b\, \Gamma \to c \in \mathcal{AX}$, that for some i, $0 \leqslant i \leqslant j$, $w_{i+1} \ldots w_j \Rightarrow^{\mu} \Delta$, that $\Xi\, c\, \Lambda \Rightarrow^{\pi} s$ and $w_1 \ldots w_i \Rightarrow^{\zeta} \Xi$ with μ, π and ζ minimal. Then $\eta = (i, \Delta \bullet a \otimes b\, \Gamma \to c, j) \in \mathcal{W}$, ξ is derived from η and

   $$\mathtt{d}(\eta) = \pi + 2\mu + 2\zeta + j$$

For ξ, we have $\Xi\Delta\, a \otimes b\, \Gamma\Lambda \Rightarrow \Xi\, c\, \Lambda \Rightarrow^{\pi} s$ and $w_1 \dots w_j \Rightarrow^{\mu+\zeta} \Xi\Delta$. Hence,

$$d(\xi) = (\pi + 1) + 2(\mu + \zeta) + j$$

We conclude $d(\xi) \leqslant d(\eta) + 1$.

3. The other cases of prediction are similar to the product prediction, while scanning is trivial and *init* has only an auxiliary status. Thus we omit the analysis of these rules.

□

As we said before, $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$ can be simplified. In some cases, we can 'collapse' several deduction steps into simpler inference rules. This process is called *contraction* and it is a special kind of *filtering* according to the terminology of [Sikkel and Nijholt, 1997]. In the following example, we show how we can contract scanning and lexical prediction into simpler rules. Then we reformulate $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$ according to these new rules.

Example 3.9. Contraction of scanning and lexical prediction.

Consider the following deduction:

$$\dfrac{\dfrac{\dfrac{(i,\, \Delta \bullet a\Lambda \to c,\, j)}{(j,\, \bullet w \to a,\, j)}\; w \to a \in \mathsf{Lex} \qquad (j, w, j+1)}{(j,\, w\bullet \to a,\, j+1)} \qquad (i,\, \Delta \bullet a\Lambda \to c,\, j)}{(i,\, \Delta\, a \bullet \Lambda \to c,\, j+1)}$$

As $w \neq \epsilon$, the following inference rule will have the same effect of the previous deduction.

$$\dfrac{(i,\, \Delta \bullet a\Lambda \to c,\, j)}{(i,\, \Delta a \bullet \Lambda \to c,\, j+1)}\; w_{j+1} \to a \in \mathsf{Lex}$$

Instead, if $\epsilon \Rightarrow^{+} a$, we can use the following rule.

$$\dfrac{(i,\, \Delta \bullet a\Lambda \to c,\, j)}{(i,\, \Delta\, a \bullet \Lambda \to c,\, j)}\; \epsilon \Rightarrow^{+} a$$

We may observe that if we replace *scanning* and *lexical prediction* in $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$ with these two new rules, we can reduce the number of items generated and of the inference steps required to generate their conclusions. Instead, all transitions $\epsilon \Rightarrow^{+} a$ can be precomputed, as it is standard practice in the implementation of the Earley parser for CF grammars (see for example [Harrison, 1978] and [Graham et al., 1980] and also Chapter 4).

We rephrase $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$ using the rules in Example 3.9 and assuming as the only axiom the previous *Init* rule.

Definition 3.16. Contracted $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$.
Let Σ be constructed as follows:

$$\Sigma = \{\, b \mid w_i \to a \in \mathsf{Lex},\ b \in \delta^*(a) \,\}$$

The contracted $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$ is presented in Figure 3.6.

$$\begin{array}{rcll}
\mathcal{I} & = & \{\, (i, \Gamma \bullet \Delta \to c, j) \mid \Gamma\Delta \to c \in \mathsf{AX},\ 0 \leqslant i \leqslant j \leqslant n \,\} & \\
\mathcal{A} & = & \{\, (0, \bullet s \to S', 0) \,\} & \\
\mathcal{R} & = & \left\{ \begin{array}{l}
\left.\begin{array}{l}
\dfrac{(i,\ \Delta \bullet a\Lambda \to c,\ j)}{(i,\ \Delta a \bullet \Lambda \to c,\ j+1)}\ w_{j+1} \to a \in \mathsf{Lex} \\[2ex]
\dfrac{(i,\ \Delta \bullet a\Lambda \to c,\ j)}{(i,\ \Delta\, a \bullet \Lambda \to c,\ j)}\ \epsilon \Rightarrow^{+} a
\end{array}\right\} \textit{Scanning} \\[4ex]
\left.\begin{array}{l}
\dfrac{(i,\ \Delta \bullet b\Lambda \to c,\ j)}{(j,\ \bullet\, a\ \ a\backslash b \to b,\ j)}\ a\backslash b \in \Sigma \\[2ex]
\dfrac{(i,\ \Delta \bullet b\Lambda \to c,\ j)}{(j,\ \bullet\, b/a\ \ a \to b,\ j)}\ b/a \in \Sigma \\[2ex]
\dfrac{(i,\ \Delta \bullet a \otimes b\, \Lambda \to c,\ j)}{(j,\ \bullet\, a\ \ b \to a \otimes b,\ j)}
\end{array}\right\} \textit{Prediction} \\[4ex]
\dfrac{(k, \Lambda\bullet \to a, j) \quad (i, \Delta \bullet a\Gamma \to c, k)}{(i, \Delta\, a \bullet \Gamma \to c, j)} \quad \textit{Completion}
\end{array} \right. &
\end{array}$$

Figure 3.6: Contracted $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$.

We observe that this parsing system is different from the one presented in Definition 3.13 on page 59 also under a few other minor respects. The set Σ is smaller (or equal) as we assume that we know already what the empty string rewrites to. Besides, items contain only sequents in AX, rather than in $\mathsf{AX} \cup \mathsf{Lex}$. Finally, the only axiom of the system is the *Init* rule of the system in Definition 3.13.

3.5 Mixed regime

In the previous sections, we considered the bottom-up and the top-down approaches to parsing. The parsing systems we presented for $\mathsf{AB}^{\otimes}$ grammar were simple adaptations of the well known CYK and Earley parsing systems for CF grammars. However, the systems $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$ and $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ both present some inadequacy which is absent from their CF analogs. In both parsing systems, in fact, the applications of some rules has to be licensed by a given set of formulas. One may notice the *complementarity* of the Σ sets driving inferences in the two systems: in $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$ the slash elimination rules are 'safe', while the product rule should be constrained in order to generate only products in Σ; in $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$, on the contrary, the product rule is 'safe', while the slash prediction rules should be applied on the basis of the set Σ. While this situation does not affect significantly the complexity of an algorithm implementing these parsing systems, since, in general, the Σ set is smaller than the grammar that triggers the rules in the CF case, we can however try to find a better solution.

We shall emphasize that the CYK system is to a large extent adequate for $\mathsf{AB}^{\otimes}$ parsing since all the $\mathsf{AB}^{\otimes}$ rules are binary. Instead, the Earley system has the great advantage of handling the product formulas in a completely natural way. Moreover, Earley can tackle grammars assigning syntactic categories to the empty string.

In this section, I present a parsing system for $\mathsf{AB}^{\otimes}$ grammars that includes *all* and *only* the pleasant features of $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$ and $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$. The system, which we call $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$, needs no other information than that encoded in the logical axioms. This means that no subformula test is required for either product or slash formulas. This parser results from the fusion of $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$ and $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$: it works partly top-down and partly bottom-up and covers larger portions of the input both from *left to right* and from *right to left*[6].

Inferences of $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ are driven in conformity with the following axiomatization of the $\mathsf{AB}^{\otimes}$ calculus.

Definition 3.17. We call $\widetilde{\mathsf{AB}}^{\otimes}$ the triple $\langle \mathcal{F}, \widetilde{\mathsf{AX}}^{\star}, \mathsf{R} \rangle$ where $\widetilde{\mathsf{AX}}^{\star} = \mathsf{P} \cup \widetilde{\mathsf{AX}}^{*}$ and:

1. For all formulas $\mathfrak{a}$ and $\mathfrak{b}$, P is the set containing all axioms of the form:

$$\mathfrak{a}\ \mathfrak{b} \rightarrow \mathfrak{a} \otimes \mathfrak{b}$$

2. Let $\widetilde{\mathsf{AX}}^{0}$ contain the identity axioms, for all formulas $\mathfrak{a}$:

$$\mathfrak{a} \rightarrow \mathfrak{a}$$

[6] This *left-to-right* and *right-to-left* interaction is original. Clearly, it is possible to rephrase the Earley system to work from right to left, since the direction of the procedure is completely arbitrary. On the other hand, the mixed regime that we are going to define seems to make sense for categorial grammars, though not for CF grammars.

3. $\widetilde{AX}^*$ is the closure of $\widetilde{AX}^0$ under the following rules:

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

4. R consists of the cut rule:

$$\frac{\Gamma \to a \quad \Delta[a] \to c}{\Delta[\Gamma] \to c}$$

Clearly, the system $\widetilde{AB}^{\otimes}$ is equivalent to $AB^{\otimes}$. An $\widetilde{AB}^{\otimes}$ grammar is a grammar based on $\widetilde{AB}^{\otimes}$.

We present the system $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$. This parsing system operates on two kinds of items which we represent as

$$(i, \Delta \triangleright \Gamma \to a, j) \quad \text{and} \quad (i, \Gamma \triangleleft \Delta \to a, j)$$

An item of the form $(i, \Delta \triangleright \Gamma \to a, j)$ asserts that $\Delta\Gamma \to a \in \widetilde{AX}^{\star}$, that $w_{i+1} \dots w_j \Rightarrow^* \Delta$, and that if, in particular, $\Delta\Gamma \to a \in P$, then for some k, $0 \leqslant k < i$, $c \in \mathcal{F}$ and $\Lambda \in \mathcal{F}^*$, $\Xi a \Lambda \to c \in \widetilde{AX}^{\star}$ and $w_{k+1} \dots w_i \Rightarrow^* \Xi$. This last condition represents the *predictive* component of the parser. It states that each item $(i, \Delta \triangleright \Gamma \to a' \otimes a'', j)$ such that $\Delta\Gamma \to a' \otimes a''$ is a product axiom from P (hence, $\Delta\Gamma \equiv a'\, a''$) has been predicted from an item $(k, \Xi \triangleright a' \otimes a'' \Lambda \to c, i)$.

Dually, an item of the form $(i, \Gamma \triangleleft \Delta \to a, j)$ asserts that $\Gamma\Delta \to a \in \widetilde{AX}^{\star}$, that $w_{i+1} \dots w_j \Rightarrow^* \Delta$ and that $\Lambda a \Xi \to c \in \widetilde{AX}^{\star}$ with $w_{j+1} \dots w_l \Rightarrow^* \Xi$ for some l, $j < l \leqslant n$, $\Lambda \in \mathcal{F}^*$ and $c \in \mathcal{F}$.

Thus items of $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ can be seen as a refinement of both $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ items and $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$ items. Completion of wider portions of the input is dualized (partly left-to-right and partly right-to-left) according to the directionality of the slashes, as in $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$, while the sequent formulation of the items enables, in the product case, the predictive behavior of $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$.

For simplicity, we write items of the form $(i, \triangleleft \Delta \to a, j)$ and $(i, \Delta \triangleright \to a, j)$ as $(i, \Delta \to a, j)$.

Definition 3.18. Let an $\widetilde{AB}^{\otimes}$ grammar G and a string $w_1 \dots w_n$ be given. The $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ deductive parser is the triple $\langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ presented in Figure 3.7 on the next page.

Step by step description

Axioms of $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ are items of the form $(i-1, a \to a, i)$ whose sequent *recognizes* the word w_i as an expression of category a, if the assignment $w_i \to a$ is in the lexicon of the input grammar. Let us spell out the meaning of the rules.

$$\mathcal{I} = \begin{array}{c} \{\, (i, \Gamma \triangleright \Delta \to a, j) \mid \Gamma\Delta \to a \in \widetilde{AX}^{\star},\ 0 \leqslant i \leqslant j \leqslant n \,\} \\ \cup \\ \{\, (i, \Gamma \triangleleft \Delta \to a, j) \mid \Gamma\Delta \to a \in \widetilde{AX}^{\star},\ 0 \leqslant i \leqslant j \leqslant n \,\} \end{array}$$

$$\mathcal{A} = \{\, (i-1, a \to a, i) \mid w_i \to a \in \mathsf{Lex} \,\}$$

$$\mathcal{R} = \left\{ \begin{array}{ll} \left.\begin{array}{l} \dfrac{(i, \Delta \triangleright a\, \Gamma \to c,\ j)}{(i, \Delta\, a \triangleright \Gamma \to c,\ j)}\ \epsilon \Rightarrow^{+} a \\[2ex] \dfrac{(i, \Gamma\, a \triangleleft \Delta \to c,\ j)}{(i, \Gamma \triangleleft a\, \Delta \to c,\ j)}\ \epsilon \Rightarrow^{+} a \end{array}\right\} & \epsilon\text{-}\textit{Scanning} \\[4ex] \left.\dfrac{(i, \Delta \to c/b,\ j)}{(i, \Delta \triangleright b \to c,\ j)} \quad \dfrac{(i, \Delta \to b\backslash c,\ j)}{(i, b \triangleleft \Delta \to c,\ j)}\right\} & \textit{Shifting} \\[4ex] \left.\dfrac{(i, \Delta \triangleright a \otimes b\, \Gamma \to c,\ j)}{(j, \triangleright a\ b \to a \otimes b,\ j)} \quad \dfrac{(i, \Gamma\, a \otimes b \triangleleft \Delta \to c,\ j)}{(i, a\ b \triangleleft \to a \otimes b,\ i)}\right\} & \otimes\text{-}\textit{Prediction} \\[4ex] \left.\begin{array}{l} \dfrac{(i, \Delta \triangleright a\, \Gamma \to c, k) \quad (k, \Lambda \to a, j)}{(i, \Delta\, a \triangleright \Gamma \to c, j)} \\[2ex] \dfrac{(i, \Lambda \to a, k) \quad (k, \Delta\, a \triangleleft \Gamma \to c, j)}{(i, \Delta \triangleleft a\, \Gamma \to c, j)} \end{array}\right\} & \textit{Completion} \end{array} \right.$$

Figure 3.7: The system $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$.

1. ϵ-*scanning* deals *only* with $\epsilon \Rightarrow^{+} a$ reductions. If the next formula to be parsed a can be reached from the empty string, then we can move a among the recognized formulas (left of $\triangleright$ or right of $\triangleleft$).

2. When an antecedent structure is completely analyzed and the succedent formula is a slash formula, *shifting* may apply. It places the 'denominator' of the succedent slash formula to the position of the next symbol to be parsed, keeping track of the slash orientation in the orientation of the triangle, expressing the direction of the parsing process.

3. $\otimes$-*prediction* handles *only* the product formulas. The only difference with respect to the product prediction of $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$ is that instead of the single $\bullet$-marker, we have here the two oriented markers $\triangleright$ and $\triangleleft$. In the conclusion, recognition proceeds according to the direction of the premise that triggered the rule.

4. The *completion* rules are oriented variants of the completion rule of $\mathcal{AB}^{\otimes}_{\texttt{Earley}}$. The left-to-right variant is as in the Earley system. Let us look at the right-to-left variant:

$$\frac{(i, \Lambda \rightarrow a, k) \quad (k, \Delta\, a \triangleleft \Gamma \rightarrow c, j)}{(i, \Delta \triangleleft a\, \Gamma \rightarrow c, j)}$$

   The item $(k, \Delta\, a \triangleleft \Gamma \rightarrow c, j)$ states that the list of formulas Γ covers the input between $k{+}1$ and j, and in order to proceed in the analysis, it should recognize an expression of category a to its left. Then since $(i, \Lambda \rightarrow a, k)$ represents a *completed* analysis of the portion of the input between $i+1$ and k as being of category a, we conclude that the item $(i, \Delta \triangleleft a\, \Gamma \rightarrow c, j)$ covers the analysis of the portion of input between $i+1$ and j. The other case of completion is the symmetric dual.

Example 3.10. In Figure 3.8 on the following page, we show the deduction in $\mathcal{AB}^{\otimes}_{\texttt{Mix}}$ of $\neg[p \wedge q]$ according to grammar PL, from Example 3.6 on page 52.

Correctness of $\mathcal{AB}^{\otimes}_{\texttt{Mix}}$

We prove correctness of $\mathcal{AB}^{\otimes}_{\texttt{Mix}}$ by defining a set of viable items which formalizes the description of items given at the beginning of the section and then a deduction length function which we use to prove completeness of the system.

Definition 3.19. We define the set $\mathcal{W}$ of viable items of $\mathcal{AB}^{\otimes}_{\texttt{Mix}}$ as follows:

$$\begin{array}{lcl}
\{\, (i, \Delta \triangleright \Gamma \rightarrow a, j) & \mid & w_{i+1} \ldots w_j \Rightarrow^* \Delta, \\
& & \Xi\, a\, \Lambda \Rightarrow^* c, \\
& & w_{k+1} \ldots w_i \Rightarrow^* \Xi,\ 0 \leqslant k < i \,\} \\
& & \cup \\
\{\, (i, \Gamma \triangleleft \Delta \rightarrow a, j) & \mid & w_{i+1} \ldots w_j \Rightarrow^* \Delta, \\
& & \Xi\, a\, \Lambda \Rightarrow^* c, \\
& & w_{j+1} \ldots w_l \Rightarrow^* \Lambda,\ j < l \leqslant n \,\}
\end{array}$$

Proposition 3.9. Soundness of $\mathcal{AB}^{\otimes}_{\texttt{Mix}}$.
$\mathsf{V}(\mathcal{D}) \subseteq \mathcal{W}$.

Proof. For each item in $\mathsf{V}(\mathcal{D})$, we shall prove that the constraints on viable items $\mathcal{W}$ are satisfied.

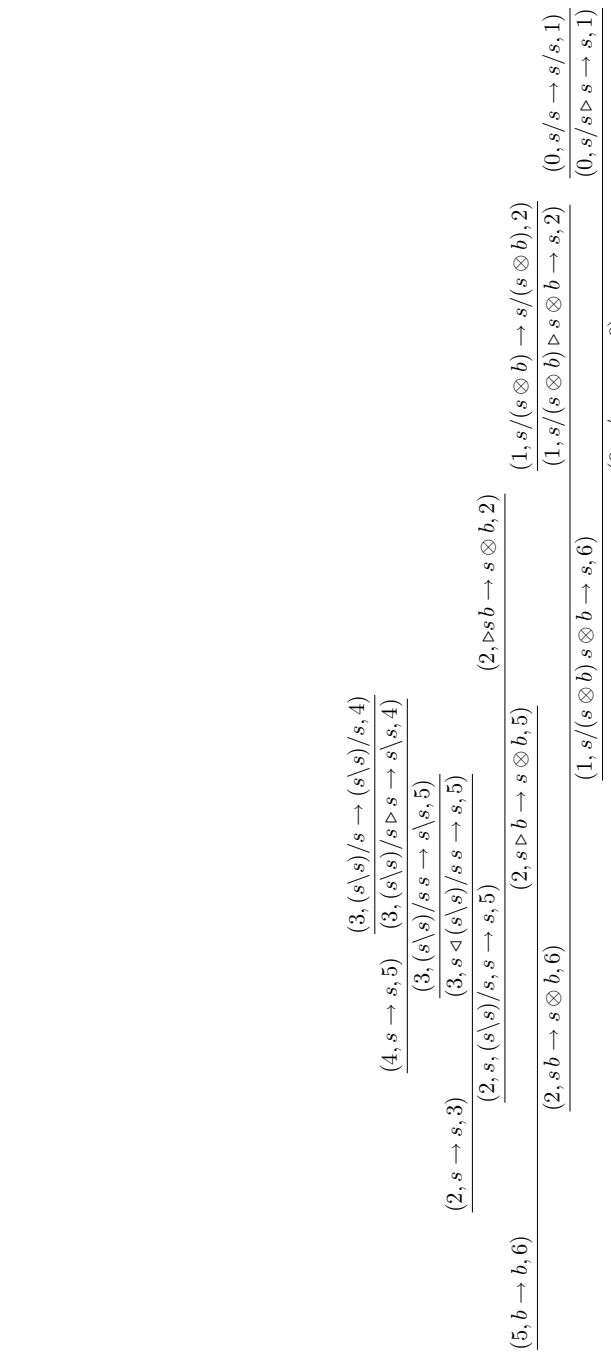


Figure 3.8: Deduction of $\neg[\mathsf{p} \wedge \mathsf{q}]$ in grammar PL.

1. $(\mathfrak{i}-1, \mathfrak{a} \to \mathfrak{a}, \mathfrak{i})$ is an axiom. Then $w_\mathfrak{i} \to \mathfrak{a} \in \mathsf{Lex}$. Hence $w_\mathfrak{i} \Rightarrow \mathfrak{a}$. Moreover, we take $\Xi \equiv \epsilon$, $\Lambda \equiv \epsilon$ and $\mathfrak{c} \equiv \mathfrak{a}$.

2. $(\mathfrak{i}, \Delta \triangleright \mathfrak{a} \to \mathfrak{b}, \mathfrak{j})$ is deduced from $(\mathfrak{i}, \Delta \to \mathfrak{b}/\mathfrak{a}, \mathfrak{j})$. By IH, $w_{\mathfrak{i}+1} \dots w_\mathfrak{j} \Rightarrow^* \Delta$ and $\Delta \to \mathfrak{b}/\mathfrak{a} \in \widetilde{\mathsf{AX}}^*$. Hence $\Delta\, \mathfrak{a} \to \mathfrak{b} \in \widetilde{\mathsf{AX}}^*$. Furthermore, $\Xi \equiv \epsilon$, $\Lambda \equiv \epsilon$ and $\mathfrak{c} \equiv \mathfrak{b}$.

3. $(\mathfrak{i}, \Delta\mathfrak{a} \triangleright \Gamma \to \mathfrak{b}, \mathfrak{j})$ is deduced from $(\mathfrak{i}, \Delta \triangleright \mathfrak{a}\Gamma \to \mathfrak{b}, \mathfrak{j})$ and $\epsilon \Rightarrow^+ \mathfrak{a}$. By IH, $w_{\mathfrak{i}+1} \dots w_\mathfrak{j} \Rightarrow^* \Delta$ and $\Xi'\, \mathfrak{b}\, \Lambda' \Rightarrow^* \mathfrak{c}'$ with $w_{\mathfrak{k}+1} \dots w_\mathfrak{i} \Rightarrow^* \Xi'$. We have $w_{\mathfrak{i}+1} \dots w_\mathfrak{j} \Rightarrow^* \Delta\, \mathfrak{a}$, $\Xi \equiv \Xi'$, $\Lambda \equiv \Lambda'$ and $\mathfrak{c} \equiv \mathfrak{c}'$.

4. $(\mathfrak{i}, \Delta\mathfrak{a} \triangleright \Gamma \to \mathfrak{b}, \mathfrak{j})$ is deduced from $(\mathfrak{i}, \Delta \triangleright \mathfrak{a}\Gamma \to \mathfrak{b}, \mathfrak{m})$ and $(\mathfrak{m}, \Gamma' \to \mathfrak{a}, \mathfrak{j})$. By IH, $w_{\mathfrak{i}+1} \dots w_\mathfrak{m} \Rightarrow^* \Delta$, $\Xi'\, \mathfrak{b}\, \Lambda' \Rightarrow^* \mathfrak{c}'$ with $w_{\mathfrak{k}+1} \dots w_\mathfrak{i} \Rightarrow^* \Xi'$ and $w_{\mathfrak{m}+1} \dots w_\mathfrak{j} \Rightarrow^* \Gamma'$. Hence $w_{\mathfrak{i}+1} \dots w_\mathfrak{j} \Rightarrow^* \Delta\Gamma' \Rightarrow \Delta\, \mathfrak{a}$, $\Xi \equiv \Xi'$, $\Lambda \equiv \Lambda'$ and $\mathfrak{c} \equiv \mathfrak{c}'$.

5. $(\mathfrak{j}, \triangleright \mathfrak{a}\ \mathfrak{b} \to \mathfrak{a} \otimes \mathfrak{b}, \mathfrak{j})$ is deduced from $(\mathfrak{i}, \Delta' \triangleright \mathfrak{a} \otimes \mathfrak{b}\, \Gamma \to \mathfrak{c}', \mathfrak{j})$. By IH, $w_{\mathfrak{i}+1} \dots w_\mathfrak{j} \Rightarrow^* \Delta'$ and $\Xi'\, \mathfrak{c}'\, \Lambda' \Rightarrow^* \mathfrak{c}''$ with $w_{\mathfrak{k}+1} \dots w_\mathfrak{i} \Rightarrow^* \Xi'$. We have $\Delta \equiv \epsilon$, $\mathfrak{c} \equiv \mathfrak{c}''$, $\Xi \equiv \Xi'\Delta'$ and $\Lambda \equiv \Gamma\Lambda'$.

□

In order to prove completeness, we shall define an appropriate *dfl*. Let us exemplify what parameters shall take part in its calculation.

Example 3.11. We consider the case of items of the form $(\mathfrak{i}, \Gamma \lhd \Delta \to \mathfrak{a}, \mathfrak{j})$.

Consider $(\mathfrak{i}, \Delta \to \mathfrak{b}\backslash\mathfrak{a}, \mathfrak{j})$ and let $\mathfrak{d}(\mathfrak{i}, \Delta \to \mathfrak{b}\backslash\mathfrak{a}, \mathfrak{j}) = \mathfrak{m}$. We have $\Delta \to \mathfrak{b}\backslash\mathfrak{a} \in \widetilde{\mathsf{AX}}^*$. Then also $\mathfrak{b}\, \Delta \to \mathfrak{a} \in \widetilde{\mathsf{AX}}^*$. For $(\mathfrak{i}, \mathfrak{b} \lhd \Delta \to \mathfrak{a}, \mathfrak{j})$, we should count at least $\mathfrak{m}+1$. The predictive component of $(\mathfrak{i}, \Delta \to \mathfrak{b}\backslash\mathfrak{a}, \mathfrak{j})$ is null, because prediction applies only product axioms. Thus also that of $(\mathfrak{i}, \mathfrak{b} \lhd \Delta \to \mathfrak{a}, \mathfrak{j})$ is null. Thus we count $\mathfrak{d}(\mathfrak{i}, \mathfrak{b} \lhd \Delta \to \mathfrak{a}, \mathfrak{j}) = \mathfrak{m} + |\Gamma|$ where $|\Gamma| = 1$. Observe that for every item $|\Gamma| \leqslant 2$.

Let $\mathfrak{d}(\mathfrak{r}, \Gamma\, \mathfrak{b} \lhd \Delta \to \mathfrak{a}, \mathfrak{j}) = \mathfrak{m}$. Suppose $\Delta' \to \mathfrak{b} \in \widetilde{\mathsf{AX}}^\star$ and $w_{\mathfrak{i}+1} \dots w_\mathfrak{r} \Rightarrow^\mu \Delta'$, with μ minimal. Since there may be steps of prediction, we count 2μ. Furthermore, we count the steps of completion between $\mathfrak{j}$ and $\mathfrak{i}$ as for the CYK system. Thus $\mathfrak{d}(\mathfrak{i}, \Gamma \lhd \mathfrak{b}\, \Delta \to \mathfrak{a}, \mathfrak{j}) = \mathfrak{m} + 2\mu + \mathfrak{j} - \mathfrak{i}$.

Consider $\otimes$-prediction of an item $(\mathfrak{i}, \mathfrak{a}\, \mathfrak{b} \triangleleft \to \mathfrak{a} \otimes \mathfrak{b}, \mathfrak{i})$. Let $\Lambda\, \mathfrak{a} \otimes \mathfrak{b}\, w_{\mathfrak{j}+1} \dots w_\mathfrak{l} \Rightarrow^* \mathfrak{c}$ for some $\mathfrak{j}$ and $\mathfrak{l}$ such that $\mathfrak{i} \leqslant \mathfrak{j} \leqslant \mathfrak{l} \leqslant \mathfrak{n}$, $\Lambda \in \mathcal{F}^*$ and $\mathfrak{c} \in \mathcal{F}$. Let us split into: $\Lambda\, \mathfrak{a} \otimes \mathfrak{b}\, \Xi \Rightarrow^\pi \mathfrak{c}$ covering the predictive part and $w_{\mathfrak{j}+1} \dots w_\mathfrak{l} \Rightarrow^\zeta \Xi$ covering the recognition of a valid suffix. Thus, for predicting $(\mathfrak{i}, \mathfrak{a}\, \mathfrak{b} \triangleleft \to \mathfrak{a} \otimes \mathfrak{b}, \mathfrak{i})$ we count $\pi + 2\zeta + \mathfrak{l} - \mathfrak{j}$: the predictive part single, the recognition part double and the portion covered.

Summing up, for $(\mathfrak{i}, \Gamma \lhd \Delta \to \mathfrak{a}, \mathfrak{j})$ we have:

$$\pi + 2\mu + 2\zeta + \mathfrak{j} - \mathfrak{i} + \mathfrak{l} - \mathfrak{j} + |\Gamma| = \pi + 2\mu + 2\zeta + \mathfrak{l} - \mathfrak{i} + |\Gamma|$$

For $(i, \Delta \triangleright \Gamma \to a, j)$ we proceed by symmetry.

Thus we define the following *dfl* for $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$.

Definition 3.20. Deduction length function for $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$.

$$
\begin{aligned}
d(i-1, a \to a, i) &= 0 \\
d(i, \Delta \triangleright \Gamma \to a, j) &= \min\{\pi + 2\mu + 2\zeta + j - k + |\Gamma| \mid w_{i+1} \dots w_j \Rightarrow^{\mu} \Delta, \\
&\qquad\qquad \Xi\, a\, \Lambda \Rightarrow^{\pi} c, \\
&\qquad\qquad w_{k+1} \dots w_i \Rightarrow^{\zeta} \Xi\} \\
d(i, \Gamma \triangleleft \Delta \to a, j) &= \min\{\pi + 2\mu + 2\zeta + l - i + |\Gamma| \mid w_{i+1} \dots w_j \Rightarrow^{\mu} \Delta, \\
&\qquad\qquad \Lambda\, a\, \Xi \Rightarrow^{\pi} c, \\
&\qquad\qquad w_{j+1} \dots w_l \Rightarrow^{\zeta} \Xi\}
\end{aligned}
$$

Proposition 3.10. Completeness of $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$.
$\mathcal{W} \subseteq V(\mathcal{D})$.

Proof. We check that that d from Definition 3.20 is a correct *dfl* for viable items of $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$.

1. Consider $\xi = (i, \Delta \triangleright b \to a, j)$. Let $\Delta \to a/b \in \widetilde{AX}^{*}$ and $w_{i+1} \dots w_j \Rightarrow^{*} \Delta$ be a minimal derivation. Then $\eta = (i, \Delta \to a/b, j) \in \mathcal{W}$ and ξ is derived from η. The predictive component of these items is null. Thus, we have

$$
\begin{aligned}
d(\eta) &= 0 + 2\mu + 0 + j - 0 + 0 \\
d(\xi) &= 0 + 2\mu + 0 + j - 0 + 1 \\
d(\xi) &= d(\eta) + 1
\end{aligned}
$$

2. Consider $\xi = (i, \Delta\, b \triangleright \Gamma \to a, j)$. We shall consider two subcases.

   a) Assume that $\epsilon \Rightarrow^{+} b$. Let $w_{i+1} \dots w_j \Rightarrow^{\mu} \Delta$, $\Xi\, a\, \Lambda \Rightarrow^{\pi} c$ and for some k, $0 \leqslant k \leqslant i$, $w_{k+1} \dots w_i \Rightarrow^{\zeta} \Xi$ with μ, π and ζ minimal. Then $\eta = (i, \Delta \triangleright b\, \Gamma \to a, j) \in \mathcal{W}$ and

$$d(\eta) = \pi + 2\mu + 2\zeta + j - k + |b\, \Gamma|$$

   For ξ, we have $w_{i+1} \dots w_j \Rightarrow^{\mu} \Delta \Rightarrow \Delta\, b$. Hence,

$$d(\xi) = \pi + 2(\mu + 1) + 2\zeta + j - k + |\Gamma| = d(\eta) + 1$$

   b) Let $w_{i+1} \dots w_m \Rightarrow^{\mu} \Delta$, $\Xi a \Lambda \Rightarrow^{\pi} c$ and for some k, $0 \leqslant k \leqslant i$, $w_{k+1} \dots w_i \Rightarrow^{\zeta} \Xi$ with μ, π and ζ minimal. Then $\eta = (i, \Delta \triangleright b\, \Gamma \to a, m) \in \mathcal{W}$ and

$$d(\eta) = \pi + 2\mu + 2\zeta + m - k + |b\, \Gamma|$$

Let $w_{m+1} \dots w_j \Rightarrow^{\mu'} \Delta'$ with μ' minimal and $\Delta' \to b \in \widetilde{AX}^\star$. Then $\theta = (m, \Delta' \to b, j) \in \mathcal{W}$. For θ, we have $\Xi\Delta\, b\, \Gamma\Lambda \Rightarrow \Xi a \Lambda \Rightarrow^{\pi} c$ and $w_{k+1} \dots w_m \Rightarrow^{\zeta+\mu} \Xi\Delta$. Thus,

$$d(\theta) = (\pi + 1) + 2\mu' + 2(\mu + \zeta) + j - k + 0$$

For ξ, we have $w_{i+1} \dots w_j \Rightarrow^{\mu+\mu'} \Delta\Delta' \Rightarrow \Delta b$.

$$d(\xi) = \pi + 2(\mu + \mu' + 1) + 2\zeta + j - k + |\Gamma|$$

Hence $d(\xi) \geqslant d(\theta) + 1$ and $d(\xi) \geqslant d(\eta) + (j - m + 1)$

3. Consider the case of prediction of $\xi = (j, \triangleright a\, b \to a \otimes b, j)$. Suppose $\Delta\, a \otimes b\, \Gamma \to c' \in \widetilde{AX}^\star$, that for some $0 \leqslant i \leqslant j$, $w_{i+1} \dots w_j \Rightarrow^{\mu} \Delta$, that $\Xi\, c'\, \Lambda \Rightarrow^{\pi} c$ and for some $0 \leqslant k \leqslant i$, $w_{k+1} \dots w_i \Rightarrow^{\zeta} \Xi$ with μ, π and ζ minimal. Then $\eta = (i, \Delta \triangleright a \otimes b\, \Gamma \to c', j) \in \mathcal{W}$, ξ is derived from η and

$$d(\eta) = \pi + 2\mu + 2\zeta + j - k + |a \otimes b\, \Gamma|$$

For ξ we have $\Xi\Delta\, a \otimes b\, \Gamma\Lambda \Rightarrow \Xi\, c'\, \Lambda \Rightarrow^{\pi} c$ and $w_{k+1} \dots w_j \Rightarrow^{\mu+\zeta} \Xi\Delta$. Hence,

$$d(\xi) = (\pi + 1) + 2(\mu + \zeta) + j - k + |a\, b|$$

Observe that $|\Gamma| \leqslant 1$. We conclude $d(\xi) \leqslant d(\eta) + 1$.

□

3.6 Approaching Lambek systems

Lambek style categorial grammars are intrinsically richer than Ajdukiewicz–Bar-Hillel categorial grammars. As we said before, the great difference is that while these grammars only compose larger structures from simpler ones (as CF grammars), Lambek grammars may also *decompose* complex structures into simpler ones.

In this chapter, we saw several ways of regimenting the composition rules by means of indices. On the other hand, the introduction rules of Lambek systems

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

do not seem to have such a clear indexed counterpart.

An early attempt of applying chart methods to hypothetical reasoning of Lambek calculi is [König, 1994]. However, as [Hepple, 1999] says, this "method requires rather complicated book-keeping".

In automatic proof search for Lambek style grammars, one often adopts a *cut-free* axiomatization[7] of the underlying logic. Then, the proof of a sequent may consist in reaching the axioms by forward application (that is, *from*

[7]Perhaps it would be more appropriate to say *non-erasing* axiomatization, meaning a system of rules that do not cancel formulas from the premises.

conclusion to premises) of each rule of the calculus[8]. A refinement of this approach is the so called *goal-directed* proof search, proposed in [Hendriks, 1993] for the associative Lambek calculus without product, in [Moortgat, 1997] and in [Andreoli, 1992, 2001] within the broader field of linear logic. 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 start the application of the rules. Secondly, one may immediately notice that in the conclusion-to-premises approach the search space also contains sequents that are not provable. For example, the following is a possible result of applying, conclusion-to-premises, the left rule for $\backslash$ of the sequent calculus[9]:

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

Observe that there is no type invariant (as the balancing constraints of [van Benthem, 1991; Moortgat, 1988]) preventing the occurrence of $\mathfrak{a}/(\mathfrak{b}\backslash\mathfrak{a}) \to \mathfrak{b}$.

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"[10]. As the one of König, Hepple's approach constrains the use of "hypotheticals" through a rather complicate

[8] In [Shieber et al., 1995] this is discussed as the only practical approach. Let us quote from the section "Inadequacy of sequent calculi" (*inadequacy* for chart parsing methods):

> ... the rule used for the hypothetical analysis [...] has the form
>
> $$\frac{\Gamma B \vdash A}{\Gamma \vdash A/B}$$
>
> 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.

[9] The rule we are referring to is usually presented as follows:

$$\frac{\Delta \to \mathfrak{b} \quad \Gamma[\mathfrak{a}] \to \mathfrak{c}}{\Gamma[(\Delta,\ \mathfrak{b}\backslash\mathfrak{a})] \to \mathfrak{c}}$$

[10] [Hepple, 1999].

mechanism of indexing and index sharing. This mechanism allows then the application of standard parsing techniques (viz. Earley style parsing) to Lambek grammars. The two main limitations of this approach are the non-logical character of the first-order compilation and the absence of product formulas. We will define in Chapter 5 a logical method for converting NL grammars into $AB^{\otimes}$ grammars which will overcome these limitations.

Other parsing algorithms for Lambek grammars have been based on *proof nets*. We mention the works of [Morrill, 1996] and [Carpenter and Morrill, 2005] that formulate CYK style parsers for proof nets formulations of the associative and non-associative Lambek calculus.

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[11]. The multi-modal system, on the other hand, is even more complex. As we mentioned at the end of Chapter 2, [Carpenter, 1996] and [Moot, 2002] show that with or without *linear* structural rules (rules that do not erase nor duplicate material), the multi-modal system is undecidable. However, [Moot, 2002] proves also that if the structural rules are linear and *non-expanding* (that is, the succedent of each structural rule contains as many unary connectives as the antecedent), the recognition problem for multi-modal type-logical grammars is P-space complete.

A better situation holds for other fragments of the Lambek calculus. [Savateev, 2006] proves that sequents of the unidirectional product-free associative Lambek calculus can be recognized in cubic time. Instead, [de Groote, 1999] and [de Groote and Lamarche, 2002] prove that two-formula sequents of the 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 [Bulińska, 2006] extend this result to NL enriched with non-logical axioms (and also empty antecedent in the case of Buliśka).

A feature shared by many of the approaches mentioned before is that the recognition process is divided into 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 second part of this book, we will follow this approach in implementing an efficient lexical 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. In Chapter 7, we will also prove that our lexical conversion keeps the size of the resulting grammar within a reasonable size. This will enable us

[11] On the other hand, the parsers of [Hepple, 1999] and [König, 1994] are designed for the *product-free* associative Lambek calculus, whose complexity is still an open problem.

to prove polynomial recognition for non-associative Lambek grammars with product.

3.7 Conclusion

In this chapter, I presented the two most well known parsing systems for CF grammars. My contribution was to extend these systems to AB grammars with product. I presented then a new parsing system, called $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$, which incorporates the most pleasant features of the two previous systems.

We saw how different formulations of a logical system (namely, $\mathsf{AB}^{\otimes}$, $\overline{\mathsf{AB}}^{\otimes}$ and $\widetilde{\mathsf{AB}}^{\otimes}$) may highlight different deduction strategies that can be encoded in a parsing system.

The implementation and complexity of the parsing systems studied in this chapter are the subject of Chapter 4. There we prove that all the parsing systems presented in this chapter can be implemented as cubic time parsing algorithms. We will also see that parsing with $\mathsf{AB}^{\otimes}$ grammars can be more efficient than parsing with context-free grammars, since the abstract inference schemes of $\mathsf{AB}^{\otimes}$ are not dependent on specific components of the grammar.

The reader who is not interested in the implementation of the parsing systems discussed in this chapter may skip the next chapter and go directly to Part III, where we present the methods for applying the parsing systems studied here to NL grammars.

Chapter 4

Implementations

The labeled inference rules of the parsing systems formulated in Chapter 3 specify what item follows from which premises. However, a parsing system does not specify any order in which its rules should be applied nor how the inference process should be iterated. In this chapter, I present some ways of implementing the parsing systems of Chapter 3. The first is the *agenda-driven chart-based* deductive procedure of [Shieber et al., 1995]. I give a functional definition of this procedure that can be straightforwardly applied to the parsing systems of Chapter 3 by just specifying the appropriate rules. Then, I formulate tabular parsing algorithms for the parsing systems that we saw before. This kind of implementation is in fact the most efficient as it results in cubic time recognition methods.

4.1 Agenda-driven chart-based procedure

An agenda-driven chart-based procedure, $\mathcal{AC}$ 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 4.1 is an adaptation of the $\mathcal{AC}$ procedure described in [Shieber et al., 1995].

Algorithm 4.1. *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 covers the whole input string with the start category, for example $(0, \mathsf{s}, \mathfrak{n})$ in $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$, $(0, \mathsf{s}\bullet \to \mathsf{S}', \mathfrak{n})$ in $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ and $(0, \Delta \to \mathsf{s}, \mathfrak{n})$ in $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$, where $\mathfrak{n}$ 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 4.1.

Functional implementation

Algorithm 4.2 below is a functional implementation of the $\mathcal{AC}$ procedure specified in Algorithm 4.1. This implementation is similar to the one in [van Eijck, 2004] who provides a functional implementation of $\mathcal{CF}_{\mathtt{Earley}}$. 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 [Cormen et al., 1990] and [Adams, 1993]. Roughly, red-black trees represent sets as trees whose nodes are *ordered*. For instance, what we write as $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}, \mathsf{f}, \mathsf{g}\}$ is interpreted as the tree:

$$\frac{\underline{\mathsf{a}\quad \mathsf{c}} \qquad \underline{\mathsf{e}\quad \mathsf{g}}}{\underline{\mathsf{b}\qquad\qquad \mathsf{f}}}$$
$$\mathsf{d}$$

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 $\mathfrak{a}$, which we write $\{\mathfrak{a}\}$, has been defined. We write $\mathsf{X}\{\mathsf{x}\}$ for a set X with a distinguished element x, so that X results from $\mathsf{X}\{\mathsf{x}\}$ 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.

Finally, we assume that the rules of the parsing system are *functions* whose inputs are the premises and whose output is the conclusion.

Algorithm 4.2. *Functional implementation of the $\mathcal{AC}$ procedure.*

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 $\mathsf{Z} := \emptyset$.

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

The function

$$\texttt{exhaust-agenda} :: (\mathit{Chart}, \mathit{Agenda}) \to (\mathit{Chart}, \mathit{Agenda})$$

is defined in Figure 4.1.

If, after termination of `exhaust-agenda`, the goal item is in Z, then the goal is proved, otherwise it is not.

$$
\begin{array}{lll}
\texttt{exhaust-agenda} & (\mathsf{Z}, \emptyset) & = (\mathsf{Z}, \emptyset) \\
\texttt{exhaust-agenda} & (\mathsf{Z}, \mathsf{Y}\{\mathsf{y}\}) & = \\
 & \text{if } \mathsf{y} \in \mathsf{Z} & \\
 & \text{then } \texttt{exhaust-agenda}\ (\mathsf{Z}, \mathsf{Y}) & \\
 & \text{else } \texttt{exhaust-agenda}\ (\mathsf{Z}', \mathsf{Y}') & \\
 & \quad \text{where} & \\
\text{a)} & \quad \mathsf{C} & := \{\ \mathsf{c} \mid \mathsf{z} \in \mathsf{Z},\ \rho \in \mathcal{R},\ \mathsf{c} \in \rho(\mathsf{y})(\mathsf{z})\ \} \\
\text{b)} & \quad \mathsf{Y}' & := \mathsf{Y} \cup \mathsf{C} \\
\text{c)} & \quad \mathsf{Z}' & := \{\mathsf{y}\} \cup \mathsf{Z}
\end{array}
$$

Figure 4.1: Functional implementation of the agenda-driven chart-based procedure.

Step by step analysis

The function `exhaust-agenda` calculates all the valid items of a given parsing system. The constructs `type` $\mathsf{x} := \mathsf{y}$ 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 variables 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 direct 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 $\mathsf{z} \in \mathsf{Z}$ on the basis of the rules in $\mathcal{R}$, in line a)[1]. 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 chart 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 $\mathcal{AB}^{\otimes}_{\mathsf{CYK}}$.

[1]Observe that the rules in $\mathcal{R}$ are functions into $\{\mathcal{I}\}$. This is because we return $\emptyset$ if a rule does not apply to some premises.

Example 4.1. Implementation of rules for $\mathcal{AB}^{\otimes}_{\mathtt{CYK}} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$.

Let a set of formulas Σ be calculated from the axioms as in Definition 3.5 on page 51. Then $\mathcal{R}$ consists of rules $\mathtt{el}$ and $\mathtt{p}\ \Sigma$ defined below.

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

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

$\mathtt{el}, \mathtt{e_i} :: \mathcal{I} \to \mathcal{I} \to \{\mathcal{I}\}$

$$\begin{aligned}
\mathtt{el}\ \mathtt{x}\ \mathtt{y} &= \textstyle\bigcup_{1 \leqslant \mathtt{i} \leqslant 4} \mathtt{e_i}\ \mathtt{x}\ \mathtt{y} \\
\mathtt{e}_1\ (\mathtt{i}, \mathtt{b}, \mathtt{k})\ (\mathtt{k}', \mathtt{b}' \backslash \mathtt{a}, \mathtt{j}) &= \text{if } \mathtt{k} \equiv \mathtt{k}' \ \&\ \mathtt{b} \equiv \mathtt{b}' \text{ then } \{(\mathtt{i}, \mathtt{a}, \mathtt{j})\} \text{ else } \emptyset \\
\mathtt{e}_2\ (\mathtt{i}, \mathtt{a}/\mathtt{b}, \mathtt{k})\ (\mathtt{k}', \mathtt{b}', \mathtt{j}) &= \text{if } \mathtt{k} \equiv \mathtt{k}' \ \&\ \mathtt{b} \equiv \mathtt{b}' \text{ then } \{(\mathtt{i}, \mathtt{a}, \mathtt{j})\} \text{ else } \emptyset \\
\mathtt{e}_3\ (\mathtt{k}', \mathtt{b}' \backslash \mathtt{a}, \mathtt{j})\ (\mathtt{i}, \mathtt{b}, \mathtt{k}) &= \text{if } \mathtt{k} \equiv \mathtt{k}' \ \&\ \mathtt{b} \equiv \mathtt{b}' \text{ then } \{(\mathtt{i}, \mathtt{a}, \mathtt{j})\} \text{ else } \emptyset \\
\mathtt{e}_4\ (\mathtt{k}', \mathtt{b}', \mathtt{j})\ (\mathtt{i}, \mathtt{a}/\mathtt{b}, \mathtt{k}) &= \text{if } \mathtt{k} \equiv \mathtt{k}' \ \&\ \mathtt{b} \equiv \mathtt{b}' \text{ then } \{(\mathtt{i}, \mathtt{a}, \mathtt{j})\} \text{ else } \emptyset \\
\mathtt{e_i}\ _\ _ &= \emptyset,\ 1 \leqslant \mathtt{i} \leqslant 4
\end{aligned}$$

Observe that $\mathtt{e_i},\ 1 \leqslant \mathtt{i} \leqslant 4$ exhausts the possible occurrences of patterns for the premises. Hence $\mathtt{el}$ is a complete definition of the cancellation rules of $\mathcal{AB}^{\otimes}_{\mathtt{CYK}}$.

The application of Algorithm 4.2 to the $\mathcal{CF}_{\mathtt{CYK}}$ system is straightforward. The rules may look like the following.

Example 4.2. Implementation of rules for $\mathcal{CF}_{\mathtt{CYK}} = \langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$.

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

$\mathtt{cfInf} :: \mathcal{P} \to \mathcal{I} \to \mathcal{I} \to \{\mathcal{I}\}$

$$\begin{aligned}
\mathtt{cfInf}\quad \mathcal{P}\quad (\mathtt{i}, \mathtt{a}, \mathtt{k})\quad (\mathtt{l}, \mathtt{b}, \mathtt{j}) \ = \\
\{\ (\mathtt{i}, \mathtt{c}, \mathtt{j}) \mid \mathtt{l} \equiv \mathtt{k},\ (\mathtt{a}'\ \mathtt{b}' \to \mathtt{c}) \in \mathcal{P},\ \mathtt{a} \equiv \mathtt{a}',\ \mathtt{b} \equiv \mathtt{b}'\ \} \\
\cup \\
\{\ (\mathtt{l}, \mathtt{c}, \mathtt{k}) \mid \mathtt{j} \equiv \mathtt{i},\ (\mathtt{b}'\ \mathtt{a}' \to \mathtt{c}) \in \mathcal{P},\ \mathtt{a} \equiv \mathtt{a}',\ \mathtt{b} \equiv \mathtt{b}'\ \}
\end{aligned}$$

The procedure can immediately be extended to the $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ and $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ systems. We omit the details about the straightforward implementation of the rules.

4.2 Tabular parsing

Though very simple, the procedure defined in Algorithm 4.2 is not an optimal solution for practical parsing. Its main limitation is that at each iteration, we should check whether the trigger item had already been computed.

The algorithms that we are going to see in the following sections 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 at any point of the computation.

For example, in the case of the CYK parser, if we are analyzing a string $w_1 \dots w_n$ 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 $w_{j+1} \dots w_i \Rightarrow^* c$. Therefore, to test whether $w_1 \dots w_n$ belongs to $L_t(G)$ according to the CYK algorithm, we compute the parse table for $w_1 \dots w_n$ and check whether the start symbol of G is in $t_{(0,n)}$.

4.3 Tabular CYK algorithm

We present the CYK method of table construction. The algorithm works for CF grammars in CNF without ϵ-productions or Ajdukiewicz–Bar-Hillel 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 $\bigtriangledown$ with the specific operations proper to a CF or to an Ajdukiewicz–Bar-Hillel grammar. Similarly, $\mathtt{Start}$ is a variable over the start symbol of the input grammar.

Algorithm 4.3.

Input: A lexicalized grammar $G = \langle V_t, \mathtt{Start}, Lex, D \rangle$ 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: see Figure 4.2 on the following page.

Given an appropriate instantiation for $\bigtriangledown$, the algorithm recognizes a string $w_1 \dots w_n$ if and only if $\mathtt{Start} \in t_{(0,n)}$. We provide the instantiation of $\bigtriangledown$ for CF first and for $AB^{\otimes}$ later.

Definition 4.1. Instantiation of $\bigtriangledown$ for CF grammars.

If the grammar input of Algorithm 4.3 is a CF grammar G, then $\bigtriangledown := \circledast_{\mathcal{P}}$, where $\mathcal{P}$ is the set of the productions of G and $(\circledast_{\mathcal{P}}) :: \{\mathcal{F}\} \to \{\mathcal{F}\} \to \{\mathcal{F}\}$ is defined as follows:

$$X \circledast_{\mathcal{P}} Y = \{\ A \mid B \in X,\ C \in Y,\ B\ C \to A \in \mathcal{P}\ \}$$

We call $\mathcal{CYK}_{CF}$ the algorithm resulting from Algorithm 4.3 by instantiating $\bigtriangledown$ with $\circledast_{\mathcal{P}}$.

```
begin
loop_1 :  for i := 1 to n do:
              t_(i-1,i) = { a | w_i → a ∈ Lex }
loop_2 :  for i := 2 to n do:
              for j := i − 2 down-to 0 do:
                  for k := j + 1 to i − 1 do:
                  t_(j,i) := t_(j,i) ∪ (t_(j,k) ∇ t_(k,i))
end.
```

Figure 4.2: Table construction method of the CYK algorithm.

Example 4.3. We examine Algorithm 4.3 applied to the string *every man loves a woman* and to grammar G_3' in Example 3.2.

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* parser for another graphical representation of the deduction to which it gives rise. The chart for Example 4.3 is presented in Figure 4.3 on the next page. The vertices of the graph indicate the positions of the words in the input string. The edge label indicates the grammatical category of the subexpression between the vertices linked by the edge.

Example 4.4. Let us consider now a second example from grammar G_1' which consists of the following rewriting rules.

$$\begin{aligned} O\,S' \mid S\,S \mid O\,C &\to S \\ S\,C &\to S' \\ (&\to O \\ ) &\to C \end{aligned}$$

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

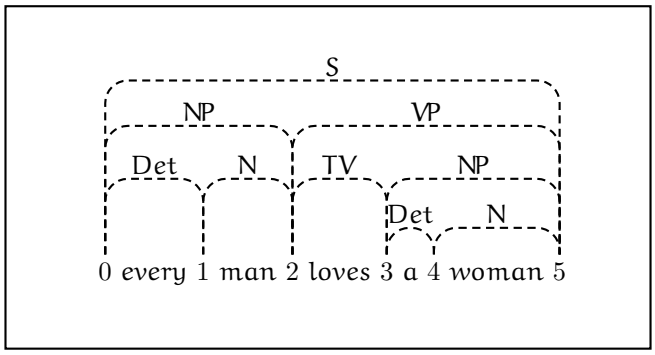


Figure 4.3: Chart for Example 4.3.

Algorithm $\mathcal{CYK}_{\mathsf{CF}}$ 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 $\mathcal{CYK}_{\mathsf{CF}}$ is known and we refer the reader to [Aho and Ullman, 1972] for the details of the proofs of soundness and completeness. However, we sketch the proof of completeness below for the case of $\mathsf{AB}^{\otimes}$ grammars.

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

Definition 4.2. Instantiation of $\bigtriangledown$ for $\mathsf{AB}^{\otimes}$ grammars.

If the grammar input of Algorithm 4.3 is an $\mathsf{AB}^{\otimes}$ grammar G, then $\bigtriangledown := \circledast_{\Sigma}$, where Σ is a set of formulas obtained from the lexical categories assigned to the input string by function δ^{-} in Definition 3.4 on page 51 and $(\circledast_{\Sigma}) :: \{\mathcal{F}\} \to \{\mathcal{F}\} \to \{\mathcal{F}\}$ is defined as follows:

$$
\begin{aligned}
X \circledast_{\Sigma} Y \quad = \quad & \{\, c \mid c/b \in X,\ b \in Y \,\} \\
& \cup \\
& \{\, c \mid b \in X,\ b\backslash c \in Y \,\} \\
& \cup \\
& \{\, a \otimes b \mid a \in X,\ b \in Y,\ a \otimes b \in \Sigma \,\}
\end{aligned}
$$

We call $\mathcal{CYK}_{AB\otimes}$ the algorithm resulting from Algorithm 4.3 by instantiating $\bigtriangledown$ with $*_\Sigma$.

We present an example of application of $\mathcal{CYK}_{AB\otimes}$.

Example 4.5. Application of $\mathcal{CYK}_{AB\otimes}$ to the string $aacbb$ and to grammar A_7 for the language $a^n c b^n$ whose lexicon is

$$\begin{aligned} a &\to s/(s \otimes b) \\ c &\to s \\ b &\to b \end{aligned}$$

Parse table:

1	2	3	4	5	i / j
$s/(s \otimes b)$				s	0
	$s/(s \otimes b)$		s	$s \otimes b$	1
		s	$s \otimes b$		2
			b		3
				b	4

Figure 4.4 shows the corresponding chart.

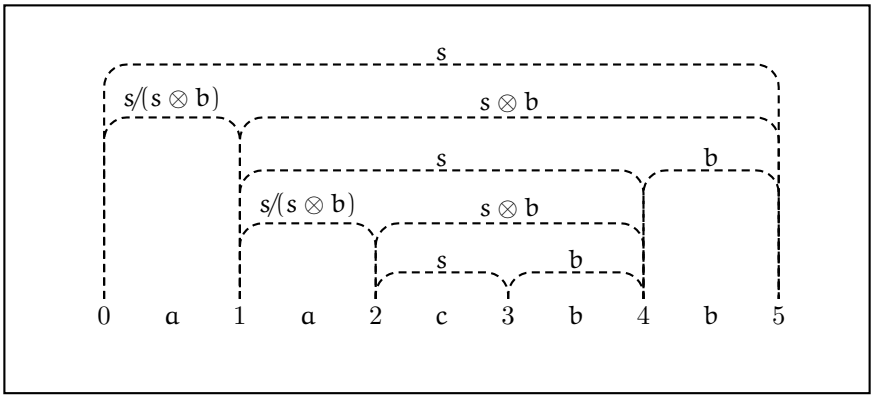


Figure 4.4: Chart for the string $aacbb$ in grammar A_7.

Let us consider a second example involving ambiguity.

Example 4.6. We apply $\mathcal{CYK}_{AB\otimes}$ to the grammar PL in Example 3.6 on page 52 and to the string $\neg p \wedge q$.

Parse table:

1	2	3	4	i / j
s/s	s		s	0
	s		s	1
		(s\s)/s	s\s	2
			s	3

Observe that the *parse* table encodes the two possible analyses of the input string, in fact $s \in t_{(0,4)}$ can be obtained either from $t_{(0,2)} \circledast t_{(2,4)}$ or from $t_{(0,1)} \circledast t_{(1,4)}$ (we omitted the Σ parameter, as in this case it is empty).

The correctness of $\mathcal{CYK}_{AB\otimes}$ is stated below.

Proposition 4.1. If algorithm $\mathcal{CYK}_{AB\otimes}$ is applied to a lexicalized grammar G and to a string $w_1 \dots w_n$, then upon termination,

$$c \in t_{(j,i)} \quad \text{iff} \quad w_{j+1} \dots w_i \Rightarrow^* c$$

Proof. The procedure in Algorithm 4.3 is a standard implementation of the CYK table construction method whose correctness is proved in [Aho and Ullman, 1972] for CF grammars. The main observation is that at the point in which $t_{(j,i)}$ is computed, the cells in row j to the left of $t_{(j,i)}$ (that is the cells $t_{(j,k)}$ with $j < k < i$) are already completed as well as the cells in column i below $t_{(j,i)}$ (that is the cells $t_{(k,i)}$ with $j < k < i$). Soundness is trivial. For completeness, one works by induction on i. The base case (when $i \equiv j + 1$) follows from the initialization of the algorithm in $\mathtt{loop}_1$. If $w_{j+1} \dots w_i \Rightarrow^* c$ and $i > j + 1$, then for some k such that $j < k < i$, $w_{j+1} \dots w_k \Rightarrow^* c/a$ and $w_{k+1} \dots w_j \Rightarrow^* a$ (or similarly for the symmetric slash and the product). By IH, $c/a \in t_{(j,k)}$ and $a \in t_{(k,i)}$. Hence $c \in t_{(j,k)} \circledast_\Sigma t_{(k,i)} \subseteq t_{(j,i)}$. □

The complexity of the CYK algorithm is calculated in terms of the *elementary operations* required to build the table. The exact definition of elementary operation can be found in [Aho and Ullman, 1972]. However, every single operation performed in each of the clauses of Algorithm 4.3 can be considered an elementary operation (apart from $\bigtriangledown$, see Remark 4.1 on the following page). We state the main result concerning the complexity of Algorithm 4.3.

Proposition 4.2. Let n be the length of the input string. Then Algorithm 4.3 requires $O(n^3)$ elementary operations to compute $t_{(j,i)}$ for all i and j.

Proof. See [Aho and Ullman, 1972]. □

The calculation is based on the three embedded cycles which define the main loop of the algorithm, namely $\mathtt{loop}_2$. In the most external one, i ranges between 2 and n, which means that the cycle for j is executed $n - 1$ times. In turn, j ranges between 0 and $i - 2$. Thus the k cycle is executed $i - 1$ times. Finally, k ranges between $j + 1$ and $i - 1$, hence in the worse case between 1 and $n - 1$. Therefore Algorithm 4.3 performs $O(n^3)$ elementary operations.

Remark 4.1. The complexity result expressed in Proposition 4.2 concerns the variation of time in relation to the length of the input string. Another parameter which is relevant in the calculation of the complexity of the CYK algorithm for CF grammars is the complexity of the operation $(\circledast_{\mathcal{P}})$, see [Nederhof and Satta, 2004]. Clearly, this depends on the size of the input grammar, expressed in terms of the number of its productions: $|G| = O(|\mathcal{P}|)$. Thus, the complexity of Algorithm 4.3 is expressed as $O(|G|n^3)$. We shall remark that conversion to CNF may square the size of the original grammar. As in practical applications, $|G|$ is much bigger than the length of the input string, squaring it may affect drastically the performance of the CYK algorithm.

Concerning $\mathcal{CYK}_{AB^{\otimes}}$, we observe that the grammar takes no part in the table completion procedure in $\mathtt{loop}_2$. Instead, only the set Σ of product subformulas is used by the rule in Definition 4.2. Hence for $\mathtt{loop}_2$ of $\mathcal{CYK}_{AB^{\otimes}}$ we have a complexity of $O(|\Sigma|n^3)$ and for the product-free variant of the algorithm $O(n^3)$. Furthermore, $AB^{\otimes}$ grammars, without ϵ-assignments are already in the required normal form.

$\mathcal{CYK}_{AB^{\otimes}}$ can easily be extended to AAB grammars, by simply adding the associative cancellation rules. One extends the definition of $\circledast$ with the following clauses

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

We observe also that 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[2] can be parsed in time $O(n^6)$.

4.4 The Earley algorithm

In this section and in the next one, I give the implementation of the Earley algorithm for CF and $AB^{\otimes}$. I will follow the formulation of [Aho and Ullman, 1972]. With some abuse of terminology, we will continue to call *items* the objects on which the algorithms in the following sections work, although such objects are distinct from those of Chapter 3.

Let a CF grammar $G = \langle V_t, S, \mathcal{F}, \mathcal{P}\rangle$ and a string $w_1 \dots w_n$ be given. Let $V = V_t \cup \mathcal{F}$. Earley items are object of the form

$$(X_1 \dots X_k \bullet X_{k+1} \dots X_m \to A, i)$$

[2]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 this system and its complexity properties.

where $X_1 \dots X_m \to A \in AX$, $\bullet$ is a symbol not in V, $0 \leqslant k \leqslant m$ and $0 \leqslant i \leqslant n$.

The algorithm works by constructing, for each integer j, $0 \leqslant j \leqslant n$, a list of items I_j such that $(\Delta \bullet \Gamma \to A, i) \in I_j$ with $0 \leqslant i \leqslant j$, if and only if for some Ξ and Λ,

$$w_{i+1} \dots w_j \Rightarrow^* \Delta,$$
$$\Xi A \Lambda \Rightarrow^* S,$$
$$w_1 \dots w_i \Rightarrow^* \Xi$$

The list $I_0 \dots I_n$ is called the *parse list* for the input string $w_1 \dots w_n$. One has that $w_1 \dots w_n \in L(G)$ if and only if $(\Delta\bullet \to S, 0) \in I_n$.

4.4.1 The Earley algorithm for CF grammars

In [Aho and Ullman, 1972], the Earley algorithm for CF grammars is presented as follows.

Algorithm 4.4. *Earley's parsing algorithm.*

Input. A CF grammar $G = \langle V_t, S, \mathcal{F}, \mathcal{P} \rangle$ and a string $w_1 \dots w_n$.

Output. The parse list $I_0 \dots I_n$.

Method. See Figure 4.5.

We first construct I_0 according to steps 1, 2 and 3.

1. For all $\Delta \to S \in \mathcal{P}$, add $(\bullet\Delta \to S, 0)$ to I_0.

Now perform steps 2 and 3 until no new items can be added to I_0.

2. If $(\Gamma\bullet \to B, 0) \in I_0$, add $(\Lambda\, B\bullet\Delta \to C, 0)$ to I_0, for all $(\Lambda\bullet B\, \Delta \to C, 0) \in I_0$.
3. If $(\Lambda \bullet B\, \Delta \to C, 0) \in I_0$, for all the $\Gamma \to B \in \mathcal{P}$ add $(\bullet\Gamma \to B, 0)$ to I_0 (provided this item is not already in I_0).

We now construct I_j having constructed $I_0 \dots I_{j-1}$.

4. For each $(\Delta \bullet w_j\, \Lambda \to A, i) \in I_{j-1}$, add $(\Delta\, w_j \bullet \Lambda \to A, i)$ to I_j

Now perform steps 5 and 6 until no new items can be added to I_j.

5. Let $(\Gamma\bullet \to B, i) \in I_j$. For each item $(\Lambda \bullet B\, \Delta \to C, k) \in I_i$, add $(\Lambda\, B \bullet \Delta \to C, k) \in I_j$
6. If $(\Lambda \bullet B\, \Delta \to C, i) \in I_j$, for all the $\Gamma \to B \in \mathcal{P}$ add $(\bullet\Gamma \to B, j)$ to I_j.

Figure 4.5: The Earley parse list construction method for CF grammars.

Example 4.7. In Figure 4.6 on the following page, we show the parse list resulting by application of Algorithm 4.4 to the grammar of balanced brackets whose productions are

$$\epsilon \mid [S]S \to S$$

and to the string [[]][].

I_0	I_1	I_2
$(\bullet[S]S \to S, 0)$	$([\bullet S]S \to S, 0)$	$([\bullet S]S \to S, 1)$
$(\bullet \to S, 0)$	$(\bullet[S]S \to S, 1)$	$(\bullet[S]S \to S, 2)$
	$(\bullet \to S, 1)$	$(\bullet \to S, 2)$
	$([S\bullet]S \to S, 0)$	$([S\bullet]S \to S, 1)$
I_3	I_4	I_5
$([S] \bullet S \to S, 1)$	$([S] \bullet S \to S, 0)$	$([\bullet S]S \to S, 4)$
$(\bullet[S]S \to S, 3)$	$(\bullet[S]S \to S, 4)$	$(\bullet[S]S \to S, 5)$
$(\bullet \to S, 3)$	$(\bullet \to S, 4)$	$(\bullet \to S, 5)$
$([S]S\bullet \to S, 1)$	$([S]S\bullet \to S, 0)$	$([S\bullet]S \to S, 4)$
$([S\bullet]S \to S, 0)$		
	I_6	
	$([S] \bullet S \to S, 4)$	
	$(\bullet[S]S \to S, 6)$	
	$(\bullet \to S, 6)$	
	$([S]S\bullet \to S, 4)$	
	$([S]S\bullet \to S, 0)$	

Figure 4.6: Parse list for the string [[]][].

Correctness and some computational properties of Algorithm 4.4 are stated below. The proofs of these statements can be found in [Aho and Ullman, 1972].

Proposition 4.3. Properties of Algorithm 4.4.

1. If parse lists are constructed as in Algorithm 4.4, then $(\Delta \bullet \Gamma \to A, i) \in I_j$ if and only if $w_{i+1} \dots w_j \Rightarrow^* \Delta$ and, moreover, for some $\Lambda \in V^*$, $w_1 \dots w_i A \Lambda \Rightarrow^* S$.

2. If the underlying grammar is unambiguous, then when executing Algorithm 4.4 we attempt to add an item $(\Delta \bullet \Gamma \to A, i)$ to list I_j at most once if $\Delta \neq \epsilon$.

3. If the underlying grammar is unambiguous, then Algorithm 4.4 can be executed in $O(n^2)$ operations when the input is of length n.

4. In all cases, Algorithm 4.4 can be executed in $O(n^3)$ operations when the input is of length n.

Proof. See [Aho and Ullman, 1972]. □

[Aho and Ullman, 1972] also provide an algorithm for the construction of a *right parse* from the parse lists, if one exists, and prove that this can be done in $O(n^2)$ elementary operations.

4.4.2 The Early algorithm for $\overline{AB}^{\otimes}$

We present the Early parser for $\overline{AB}^{\otimes}$ in the style of [Aho and Ullman, 1972]. Observe that the algorithm implements the rules of the contracted $\mathcal{AB}^{\otimes}_{\mathtt{Earley}}$ system from Definition 3.16 on page 65.

Algorithm 4.5. *Earley's parsing algorithm for* $\overline{AB}^{\otimes}$.

Input. An $\overline{AB}^{\otimes}$ grammar G and a string $w_1 \dots w_n$.

Let Σ be the set containing all the $\delta^*(a_i)$ such that $w_i \to a_i \in \mathsf{Lex}$, where δ^* has been given in Definition 3.12 on page 59.

Output. The parse list $I_0 \dots I_n$.

Method. See Figure 4.7 on the next page.

The input string $w_1 \dots w_n$ is accepted if and only if $(s\bullet \to S', 0) \in I_n$. Let us examine an example application of Algorithm 4.5.

Example 4.8. In Figure 4.8 on page 91, we show the parse list resulting by application of Algorithm 4.5 to the string [[]][] and to the $\overline{AB}^{\otimes}$ grammar for the language balanced brackets whose assignments are

$$\begin{aligned} \epsilon &\to s \\ [\ &\to s/(c \otimes s)/s \\ ]\ &\to c \end{aligned}$$

We have $\Sigma = \{s/(c \otimes s)/s, s/(c \otimes s)\}$

One can verify that Algorithm 4.5 satisfies all properties stated in Proposition 4.3 for Algorithm 4.4.

Proposition 4.4. Properties of Algorithm 4.5.

1. If parse lists are constructed as in Algorithm 4.5, then $(\Delta \bullet \Gamma \to a, i) \in I_j$ if and only if $w_{i+1} \dots w_j \Rightarrow^* \Delta$ and, moreover, for some $\Lambda \in V^*$, $w_1 \dots w_i a \Lambda \Rightarrow^* s \Rightarrow S'$.
2. If the underlying grammar is unambiguous, then when executing Algorithm 4.5 we attempt to add an item $(\Delta \bullet \Gamma \to a, i)$ to list I_j at most once if $\Delta \neq \epsilon$.
3. If the underlying grammar is unambiguous, then Algorithm 4.5 can be executed in $O(n^2)$ operations when the input is of length n.
4. In all cases, Algorithm 4.5 can be executed in $O(|\Sigma| n^3)$ operations.

We first construct I_0 according to steps 1, 2 and 3.

1. Add $(\bullet \mathsf{s} \rightarrow \mathsf{S}', 0)$ to I_0.

Now perform steps 2 and 3 until no new items can be added to I_0.

2. If $(\Lambda \bullet \mathsf{b}\, \Delta \rightarrow \mathfrak{a}, 0) \in I_0$, then
   (a) for some formula x,
      - if $\mathsf{x} \backslash \mathsf{b} \in \Sigma$, then add $(\bullet \mathsf{x}\ \mathsf{x} \backslash \mathsf{b} \rightarrow \mathsf{b}, 0)$ to I_0,
      - if $\mathsf{b} / \mathsf{x} \in \Sigma$, then add $(\bullet \mathsf{b} / \mathsf{x}\ \mathsf{x} \rightarrow \mathsf{b}, 0)$ to I_0,

   (b) if $\mathsf{b} \equiv \mathsf{b}' \otimes \mathsf{b}''$, then add $(\bullet \mathsf{b}'\ \mathsf{b}'' \rightarrow \mathsf{b}, 0)$ to I_0.
   (c) if $\epsilon \Rightarrow^+ \mathsf{b}$, then add $(\Lambda\, \mathsf{b} \bullet \Delta \rightarrow \mathfrak{a}, 0)$ to I_0.
3. If $(\Gamma \bullet \rightarrow \mathsf{b}, 0) \in I_0$, then add $(\Lambda\, \mathsf{b} \bullet \Delta \rightarrow \mathfrak{a}, 0)$ to I_0, for all $(\Lambda \bullet \mathsf{b}\, \Delta \rightarrow \mathfrak{a}, 0) \in I_0$.

We now construct I_j having constructed $I_0 \ldots I_{j-1}$.

4. For each $(\Delta \bullet \mathsf{b}\, \Lambda \rightarrow \mathfrak{a}, i) \in I_{j-1}$, if $w_j \rightarrow \mathsf{b} \in \mathsf{Lex}$, add $(\Delta\, \mathsf{b} \bullet \Lambda \rightarrow \mathfrak{a}, i)$ to I_j.

Now perform steps 5 and 6 until no new items can be added to I_j.

5. Let $(\Gamma \bullet \rightarrow \mathsf{b}, i) \in I_j$. For each item $(\Lambda \bullet \mathsf{b}\, \Delta \rightarrow \mathfrak{a}, k) \in I_i$, add $(\Lambda\, \mathsf{b} \bullet \Delta \rightarrow \mathfrak{a}, k)$ to I_j.
6. If $(\Lambda \bullet \mathsf{b}\, \Delta \rightarrow \mathfrak{a}, i) \in I_j$, then
   (a) for some formula x,
      - if $\mathsf{x} \backslash \mathsf{b} \in \Sigma$, then add $(\bullet \mathsf{x}\ \mathsf{x} \backslash \mathsf{b} \rightarrow \mathsf{b}, j)$ to I_j,
      - if $\mathsf{b} / \mathsf{x} \in \Sigma$, then add $(\bullet \mathsf{b} / \mathsf{x}\ \mathsf{x} \rightarrow \mathsf{b}, j)$ to I_j,

   (b) if $\mathsf{b} \equiv \mathsf{b}' \otimes \mathsf{b}''$, then add $(\bullet \mathsf{b}'\ \mathsf{b}'' \rightarrow \mathsf{b}, j)$ to I_j.
   (c) if $\epsilon \Rightarrow^+ \mathsf{b}$, then add $(\Lambda\, \mathsf{b} \bullet \Delta \rightarrow \mathfrak{a}, i)$ to I_j.

Figure 4.7: The Earley algorithm for $\overline{\mathsf{AB}}^{\otimes}$ grammars.

4.5 Conclusion

In this chapter, I have formulated parsing algorithms for the parsing systems defined in Chapter 3. The tabular algorithms were simple adaptations of well known context-free parsing procedures. The problem of a tabular algorithm for the system $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ remains open. However, the chart-based agenda driven procedure can easily be applied to this parsing system.

I_0

$(\bullet s \to S', 0)$
$(s\bullet \to S', 0)$
$(\bullet s/(c \otimes s)\ c \otimes s \to s, 0)$
$(\bullet s/(c \otimes s)/s\ s \to s/(c \otimes s), 0)$

I_1

$(s/(c \otimes s)/s \bullet s \to s/(c \otimes s), 0)$
$(s/(c \otimes s)/s\ s\bullet \to s/(c \otimes s), 0)$
$(\bullet s/(c \otimes s)\ c \otimes s \to s, 1)$
$(\bullet s/(c \otimes s)/s\ s \to s/(c \otimes s), 1)$
$(s/(c \otimes s) \bullet c \otimes s \to s, 0)$
$(\bullet c\ s \to c \otimes s, 1)$

I_2

$(s/(c \otimes s)/s \bullet s \to s/(c \otimes s), 1)$
$(s/(c \otimes s)/s\ s\bullet \to s/(c \otimes s), 1)$
$(\bullet s/(c \otimes s)\ c \otimes s \to s, 2)$
$(\bullet s/(c \otimes s)/s\ s \to s/(c \otimes s), 2)$
$(s/(c \otimes s) \bullet c \otimes s \to s, 1)$
$(\bullet c\ s \to c \otimes s, 2)$

I_3

$(c \bullet s \to c \otimes s, 2)$
$(c\ s\bullet \to c \otimes s, 2)$
$(s/(c \otimes s)\ c \otimes s\bullet \to s, 1)$
$(\bullet s/(c \otimes s)\ c \otimes s \to s, 3)$
$(\bullet s/(c \otimes s)/s\ s \to s/(c \otimes s), 3)$
$(s/(c \otimes s)/s\ s\bullet \to s/(c \otimes s), 0)$
$(s/(c \otimes s) \bullet c \otimes s \to s, 0)$
$(\bullet c\ s \to c \otimes s, 3)$

I_4

$(c \bullet s \to c \otimes s, 3)$
$(c\ s\bullet \to c \otimes s, 3)$
$(\bullet s/(c \otimes s)\ c \otimes s \to s, 4)$
$(\bullet s/(c \otimes s)/s\ s \to s/(c \otimes s), 4)$
$(s/(c \otimes s)\ c \otimes s\bullet \to s, 0)$
$(s\bullet \to S', 0)$

I_5

$(s/(c \otimes s)/s \bullet s \to s/(c \otimes s), 4)$
$(\bullet s/(c \otimes s)\ c \otimes s \to s, 5)$
$(\bullet s/(c \otimes s)/s\ s \to s/(c \otimes s), 5)$
$(s/(c \otimes s)/s\ s\bullet \to s/(c \otimes s), 4)$
$(s/(c \otimes s) \bullet c \otimes s \to s, 4)$
$(\bullet c\ s \to c \otimes s, 5)$

I_6

$(c \bullet s \to c \otimes s, 5)$
$(c\ s\bullet \to c \otimes s, 5)$
$(\bullet s/(c \otimes s)\ c \otimes s \to s, 6)$
$(\bullet s/(c \otimes s)/s\ s \to s/(c \otimes s), 6)$
$(s/(c \otimes s)\ c \otimes s\bullet \to s, 4)$
$(c\ s\bullet \to c \otimes s, 3)$
$(s/(c \otimes s)\ c \otimes s\bullet \to s, 0)$
$(s\bullet \to S', 0)$

Figure 4.8: Parse list for the string [[]][].

Part II

The Non-associative Lambek Calculus

Chapter 5

Normal Derivations in NL

In this chapter, I present a recognition method for sequents of the non-associative Lambek calculus based on the construction of *normal derivations*.

We will start by restricting the discussion to *two-formula* sequents. The problem we are going to deal with can be stated as follows.

- Given a sequent $a \to c$, how do we prove whether $\vdash_{NL} a \to c$?

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. The problem we are going to address here is substantially different from the one we addressed in Chapters 3 and 4. In the case of CF or AB grammars, recognition of a structured sequent is a trivial task. However, this is not the case for structured sequents of NL: even if the structure is given, recognition of NL sequents can be computationally more demanding than CF parsing.

In fact, the application of the rules of the calculus is non-deterministic, and the different options cannot easily be discarded, 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, since embedded formulas may have to be reduced before, in order to make other external patterns available for reduction. As a sequent of NL may have an exponential number of *readings*

(see Chapter 6), we may easily end up with storing an exponential number of intermediate results with severe consequences for the efficiency of the proof search.

In Section 5.3.1, I formulate a new method for automatic construction of derivations of two-formula sequents. This method is based on the definition of two simple recursive functions which can be immediately translated into a functional or a logical program. The recognition procedure works *bottom-up*, an aspect that distinguishes it from most theorem provers for logical calculi, based on non-erasing rules (like the rules of the cut-free sequent calculus) and top-down proof search. Besides its operations are always *simplifying* and *goal oriented*: two aspects that guarantee termination and limit the non-determinism of the rules. We interpret the task of proving a sequent $\mathsf{a} \rightarrow \mathsf{c}$ 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 procedure. I will define two forms of simplification (or *contraction*), one from left to right, which I call *reduction* and one from right to left, which I call *expansion*. The key intuition behind this work is that patterns called expanding, such as lifting or co-application, are also simplifying, although the direction of the simplification is the reverse of the arrow symbol.

In order to prove the correctness of the recognition method, we will 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]. Our recognition method shares many features with Kandulski's and Buszkowski's method of normal derivation construction. However, while in the approach of these authors there can be several equivalent normal derivations, the derivations constructed in our system are free from spurious ambiguity, as we will prove in Chapter 6.

The system that I am going to design applies a technique similar 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 makes it possible to transform the computation of an NL grammar into the computation of an Ajdukiewicz–Bar-Hillel grammar *with product* by finite lexical extension, enabling us to indirectly apply the parsing methods discussed in Chapter 3 to NL grammar, as we will see in Section 5.4. However, 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 the method of Le Nir, as it stands, contains mistakes which I will repair.

With respect to complexity, we will see in Chapter 7 that a simple generalization of the method presented here gives a polynomial algorithm for NL. Hence, I will provide there a constructive proof of the result of polynomiality of [de Groote, 1999] and [de Groote and Lamarche, 2002].

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 $\mathfrak{a} \to \mathfrak{c}$. We propose once more NL in a slightly more compact way and without the lambda terms. 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 5.1. The system C.

For $\mathfrak{a}, \mathfrak{b}$ and $\mathfrak{c}$ ranging over formulas, we have

- **Identities**:

$$\text{Axioms} \qquad \mathfrak{a} \to \mathfrak{a} \qquad\qquad \text{Cut} \qquad \frac{\mathfrak{a} \to \mathfrak{b} \quad \mathfrak{b} \to \mathfrak{c}}{\mathfrak{a} \to \mathfrak{c}}$$

- **Residuation Rules**:

$$\frac{\mathfrak{a} \otimes \mathfrak{b} \to \mathfrak{c}}{\mathfrak{a} \to \mathfrak{c}/\mathfrak{b}} \qquad \frac{\mathfrak{a} \otimes \mathfrak{b} \to \mathfrak{c}}{\mathfrak{b} \to \mathfrak{a}\backslash\mathfrak{c}}$$

The following are some well known theorems of NL. We label them as $\mathcal{AX}^0$ since they will be important in the construction that follows.

Definition 5.2. For all formulas $\mathfrak{a}$ and $\mathfrak{c}$, let $\mathcal{AX}^0$ be the set consisting of the axioms:

$$\begin{array}{ll} \mathfrak{c}/\mathfrak{a} \otimes \mathfrak{a} \to \mathfrak{c} & \mathfrak{a} \otimes \mathfrak{a}\backslash\mathfrak{c} \to \mathfrak{c} \\ \mathfrak{a} \to (\mathfrak{a} \otimes \mathfrak{c})/\mathfrak{c} & \mathfrak{a} \to \mathfrak{c}\backslash(\mathfrak{c} \otimes \mathfrak{a}) \\ \mathfrak{a} \to \mathfrak{c}/(\mathfrak{a}\backslash\mathfrak{c}) & \mathfrak{a} \to (\mathfrak{c}/\mathfrak{a})\backslash\mathfrak{c} \end{array}$$

Moreover, we have the following set of *derived rules* of inference, which we call M^0.

Definition 5.3. For all formulas $\mathfrak{a}$ and $\mathfrak{b}$, we define M^0 as the set consisting of the following rules.

$$\frac{\mathfrak{a} \to \mathfrak{a}'}{\mathfrak{a} \otimes \mathfrak{b} \to \mathfrak{a}' \otimes \mathfrak{b}}\ \otimes\mathsf{M} \qquad \frac{\mathfrak{b} \to \mathfrak{b}'}{\mathfrak{a} \otimes \mathfrak{b} \to \mathfrak{a} \otimes \mathfrak{b}'}\ \otimes\mathsf{M}'$$

$$\frac{\mathfrak{a}' \to \mathfrak{a}}{\mathfrak{b}\backslash\mathfrak{a}' \to \mathfrak{b}\backslash\mathfrak{a}}\ \backslash\mathsf{M} \qquad \frac{\mathfrak{a}' \to \mathfrak{a}}{\mathfrak{a}'/\mathfrak{b} \to \mathfrak{a}/\mathfrak{b}}\ /\mathsf{M}$$

$$\frac{\mathfrak{b} \to \mathfrak{b}'}{\mathfrak{b}'\backslash\mathfrak{a} \to \mathfrak{b}\backslash\mathfrak{a}}\ \backslash\mathsf{M}' \qquad \frac{\mathfrak{b} \to \mathfrak{b}'}{\mathfrak{a}/\mathfrak{b}' \to \mathfrak{a}/\mathfrak{b}}\ /\mathsf{M}'$$

Let us define the set of sequents which can be obtained from $\mathcal{AX}^0$ and the identity axioms, by means of the rules in M^0 and of two special instances of the cut rule encoded in clause 3 of the following construction. This set will be used below to give an alternative axiomatization of NL, which is, with some minor modification the one used by [Kandulski, 1988].

Definition 5.4. Let $\mathcal{AX}$ denote the smallest set such that:

1. $\mathcal{AX}$ contains the axioms of C and $\mathcal{AX}^0$.
2. $\mathcal{AX}$ is closed under rules M^0.
3. $\mathcal{AX}$ is closed under the following two rules:
   a) if $\mathfrak{a} \to \mathfrak{b} \in \mathcal{AX}$, $|\mathfrak{a}| > |\mathfrak{b}|$ and $\mathfrak{b} \to \mathfrak{c} \in \mathcal{AX}$, $|\mathfrak{b}| > |\mathfrak{c}|$, then $\mathfrak{a} \to \mathfrak{c} \in \mathcal{AX}$.
   b) if $\mathfrak{a} \to \mathfrak{b} \in \mathcal{AX}$, $|\mathfrak{a}| < |\mathfrak{b}|$ and $\mathfrak{b} \to \mathfrak{c} \in \mathcal{AX}$, $|\mathfrak{b}| < |\mathfrak{c}|$, then $\mathfrak{a} \to \mathfrak{c} \in \mathcal{AX}$.

Remark 5.1. In [Kandulski, 1988] one finds a similar construction of a set Ax of axioms. Indeed, our definition of $\mathcal{AX}$ results from Kandulski's Ax set by adding clause 3. This clause is added to include in the set $\mathcal{AX}$ 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 5.9 with the set $\mathcal{AX}$. Roughly, we want to include in this set sequents like, for instance, $\mathfrak{a}/\mathfrak{b} \to \mathfrak{a}'/\mathfrak{b}'$ obtained by monotonous cut on the premises $\mathfrak{a}/\mathfrak{b} \to \mathfrak{a}'/\mathfrak{b}$ and $\mathfrak{a}'/\mathfrak{b} \to \mathfrak{a}'/\mathfrak{b}'$, since such sequents will be derived without using cut in Definition 5.9.

Clearly, every sequent in $\mathcal{AX}$ can be derived in C. Moreover, we can prove the following.

Proposition 5.1. [Kandulski, 1988]: Every sequent derivable in C can be obtained from $\mathcal{AX}$ by means of the Cut rule only.

Thus, we can take $\mathcal{AX}$ plus *cut* as an alternative axiomatization of NL, although we will see in the next section that only one particular instance of cut is required.

Definition 5.5. Let K be the smallest set such that:

1. K contains $\mathcal{AX}$.
2. K is closed under Cut.

From Proposition 5.1, 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. Let us exemplify what is the advantage of using this axiomatization of NL.

The reader might have noticed that K would be equivalent to C even without assuming $a \to c/(a\backslash c)$ and its symmetric form as primitive axioms in $\mathcal{AX}$, as they could be derived.

$$\dfrac{a \to (a \otimes a\backslash c)/(a\backslash c) \quad \dfrac{a \otimes a\backslash c \to c}{(a \otimes a\backslash c)/(a\backslash c) \to c/(a\backslash c)}}{a \to c/(a\backslash c)}$$

The same holds for whatever is derived by means of $\backslash M'$ or of $/M'$, as also these rules could be derived.

$$\dfrac{a/b' \to (a/b' \otimes b)/b \qquad \dfrac{\dfrac{b \to b'}{a/b' \otimes b \to a/b' \otimes b'} \quad a/b' \otimes b' \to a}{a/b' \otimes b \to a}}{(a/b' \otimes b)/b \to a/b}}{a/b' \to a/b}$$

On the other hand, "it is expedient to have them"[1] in $\mathcal{AX}$. In fact, we will see in the next section that every sequent $a \to c$ provable in C is provable in K by means of premises $a \to b$ and $b \to c$ in $\mathcal{AX}$ such that $|a| > |b|$ and $|c| > |b|$.

[Kandulski, 1988] extended the normalization procedure in [Buszkowski, 1986] for NL grammars without product to grammar based on the full non-associative Lambek calculus with product. The procedure, which we present in the next section, proves the reducibility of non-associative Lambek grammars with product formulas to *Ajdukiewicz–Bar-Hillel* grammars with product formulas, and hence the equivalence of non-associative Lambek grammars with product and context-free grammars. We will rely on Kandulski's construction to prove the correctness of the recognition procedure which we formulate in Definition 5.9 on page 103 and Proposition 5.8 on page 111. As we said, our procedure has been suggested by the recursive definition of [Le Nir, 2004] for the compilation of the AB grammar (*without* product) inferable from a given non-associative Lambek grammar *without* product.

5.2 Normal derivations

Since the method of [Kandulski, 1988] applies to sequents of the form $\Gamma \to c$, and we are currently interested only in sequents of the form $a \to c$, we will consider only a few, relevant, cases of the construction. As we said before, 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.

The basic notions that will accompany us throughout this chapter is that of expanding and reducing sequents.

[1] [Kandulski, 1988].

Definition 5.6. A two-formula sequent $a \to c$ from $\mathcal{AX}$ is called *expanding* (resp. *reducing*), if $|a| < |c|$ (resp. $|c| < |a|$).

Observe that all the elements of $\mathcal{AX}$, except the axioms, are either expanding or reducing.

A derivation of a sequent $a \to c$ in K can be seen as a sequence of formulas $x_0 \dots x_n$, which we call *derivation list*, such that

- $x_0 = a$,
- $x_n = c$ and
- for all $i,\ 0 \leqslant i < n$, $x_i \to x_{i+1} \in \mathcal{AX}$.

The "expedient" of Kandulski's construction is schematically illustrated in the following remark.

Remark 5.2. The derivation of a sequent $a \to c$ 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 $i,\ 0 \leqslant i < k,\ x_i \to x_{i+1}$ are reducing patterns from $\mathcal{AX}$ and
- for all $j,\ k \leqslant j < n,\ x_j \to x_{j+1}$ are expanding patterns from $\mathcal{AX}$.

The main property of $\mathcal{AX}$ (which, however, in its original formulation did not include the sequents obtained by clause 3 in the construction of Definition 5.4) is proved in the following proposition, from [Kandulski, 1988].

Proposition 5.2. Given formulas x, y and z, $x \neq z$ and both $x \to y$ and $y \to z$ are in $\mathcal{AX}$, then if $x \to y$ and $y \to z$ are expanding and reducing, respectively, then there is a formula y' such that $|y'| < |y|$ and $x \to y'$ and $y' \to z$ are in $\mathcal{AX}$.

Proof. see [Kandulski, 1988] for NL, the proof for the system without product was given in [Buszkowski, 1986]. □

Example 5.1. Some relevant instances of the proof of Proposition 5.2 are given below.

x	$\to$	y	$\to$	z	y'
a	$\to$	$(a \otimes a\backslash b)/(a\backslash b)$	$\to$	$b/(a\backslash b)$	$b/(a\backslash b)$
a	$\to$	$(a \otimes b)/(a\backslash(a \otimes b))$	$\to$	$(a \otimes b)/b$	$(a \otimes b)/b$
$a \otimes a\backslash b$	$\to$	$b/(a\backslash b) \otimes a\backslash b$	$\to$	b	$a \otimes a\backslash b$

Proposition 5.2 guarantees that every sequent $a \to c$ derivable in NL has a derivation $D = x_0 \dots x_n$ 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 5.7. A derivation $D = x_0 \dots x_n$ of a sequent $a \to c$ 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 $|x_0 \dots x_n| = |x_0| + \dots + |x_n|$.

Definition 5.8. A derivation D of $a \to c$ in K is minimal if and only if for all the derivations D' of $a \to c$ in K, $|D| \leqslant |D'|$.

Remark 5.3. We note that a derivation $x_0 \dots x_n$ 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 5.3. 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 Remark 5.2. 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 process results by application of a transition axioms from $\mathcal{AX}$, expressing a form of *contraction*, of the preceding formula, in the reduction case, or of the following formula, in the expansion case.

Example 5.2. 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 $\mathcal{AX}$. One has the following derivation list:

$$a \otimes a\backslash b,\ b,\ c/(b\backslash c)$$

[2]We used the notation $x < y > z$ for $x < y$ and $y > z$.

5.3 Automatic recognition

In the next sections, we present and discuss our recognition procedure for NL. We will see that, in fact, it can be seen as an alternative definition of Kandulski's normal derivations for NL which has, however, many advantages over the original. In first place, the functions that we present in Definition 5.9 on the next page can immediately be translated into a functional program that works efficiently in any practical case (also by using lists in place of sets)[3]. Secondly, they can be implemented straightforwardly as a chart algorithm (what we do in Algorithm 7.1 on page 134). Finally, they do exactly the work that is required, without *redundancies*.

In cotrast, the sets $\mathcal{AX}$ and K on which Kandulski's normal derivations are constructed are 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 axioms and of the transition schemes, and, closely related, the direction of the proof construction process.

In the search of a proof for a sequent $\mathsf{a} \to \mathsf{c}$, we use only a *finite* number of elements of $\mathcal{AX}$ to perform the transitions from a to c and the choice of these elements is dictated by what a and c look like.

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. Observe that this is not guaranteed in every top-down approach, witness

$$\frac{(\mathsf{a}/\mathsf{b})\backslash\mathsf{a} \not\to \mathsf{b}}{\mathsf{a}/\mathsf{b} \to \mathsf{a}/((\mathsf{a}/\mathsf{b})\backslash\mathsf{a})}$$

which is a possible way of unfolding the valid conclusion through /M. Therefore, our procedure will construct derivations from the leaves to the root, avoiding thus to ever consider invalid sequents.

The key intuition underlying our method is the following. One may observe that *expanding* schemes such as lifting, $\mathsf{a} \to \mathsf{b}/(\mathsf{a}\backslash\mathsf{b})$, or coapplication, $\mathsf{a} \to (\mathsf{a}\otimes\mathsf{b})/\mathsf{b}$, simplify formulas in the same way as the reducing scheme $\mathsf{a}\otimes\mathsf{a}\backslash\mathsf{b} \to \mathsf{b}$ does, although in the opposite direction. In the case of $\mathsf{a}\otimes\mathsf{a}\backslash\mathsf{b} \to \mathsf{b}$ one sees b as the result of contracting $\mathsf{a}\otimes\mathsf{a}\backslash\mathsf{b}$. Instead, in the case of $\mathsf{a} \to \mathsf{b}/(\mathsf{a}\backslash\mathsf{b})$ or $\mathsf{a} \to (\mathsf{a}\otimes\mathsf{b})/\mathsf{b}$ one can see a as the result of contracting $\mathsf{b}/(\mathsf{a}\backslash\mathsf{b})$ or $(\mathsf{a}\otimes\mathsf{b})/\mathsf{b}$. 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.

[3] In Chapter 7, we will discuss this issue in more detail. This claim is justified by the fact that the functions in Definition 5.9 work efficiently for any kind of formula which can appear in a realistic categorial lexicon.

5.3.1 Expansion and reduction

In this section, we present the core routines of our recognition method and prove their correctness. We define two functions e and r of type $\mathcal{F} \to \{\mathcal{F}\}$. To make more compact the presentation below, we use the construct

$$\text{let } \mathtt{x} \text{ be } \nu \text{ in } \mathtt{t}$$

which is another way of expressing the substitution of ν for $\mathtt{x}$ in $\mathtt{t}$, denoted before as $\mathtt{t}[\mathtt{x} := \nu]$. Moreover, we label the clauses in the algorithm so 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*, $\cup$.

Definition 5.9. The functions *expand*, e, and *reduce*, r (we omitted the symmetric cases):

1) $\mathsf{e}(\mathtt{a}) = \{\mathtt{a}\}$, if $\mathtt{a}$ is an atom

2) $\mathsf{e}(\mathtt{a} \otimes \mathtt{b}) = \{\ \mathtt{a}' \otimes \mathtt{b}' \mid \mathtt{a}' \in \mathsf{e}(\mathtt{a})\ \&\ \mathtt{b}' \in \mathsf{e}(\mathtt{b})\ \}$

3) $\mathsf{e}(\mathtt{a}/\mathtt{b}) =$ let $\mathtt{mon}$ be $\{\ \mathtt{a}'/\mathtt{b}' \mid \mathtt{a}' \in \mathsf{e}(\mathtt{a})\ \&\ \mathtt{b}' \in \mathsf{r}(\mathtt{b})\ \}$ in

(a) $\mathtt{mon}$

$\cup$

(b) $\{\ \mathtt{c} \mid (\mathtt{c} \otimes \mathtt{b}')/\mathtt{b}' \in \mathtt{mon}\ \}$

$\cup$

(c) $\{\ \mathtt{c} \mid \mathtt{a}'/(\mathtt{c}\backslash\mathtt{a}') \in \mathtt{mon}\ \}$

1') $\mathsf{r}(\mathtt{a}) = \{\mathtt{a}\}$, if $\mathtt{a}$ is an atom

2') $\mathsf{r}(\mathtt{a} \otimes \mathtt{b}) =$ let $\mathtt{mon}$ be $\{\ \mathtt{a}' \otimes \mathtt{b}' \mid \mathtt{a}' \in \mathsf{r}(\mathtt{a})\ \&\ \mathtt{b}' \in \mathsf{r}(\mathtt{b})\ \}$ in

(a) $\mathtt{mon}$

$\cup$

(b) $\{\ \mathtt{c} \mid \mathtt{c}/\mathtt{b}' \otimes \mathtt{b}' \in \mathtt{mon}\ \}$

$\cup$

(c) $\{\ \mathtt{c} \mid \mathtt{a}' \otimes \mathtt{a}'\backslash\mathtt{c} \in \mathtt{mon}\ \}$

3') $\mathsf{r}(\mathtt{a}/\mathtt{b}) = \{\ \mathtt{a}'/\mathtt{b}' \mid \mathtt{a}' \in \mathsf{r}(\mathtt{a})\ \&\ \mathtt{b}' \in \mathsf{e}(\mathtt{b})\ \}$

Observe that the sets $\mathsf{e}(\mathtt{x})$ and $\mathsf{r}(\mathtt{x})$ are *finite* for every formula $\mathtt{x}$ as all their elements are shorter than $\mathtt{x}$ or identical to $\mathtt{x}$. Let us show some examples of the way these functions work.

Example 5.3. We calculate reduction and expansion of some formulas showing the trace of the recursion as a tree.

$r((s/(n\backslash s))\backslash s)$:

$$\dfrac{r(s)=\{s\} \quad \dfrac{e(s)=\{s\} \quad \dfrac{r(s)=\{s\} \quad e(n)=\{n\}}{r(n\backslash s)=\{n\backslash s\}}}{e(s/(n\backslash s))=\{s/(n\backslash s),\ n\}}}{r((s/(n\backslash s))\backslash s)=\{(s/(n\backslash s))\backslash s,\ n\backslash s\}}$$

$e((s/(n\backslash s))\backslash s)$:

$$\dfrac{e(s)=\{s\} \quad \dfrac{r(s)=\{s\} \quad \dfrac{e(s)=\{s\} \quad r(n)=\{n\}}{e(n\backslash s)=\{n\backslash s\}}}{r(s/(n\backslash s))=\{s/(n\backslash s)\}}}{e((s/(n\backslash s))\backslash s)=\{(s/(n\backslash s))\backslash s,\ n\backslash s\}}$$

$r((a\otimes a\backslash c)/b\otimes b)$:

$$\dfrac{\dfrac{r(a)=\{a\} \quad \dfrac{r(c)=\{c\} \quad e(a)=\{a\}}{r(a\backslash c)=\{a\backslash c\}}}{r(a\otimes a\backslash c)=\{a\otimes a\backslash c,\ c\}} \quad e(b)=\{b\}}{r((a\otimes a\backslash c)/b)=\{(a\otimes a\backslash c)/b,\ c/b\}} \quad r(b)=\{b\}}{r((a\otimes a\backslash c)/b\otimes b)=\{(a\otimes a\backslash c)/b\otimes b,\ c/b\otimes b,\ a\otimes a\backslash c,\ c\}}$$

We may observe that for each of these examples, the following two conditionals hold:

1. If $x\in e(y)$, then $x\to y\in \mathcal{AX}$ and $|x|<|y|$ or $x\equiv y$.
2. If $y\in r(x)$, then $x\to y\in \mathcal{AX}$ and $|x|>|y|$ or $x\equiv y$.

We refer to these two statements as to *soundness* of Definition 5.9 and we prove them in Proposition 5.5. We will see that also the converse statements hold, namely

1. If $x\to y\in \mathcal{AX}$ and $|x|<|y|$ or $x\equiv y$, then $x\in e(y)$.
2. If $x\to y\in \mathcal{AX}$ and $|x|>|y|$ or $x\equiv y$, then $y\in r(x)$.

We refer to these two statements as to *completeness* of Definition 5.9 and we prove them in Proposition 5.6.

Finally, by *correctness* of Definition 5.9, we mean that it is both sound and complete.

Correctness of Definition 5.9

The functions in Definition 5.9 are recursive functions. Inferences are drawn from the premises to the conclusion as a result of recursion: a complex problem, represented by a non-atomic input formula, is divided into two subproblems.

Atoms are the trivial solutions, that is the leaves of the search, or proof, tree. The sets of solutions for the subproblems provide the premises for the solution of the problem.

Before proving correctness of Definition 5.9 we add the following remark.

Remark 5.4. The reader may have noticed that subclauses (b) and (c) in 3) and 2') do *not* compute the *closure* of the set $\mathtt{mon}$ under pattern contraction. In other words, if some of the $\mathtt{c}$ formulas returned, for instance, by clause 3b) or 3c) is itself of the form $(\mathtt{y} \otimes \mathtt{x})/\mathtt{x}$ or $\mathtt{x}/(\mathtt{y}\backslash\mathtt{x})$, no further contraction is applied. This is a very pleasant property of Definition 5.9 as we will see in the next pages that in fact this is enough to guarantee completeness of the definition. In Chapter 6, we will see that computing the closure of the set $\mathtt{mon}$ under pattern contraction would give rise to redundancies.

In order to make the following discussion more compact and more clear, let us introduce the following abbreviations.

Notation 5.1. Let $\mathcal{AX}$ be the set of sequents in Definition 5.4. We adopt the following notational conventions.

$\mathtt{x} \xrightarrow{\mathtt{e}} \mathtt{y} \ := \ \mathtt{x} \to \mathtt{y} \in \mathcal{AX}$ and $|\mathtt{x}| < |\mathtt{y}|$ or $\mathtt{x} \equiv \mathtt{y}$.

$\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y} \ := \ \mathtt{x} \to \mathtt{y} \in \mathcal{AX}$ and $|\mathtt{x}| > |\mathtt{y}|$ or $\mathtt{x} \equiv \mathtt{y}$.

We now proceed to prove the correctness of Definition 5.9, which we may state as follows.

- $\mathtt{x} \in \mathtt{e}(\mathtt{y})$ if and only if $\mathtt{x} \xrightarrow{\mathtt{e}} \mathtt{y}$.
- $\mathtt{y} \in \mathtt{r}(\mathtt{x})$ if and only if $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y}$.

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 5.4. For all $\mathtt{x}$, $\mathtt{x} \in \mathtt{r}(\mathtt{x})$ and $\mathtt{x} \in \mathtt{e}(\mathtt{x})$.

Proof. Induction on $\mathtt{x}$. If $\mathtt{x}$ is atomic, then $\mathtt{x} \in \mathtt{r}(\mathtt{x})$ and $\mathtt{x} \in \mathtt{e}(\mathtt{x})$. Otherwise $\mathtt{x} \equiv \mathtt{y}\sharp\mathtt{z}$, $\sharp \in \{\otimes, /, \backslash\}$. By IH, $\mathtt{y} \in \mathtt{r}(\mathtt{y})$ and $\mathtt{y} \in \mathtt{e}(\mathtt{y})$, and $\mathtt{z} \in \mathtt{r}(\mathtt{z})$ and $\mathtt{z} \in \mathtt{e}(\mathtt{z})$, we conclude $\mathtt{y}\sharp\mathtt{z} \in \mathtt{r}(\mathtt{y}\sharp\mathtt{z})$ and $\mathtt{y}\sharp\mathtt{z} \in \mathtt{e}(\mathtt{y}\sharp\mathtt{z})$. □

Soundness of Definition 5.9 is proved as follows.

Proposition 5.5. Soundness:

(A) If $\mathtt{y} \in \mathtt{r}(\mathtt{x})$, then $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y}$.

(B) If $\mathtt{y} \in \mathtt{e}(\mathtt{x})$, then $\mathtt{y} \xrightarrow{\mathtt{e}} \mathtt{x}$.

Proof. Induction on $\mathtt{x}$. If $\mathtt{x}$ is atomic, we have $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{x}$ and $\mathtt{x} \xrightarrow{\mathtt{e}} \mathtt{x}$ by clause 1) of Def. 5.4. Otherwise we have the following cases.

Proof of (A):

1. If $x \equiv x' \otimes x''$, we have the following subcases for y:
   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 Def. 5.4, 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 Def. 5.4.
   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 Def. 5.4, we conclude $x' \otimes x'' \xrightarrow{r} y$ by clause 3a) of Def. 5.4.
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 Def. 5.4, 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 Def. 5.4.

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, $y' \xrightarrow{e} x'$ and $y'' \xrightarrow{r} x''$. By clause 2) of Def. 5.4, we have $y' \otimes y'' \xrightarrow{e} y' \otimes x''$ and $y' \otimes x'' \xrightarrow{e} x' \otimes x''$. Hence, $y' \otimes y'' \xrightarrow{e} x' \otimes x''$ by clause 3b) of Def. 5.4.
2. If $x \equiv x'/x''$, then we have the following subcases for y:
   a) $y \equiv y'/y''$, with $y' \in e(x')$ and $y'' \in r(x'')$. By IH, $y' \xrightarrow{e} x'$ and $x'' \xrightarrow{r} y''$. By clause 2) of Def. 5.4, we have $y'/y'' \xrightarrow{e} x'/y''$ and $x'/y'' \xrightarrow{e} x'/x''$. Hence, $y'/y'' \xrightarrow{e} x'/x''$ by clause 3b) of Def. 5.4.
   b) Let $(y \otimes v)/v \in e(x'/x'')$, with $y \otimes v \in e(x')$ and $v \in r(x'')$. We obtain $(y \otimes v)/v \xrightarrow{e} x'/x''$ like in the previous case. By clause 1) of Def. 5.4, $y \xrightarrow{e} (y \otimes v)/v$. We conclude $y \xrightarrow{e} x'/x''$ by clause 3b) of Def. 5.4.
   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 $v/(y\backslash v) \xrightarrow{e} x'/x''$ like in case 2a. By clause 1) of Def. 5.4, $y \xrightarrow{e} v/(y\backslash v)$. Hence, $y \xrightarrow{e} x'/x''$ by clause 3b) of Def. 5.4.

□

Completeness of Definition 5.9 is proved as follows.

Proposition 5.6. Completeness:

(A) If $x \xrightarrow{r} y$, then $y \in r(x)$.

(B) If $y \xrightarrow{e} x$, then $y \in e(x)$.

Proof. Induction on the $\mathcal{AX}$ derivation. If $\mathtt{x} \equiv \mathtt{y}$, then (A) and (B) hold by Prop. 5.4. Otherwise:

Proof of (A):

1. $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y} \equiv \mathtt{b}/\mathtt{a} \otimes \mathtt{a} \to \mathtt{b}$. By Prop. 5.4, $\mathtt{b}/\mathtt{a} \otimes \mathtt{a} \in \mathtt{r}(\mathtt{b}/\mathtt{a} \otimes \mathtt{a})$. Hence $\mathtt{b} \in \mathtt{r}(\mathtt{b}/\mathtt{a} \otimes \mathtt{a})$ by clause 2'b) of Def. 5.9.

2. $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y}$ is obtained by clause 2) of Def. 5.4. We have the following subcases.

   a) $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y} \equiv \mathtt{a}/\mathtt{b} \xrightarrow{\mathtt{r}} \mathtt{a}'/\mathtt{b}$ and $\mathtt{a} \xrightarrow{\mathtt{r}} \mathtt{a}'$. By IH, $\mathtt{a}' \in \mathtt{r}(\mathtt{a})$. By Prop. 5.4, $\mathtt{b} \in \mathtt{e}(\mathtt{b})$. Hence, $\mathtt{a}'/\mathtt{b} \in \mathtt{r}(\mathtt{a}/\mathtt{b})$ by clause 3') of Def. 5.9.

   b) The other cases in which $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y}$ is obtained by rules in M^0 are similar and we omit them.

3. $\mathtt{x} \xrightarrow{\mathtt{r}} \mathtt{y}$ is obtained by clause 3a) of Def. 5.4. We shall consider the following subcases.

   a)
   $$\dfrac{\dfrac{\mathtt{x}' \xrightarrow{\mathtt{r}} \mathtt{y}'}{\mathtt{x}'/\mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}'/\mathtt{x}''} \quad \dfrac{\mathtt{y}'' \xrightarrow{\mathtt{e}} \mathtt{x}''}{\mathtt{y}'/\mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}'/\mathtt{y}''}}{\mathtt{x}'/\mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}'/\mathtt{y}''}$$
   By IH, $\mathtt{y}' \in \mathtt{r}(\mathtt{x}')$ and $\mathtt{y}'' \in \mathtt{e}(\mathtt{x}'')$. Then, by clause 3') of Def. 5.9, $\mathtt{y}'/\mathtt{y}'' \in \mathtt{r}(\mathtt{x}'/\mathtt{x}'')$.

   b) The following case is resolved like case 3a:
   $$\dfrac{\dfrac{\mathtt{y}'' \xrightarrow{\mathtt{e}} \mathtt{x}''}{\mathtt{x}'/\mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{x}'/\mathtt{y}''} \quad \dfrac{\mathtt{x}' \xrightarrow{\mathtt{r}} \mathtt{y}'}{\mathtt{x}'/\mathtt{y}'' \xrightarrow{\mathtt{r}} \mathtt{y}'/\mathtt{y}''}}{\mathtt{x}'/\mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}'/\mathtt{y}''}$$

   c)
   $$\dfrac{\dfrac{\mathtt{x}' \xrightarrow{\mathtt{r}} \mathtt{y}'}{\mathtt{x}' \otimes \mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}' \otimes \mathtt{x}''} \quad \dfrac{\mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}''}{\mathtt{y}' \otimes \mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}' \otimes \mathtt{y}''}}{\mathtt{x}' \otimes \mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}' \otimes \mathtt{y}''}$$
   By IH, $\mathtt{y}' \in \mathtt{r}(\mathtt{x}')$ and $\mathtt{y}'' \in \mathtt{r}(\mathtt{x}'')$. Then, by clause 2'a) of Def. 5.9, $\mathtt{y}' \otimes \mathtt{y}'' \in \mathtt{r}(\mathtt{x}' \otimes \mathtt{x}'')$.

   d) The following case is resolved like case 3c:
   $$\dfrac{\dfrac{\mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}''}{\mathtt{x}' \otimes \mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{x}' \otimes \mathtt{y}''} \quad \dfrac{\mathtt{x}' \xrightarrow{\mathtt{r}} \mathtt{y}'}{\mathtt{x}' \otimes \mathtt{y}'' \xrightarrow{\mathtt{r}} \mathtt{y}' \otimes \mathtt{y}''}}{\mathtt{x}' \otimes \mathtt{x}'' \xrightarrow{\mathtt{r}} \mathtt{y}' \otimes \mathtt{y}''}$$

e)
$$\dfrac{\dfrac{x' \xrightarrow{r} y/z}{x' \otimes x'' \xrightarrow{r} y/z \otimes x''} \quad \dfrac{x'' \xrightarrow{r} z}{y/z \otimes x'' \xrightarrow{r} y/z \otimes z}}{\dfrac{x' \otimes x'' \xrightarrow{r} y/z \otimes z \qquad y/z \otimes z \xrightarrow{r} y}{x' \otimes x'' \xrightarrow{r} y}}$$

By IH, $y/z \in r(x')$ and $z \in r(x'')$. Then, by clause 2'a) of Def. 5.9, $y/z \otimes z \in r(x' \otimes x'')$ and by clause 2'b) $y \in r(x' \otimes x'')$.

f) The following case is resolved like case 3e.
$$\dfrac{\dfrac{x'' \xrightarrow{r} z}{x' \otimes x'' \xrightarrow{r} x' \otimes z} \quad \dfrac{x' \xrightarrow{r} y/z}{x' \otimes x'' \xrightarrow{r} y/z \otimes x''}}{\dfrac{x' \otimes x'' \xrightarrow{r} y/z \otimes z \qquad y/z \otimes z \xrightarrow{r} y}{x' \otimes x'' \xrightarrow{r} y}}$$

g) Suppose that $x \xrightarrow{r} y \in \mathcal{A}X^0$ and $y \xrightarrow{r} z \in \mathcal{A}X^0$. Then $x \xrightarrow{r} z \in \mathcal{A}X$. For instance,
$$\dfrac{(c/b \otimes b)/a \otimes a \to c/b \otimes b \quad c/b \otimes b \to c}{(c/b \otimes b)/a \otimes a \to c}$$
Then, let $x[y := z]$ be the result of substituting z for y in x (for instance, $((c/b \otimes b)/a \otimes a)[c/b \otimes b := c] = c/a \otimes a$)[4]. Then $x \xrightarrow{r} x[y := z] \notin \mathcal{A}X^0$ and $x[y := z] \xrightarrow{r} z \in \mathcal{A}X^0$. Thus one may apply case 3e.

h) Finally, suppose that $x' \otimes x'' \xrightarrow{r} y$, derived as in case 3e, is the left premise and $y \xrightarrow{r} v \in \mathcal{A}X^0$ is the right premise. Then, the analysis of case 3g shows that there is another derivation of $x' \otimes x'' \xrightarrow{r} v/z \otimes z$ and $v/z \otimes z \xrightarrow{r} v \in \mathcal{A}X^0$, to which case 3e applies.

Proof of (B):

1. $y \xrightarrow{e} x \equiv a \xrightarrow{e} (a \otimes b)/b$. By Prop. 5.4, $(a \otimes b)/b \in e((a \otimes b)/b)$. Hence $a \in e((a \otimes b)/b)$ by clause 3b) of Def. 5.9.

2. $y \xrightarrow{e} x \equiv a \xrightarrow{e} b/(a\backslash b)$. By Prop. 5.4, $b/(a\backslash b) \in e(b/(a\backslash b))$. Hence $a \in e(b/(a\backslash b))$ by clause 3c) of Def. 5.9.

3. $y \xrightarrow{e} x$ is obtained by clause 2) of Def. 5.4. We have the following subcases.

[4] We refer to Proposition 6.1 in Chapter 6 for the exact definition of this form of substitution which replaces only a precise subformula *occurrence*, which we call there *inner* formulas. For instance, $(a/a \otimes a)[a := b] = b/a \otimes a$, $((a \otimes a)/a)[a := b] = (b \otimes a)/a$ and $(a/(a\backslash a))[a := b] = a/(b\backslash a)$.

a) $\mathfrak{y} \xrightarrow{e} \mathfrak{x} \equiv \mathfrak{a}'/\mathfrak{b} \xrightarrow{e} \mathfrak{a}/\mathfrak{b}$ and $\mathfrak{a}' \xrightarrow{e} \mathfrak{a}$. By IH, $\mathfrak{a}' \in e(\mathfrak{a})$. By Prop. 5.4, $\mathfrak{b} \in r(\mathfrak{b})$. Hence, $\mathfrak{a}'/\mathfrak{b} \in e(\mathfrak{a}/\mathfrak{b})$ by clause 3a) of Def. 5.9.

b) The other cases in which $\mathfrak{x} \xrightarrow{e} \mathfrak{y}$ is obtained by rules in M^0 are similar and we omit them.

4. $\mathfrak{y} \xrightarrow{e} \mathfrak{x}$ is obtained by clause 3b) of Def. 5.4. These cases are dual to those considered in case 3 in the proof of the (A) statement. Therefore we consider only a few subcases.

a)
$$\dfrac{\dfrac{\mathfrak{y}' \xrightarrow{e} \mathfrak{x}'}{\mathfrak{y}' \otimes \mathfrak{y}'' \xrightarrow{e} \mathfrak{x}' \otimes \mathfrak{y}''} \quad \dfrac{\mathfrak{y}'' \xrightarrow{e} \mathfrak{x}''}{\mathfrak{x}' \otimes \mathfrak{y}'' \xrightarrow{e} \mathfrak{x}' \otimes \mathfrak{x}''}}{\mathfrak{y}' \otimes \mathfrak{y}'' \xrightarrow{e} \mathfrak{x}' \otimes \mathfrak{x}''}$$

By IH, $\mathfrak{y}' \in e(\mathfrak{x}')$ and $\mathfrak{y}'' \in e(\mathfrak{x}'')$. Then, by clause 2) of Def. 5.9, $\mathfrak{y}' \otimes \mathfrak{y}'' \in e(\mathfrak{x}' \otimes \mathfrak{x}'')$.

b)
$$\dfrac{\dfrac{\mathfrak{y}' \xrightarrow{e} \mathfrak{x}'}{\mathfrak{y}'/\mathfrak{y}'' \xrightarrow{e} \mathfrak{x}'/\mathfrak{y}''} \quad \dfrac{\mathfrak{x}'' \xrightarrow{r} \mathfrak{y}''}{\mathfrak{x}'/\mathfrak{y}'' \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}}{\mathfrak{y}'/\mathfrak{y}'' \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}$$

By IH, $\mathfrak{y}' \in e(\mathfrak{x}')$ and $\mathfrak{y}'' \in r(\mathfrak{x}'')$. Then, by clause 3a) of Def. 5.9, $\mathfrak{y}'/\mathfrak{y}'' \in e(\mathfrak{x}'/\mathfrak{x}'')$.

c)
$$\dfrac{\mathfrak{y} \xrightarrow{e} (\mathfrak{y} \otimes \mathfrak{z})/\mathfrak{z} \qquad \dfrac{\dfrac{\mathfrak{y} \otimes \mathfrak{z} \xrightarrow{e} \mathfrak{x}'}{(\mathfrak{y} \otimes \mathfrak{z})/\mathfrak{z} \xrightarrow{e} \mathfrak{x}'/\mathfrak{z}} \quad \dfrac{\mathfrak{x}'' \xrightarrow{r} \mathfrak{z}}{\mathfrak{x}'/\mathfrak{z} \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}}{(\mathfrak{y} \otimes \mathfrak{z})/\mathfrak{z} \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}}{\mathfrak{y} \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}$$

By IH, $\mathfrak{y} \otimes \mathfrak{z} \in e(\mathfrak{x}')$ and $\mathfrak{z} \in r(\mathfrak{x}'')$. Then, by clause 3a) of Def. 5.9, $(\mathfrak{y} \otimes \mathfrak{z})/\mathfrak{z} \in e(\mathfrak{x}'/\mathfrak{x}'')$ and by clause 3a) $\mathfrak{y} \in e(\mathfrak{x}'/\mathfrak{x}'')$.

d)
$$\dfrac{\mathfrak{y} \xrightarrow{e} \mathfrak{z}/(\mathfrak{y}\backslash\mathfrak{z}) \qquad \dfrac{\dfrac{\mathfrak{x}'' \xrightarrow{r} \mathfrak{y}\backslash\mathfrak{z}}{\mathfrak{z}/(\mathfrak{y}\backslash\mathfrak{z}) \xrightarrow{e} \mathfrak{z}/\mathfrak{x}''} \quad \dfrac{\mathfrak{z} \xrightarrow{e} \mathfrak{x}'}{\mathfrak{z}/\mathfrak{x}'' \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}}{\mathfrak{z}/(\mathfrak{y}\backslash\mathfrak{z}) \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}}{\mathfrak{y} \xrightarrow{e} \mathfrak{x}'/\mathfrak{x}''}$$

By IH, $\mathfrak{z} \in e(\mathfrak{x}')$ and $\mathfrak{y}\backslash\mathfrak{z} \in r(\mathfrak{x}'')$. Then, by clause 3a) of Def. 5.9, $\mathfrak{z}/(\mathfrak{y}\backslash\mathfrak{z}) \in e(\mathfrak{x}'/\mathfrak{x}'')$ and by clause 3c) $\mathfrak{y} \in e(\mathfrak{x}'/\mathfrak{x}'')$.

e) Suppose that $z \xrightarrow{e} y \in \mathcal{AX}^0$ and $y \xrightarrow{e} x \in \mathcal{AX}^0$. Then $z \xrightarrow{r} x \in \mathcal{AX}$. For instance,

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

Then, let $x[y := z]$ be the result of substituting z for y in x (for instance, $(a/(((c \otimes b)/b)\backslash a))[(c \otimes b)/b := c] = a/(c\backslash a)$). Then $z \xrightarrow{e} x[y := z] \in \mathcal{AX}^0$ and $x[y := z] \xrightarrow{e} x \notin \mathcal{AX}^0$. Thus one may apply one of the previous cases.

f) Finally, suppose that $y \xrightarrow{e} x'/x''$, derived as in case 4c (resp. as in case 4d), is the right premise and $v \xrightarrow{e} y \in \mathcal{AX}^0$ is the left premise. Then, the analysis of case 4e shows that there is another derivation of $v \xrightarrow{e} (v \otimes z)/z \in \mathcal{AX}^0$ (resp. of $v \xrightarrow{e} z/(v\backslash z) \in \mathcal{AX}^0$) and of $(v \otimes z)/z \xrightarrow{e} x'/x''$ (resp. of $z/(v\backslash z) \xrightarrow{e} x'/x''$), to which case 4c (resp. case 4d) applies.

□

We have proved the equivalence of the set of reducing, expanding and identity sequents of $\mathcal{AX}$ with the sets generated by r and e from Definition 5.9. By construction, $\mathcal{AX}$ 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 5.7.

If $y \in r(x)$, then $r(y) \subseteq r(x)$.

If $y \in e(x)$, then $e(y) \subseteq e(x)$.

Proof. This follows from Proposition 5.5 and Proposition 5.6. However, we examine one non trivial case, as this proposition also justifies Remark 5.4 on page 105.

Assume that $a/((b/(c\backslash b))\backslash a) \in e(x)$. We know from Proposition 5.5 that $a/((b/(c\backslash b))\backslash a) \xrightarrow{e} x$. Then we have also $b/(c\backslash b) \xrightarrow{e} x$ and $c \xrightarrow{e} x$ by monotonous cut. On the other hand, as we observed in Remark 5.4, only one cut is applied for each pattern of Def. 5.9. Thus one may suspect that $c \notin e(x)$. However, if $a/((b/(c\backslash b))\backslash a) \in e(x)$, then for some x' such that $x \equiv y/(x'\backslash y')$, $a \xrightarrow{e} y$ and $y' \xrightarrow{r} a$, we have $b/(c\backslash b) \in e(x')$, hence $c \in e(x')$ by clause 3c) of Def. 5.9. Thus, also $a/(c\backslash a) \in e(x)$, by clause 3a), since $a \in e(y)$ and $a \in r(y')$ by IH. Hence $c \in e(x)$ by clause 3c) of Def. 5.9. The reader is also referred to the third of the examples in 5.3 on page 103 for a similar case.

The other cases are similar and we omit them. □

We now give the connection between the two sets generated by the functions e and r to provability in general. As we know that a sequent $a \to c$ is provable in NL if and only if it has a normal derivation in K and that a normal derivation (or deduction) is structured, roughly[5], as

$$\frac{a \xrightarrow{r} b \quad b \xrightarrow{e} c}{a \to c}$$

we state the following result.

Proposition 5.8.

$$\vdash_{NL} a \to c \quad \text{iff} \quad r(a) \cap e(c) \neq \emptyset.$$

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 Prop. 5.5.

Only if part: If $\vdash_{NL} a \to c$, then 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, $0 \leqslant i < k$, $x_i \xrightarrow{r} x_{i+1}$, and for all j, $k \leqslant j < n$, $x_j \xrightarrow{e} x_{j+1}$. By Prop. 5.6 and Prop. 5.7, $x_k \in r(x_0)$ and $x_k \in e(x_n)$. □

Example 5.4.

We prove that $\vdash_{NL} a \otimes a\backslash b \to c/(b\backslash c)$ as follows.

$$r(a \otimes a\backslash b) \cap e(c/(b\backslash c)) = \{a \otimes a\backslash b,\ b\} \cap \{c/(b\backslash c),\ b\} = \{b\}$$

Similarly, from Example 5.3 on page 103, by application of Prop. 5.8 we can conclude:

$$\begin{aligned} &r((s/(n\backslash s))\backslash s) \cap e((s/(n\backslash s))\backslash s) = \\ &\{(s/(n\backslash s))\backslash s,\ n\backslash s\} \cap \{(s/(n\backslash s))\backslash s,\ n\backslash s\} = \\ &\{(s/(n\backslash s))\backslash s,\ n\backslash s\} \end{aligned}$$

5.3.2 Remarks on expansion and reduction

[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 there are some remarkable differences with our construction.

[5] We observed that the case in which one of the premises is an identity should be excluded.

Definition 5.10. [Le Nir, 2004]: expansion and reduction operations for the product free fragment of NL (symmetric cases omitted and some irrelevant notational changes).

$$
\begin{array}{lcl}
\mathsf{E}(a) & = & \{a\}, \text{ if } a \text{ is an atom} \\
\mathsf{R}(a) & = & \{a\}, \text{ if } a \text{ is an atom} \\
\mathsf{R}(a/b) & = & \{\ a'/b' \mid a' \in \mathsf{R}(a)\ \&\ b' \in \mathsf{E}(b)\ \} \\
\mathsf{E}(a/b) & = & \{\ a'/b' \mid a' \in \mathsf{E}(a)\ \&\ b' \in \mathsf{R}(b)\ \} \\
 & & \qquad\qquad \cup \\
 & & \{\ z \mid b \equiv x\backslash c\ \&\ z \in \mathsf{E}(x)\ \&\ a \in \mathsf{R}(c) \vee a \in \mathsf{E}(c)\ \}
\end{array}
$$

Observe that the second clause of $\mathsf{E}(a/b)$, which is in fact 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, $x \in \mathsf{E}(z/(x\backslash(y/(z\backslash y))))$. On the other hand $x \to z/(x\backslash(y/(z\backslash y)))$ is not a valid sequent, as the reader can easily verify. Indeed, Le Nir's arguments involving expanding patters are rather confusing: in many places he seems to write $x \in \mathsf{E}(y)$ meaning $y \in \mathsf{E}(x)$. On the other hand, he explicitly states that $b \in \mathsf{E}(a/(b\backslash a))$. The same observations hold for [Le Nir, 2003b]. Thus one shall expect overgeneration by Le Nir's method for the compilation of an AB grammar out of an NL grammar without product. From Definition 5.9, one can easily recover the product-free cases.

Interpolation

We observe that in the proof of a sequent $a \to c$ according to the method in Proposition 5.8, all the formulas b in $\mathsf{e}(a) \cap \mathsf{r}(c)$ are *interpolants* in the sense of [Roorda, 1991, 1994]. This means that every atom occurring with polarity p in b also occurs with polarity p in a and in c. We refer also to [Retoré, 2005] for a detailed analysis of interpolation in the associative Lambek calculus. In Chapter 7, we will see how to constrain the search of interpolants to *subformulas* of the input sequent.

5.3.3 Extensions

An immediate extension of the expansion and reduction procedures is the application to the unary operators of [Moortgat, 1997]. The set $\mathcal{AX}$ can be extended to include the axioms and rules for the diamond and box operators.

$$
a \to \Box\Diamond a \qquad\qquad \Diamond\Box a \to a
$$

$$
\frac{a \to c}{\Box a \to \Box c}\ \Box\mathsf{M} \qquad\qquad \frac{a \to c}{\Diamond a \to \Diamond c}\ \Diamond\mathsf{M}
$$

In turn, Definition 5.9 can be extended to deal with the unary operators by just adding the clauses in Figure 5.1 on the next page.

$$
\begin{array}{lcl}
e(\Diamond a) & = & \{\, \Diamond a' \mid a' \in e(a) \,\} \\
e(\Box a) & = & \text{let } mon \text{ be } \{\, \Box a' \mid a' \in e(a) \,\} \text{ in} \\
 & & \quad mon \\
 & & \quad \cup \\
 & & \quad \{\, c \mid \Box\Diamond c \in mon \,\} \\
r(\Diamond a) & = & \text{let } mon \text{ be } \{\, \Diamond a' \mid a' \in r(a) \,\} \text{ in} \\
 & & \quad mon \\
 & & \quad \cup \\
 & & \quad \{\, c \mid \Diamond\Box c \in mon \,\} \\
r(\Box a) & = & \{\, \Box a' \mid a' \in r(a) \,\}
\end{array}
$$

Figure 5.1: Expansion and reduction for unary opeartors

5.3.4 The underlying deductive system

In Definition 5.9 and Proposition 5.8, we implicitly made use of transition schemes and inference rules which can be used to formulate a new axiomatization of NL. We call this deductive system ER and we will use it also in the following chapters.

Definition 5.11. The system ER.

- **Identities**:

Axioms	Cut
$a \to a$	$\dfrac{a \to b \quad b \to c}{a \to c}$

- **Unary Rules**

Application:	$\dfrac{a \to b \otimes b\backslash c}{a \to c}$	$\dfrac{a \to c/b \otimes b}{a \to c}$
Lifting:	$\dfrac{(b/a)\backslash b \to c}{a \to c}$	$\dfrac{b/(a\backslash b) \to c}{a \to c}$
Coapplication:	$\dfrac{b\backslash(b \otimes a) \to c}{a \to c}$	$\dfrac{(a \otimes b)/b \to c}{a \to c}$

- **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 of Definition 5.9 gives rise to the following deduction rules.

Definition 5.12. *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 Connection to parsing

Although Definition 5.9 provides a recognition procedure for two-formula sequents, the method can be easily generalized to the more general problem of *parsing*, or more precisely of the recognition of sequents $a_1, \ldots, a_n \to c$ whose antecedent structure is not given.

The result in [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 $AB^{\otimes}$ computations as follows (see also [Buszkowski, 1997]).

Proposition 5.9. If $a_1, \ldots, a_n \to c$ 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 $b_1, \ldots, b_n \to b'$ is derivable only by means of the rules 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 5.10. From a NL categorial grammar $G = \langle V_t, s, Lex, NL\rangle$, we compute an AB grammar $G' = \langle V_t, s, Lex', AB^{\otimes}\rangle$ where

$$Lex' = \{\, w \to x' \mid w \to x \in Lex,\ x' \in r(x) \,\}$$

such that $L_t(G) = L_t(G')$.

Example 5.5. Let us use starred variables for symbols in V_t and write $x \to y_1 | \ldots | y_n$ for $x \to y_1, \ldots, x \to y_n$.

A_6 is the NL grammar with the following lexicon:

$$\begin{aligned} n^\star &\to n \mid s/(n\backslash s) \mid tv\backslash(s/(n\backslash s))\backslash s \mid (s/(n\backslash s) \otimes tv)\backslash s \\ tv^\star &\to tv \\ hv^\star &\to (s/(n\backslash s))\backslash s \end{aligned}$$

Lexical expansion, according to Proposition 5.10, of grammar A_6 gives the $AB^\otimes$ grammar whose lexicon is the following.

$$\begin{aligned} n^\star &\to n \mid s/(n\backslash s) \mid tv\backslash(s/(n\backslash s))\backslash s \mid tv\backslash n\backslash s \mid (s/(n\backslash s) \otimes tv)\backslash s \mid (n \otimes tv)\backslash s \\ tv^\star &\to tv \\ hv^\star &\to (s/(n\backslash s))\backslash s \mid n\backslash s \end{aligned}$$

Thus the parsing methods developed in Chapter 3 can be immediately applied to the resulting grammar.

5.5 Conclusion

We started this chapter by discussing Buszkowski's and Kandulski's method for the construction of normal derivations. Then I defined two recursive functions, called e and r, which generate respectively the set of expanding and the set of reducing sequents of Kandulski's construction[6]. I 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, I will investigate the issue of *ambiguity* of normal derivations.

Chapter 7 examines the complexity of the expansion and reduction procedure. We will observe that the complexity of an algorithm based on Definition 5.9 is exponential and will design a new algorithm, closely related to the first, which recognizes valid sequents of NL in polynomial time.

[6]To be precise, these functions generate sets of formulas. However, we have that $a \in e(c)$ if and only if $a \to c$ is an expanding (or identity) sequent and $c \in r(a)$ if and only if $a \to c$ is a reducing (or identity) sequent, in Kandulski's sense.

Chapter 6

Normal Derivations and Ambiguity

Proposition 5.8 on page 111 provides a simple and elegant method for proving two-formula sequents. The proof of correctness of this method has been based on the results of [Buszkowski, 1986] and [Kandulski, 1988] on normal derivations in NL. We proved the equivalence of our recursive functions e and r in Definition 5.9 with Kandulski's characterization of the sets of expanding and reducing sequents of NL.

In this chapter, I show that while the functions in Definition 5.9 do the same job as Kandulski's construction, in fact they do it better. In the first place, this should already be clear, because Definition 5.9 and Proposition 5.8 represent, in fact, a recognition *algorithm*. Secondly, and this is the central topic of this chapter, because our recognition method is not affected by the problem of *spurious ambiguity*. We mentioned before that *normal* in the sense of [Buszkowski, 1986] and [Kandulski, 1988], does not imply *uniqueness* since we may have several equivalent derivations of the same two-formula sequent according to Kandulski's method[1]. Instead, the recursive functions e and r in Definition 5.9 return exactly one deduction for every semantic reading that a sequent may have, as I prove in Section 6.1. Thus, the method we designed in Proposition 5.8 for proving two formula sequents is a *redundancy-free* theorem prover.

I will address also a second problem, which can be stated as follows.

- Given a provable sequent $\mathsf{a} \to \mathsf{c}$, how many *proofs*, that is how many different semantic readings, may this sequent have?

This question has been previously addressed in [van Benthem, 1991] and [Tiede, 1999a] and represents an important issue for proof theoretic grammars. In

[1] Indeed, uniqueness was not a concern for these authors. [Buszkowski, 1997] simply states that "each proof can be reduced to a *normal* form which has some *nice* computational features". Cursive mine.

[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"[2]. I present such an "explicit function" for NL sequents. The only parameter of the function is the length of the input sequent. While van Benthem and later [Tiede, 1999a] prove the so called *finite reading property* for sequents of the Lambek calculus (with permutation), in Section 6.2, I 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) $(\mathsf{a} \otimes \mathsf{a}\backslash\mathsf{c})/\mathsf{b} \otimes \mathsf{b},\ \mathsf{c}/\mathsf{b} \otimes \mathsf{b},\ \mathsf{c}$

(6.2) $(\mathsf{a} \otimes \mathsf{a}\backslash\mathsf{c})/\mathsf{b} \otimes \mathsf{b},\ \mathsf{a} \otimes \mathsf{a}\backslash\mathsf{c},\ \mathsf{c}$

These two derivations have the same length. Moreover, no shorter derivation is available for the sequent $(\mathsf{a} \otimes \mathsf{a}\backslash\mathsf{c})/\mathsf{b} \otimes \mathsf{b} \to \mathsf{c}$. Thus, they are both minimal, hence *normal* by Proposition 5.3. However, while distinct, these derivations are also in some sense *equivalent*. We have seen in Chapter 2 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 derivations, however, as well as for other logical systems, the problem of equivalence is more subtle and deep. In order to express more clearly the problem of derivational ambiguity in NL, we use the lambda term semantics of derivations. This will allow us to define equivalence of different NL derivations of the same sequent.

Let us assign to each axiom and rule that we used in the construction in Definition 5.4 and in Definition 5.5 on page 98 a lambda term. Instead of $\lambda\mathsf{x}\lambda\mathsf{y}.\mathsf{t}$ we write $\lambda\mathsf{x}\mathsf{y}.\mathsf{t}$.

Definition 6.1. The extended system K' is the closure of $\mathcal{AX}'$ under labeled cut

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

where $\mathcal{AX}'$ is defined as $\mathcal{AX}$ in Definition 5.4 but on the basis of the extended axioms $\mathcal{AX}'^0$ and inference rules M'^0 given below[3].

[2] Cursive mine. The quotation of [van Benthem, 1991] is taken from [Tiede, 1999a] who quotes the American edition of the book.

[3] The monotonous cut of clause 3) of Definition 5.4 are special cases of the labeled cut rule.

- $\mathcal{AX}'^0$:

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

$$\lambda x.(\pi^2 x\ \pi^1 x)\colon b \otimes b\backslash c \to c \qquad \lambda x.(\pi^1 x\ \pi^2 x)\colon c/b \otimes b \to c$$

$$\lambda xy.(y\ x)\colon a \to (b/a)\backslash b \qquad \lambda xy.(y\ x)\colon a \to b/(a\backslash b)$$

$$\lambda xy.\langle y, x\rangle\colon a \to b\backslash(b \otimes a) \qquad \lambda xy.\langle x, y\rangle\colon a \to (a \otimes b)/b$$

- M'^0:

$$\frac{u\colon a \to a'}{\lambda x.\langle(u\ \pi^1 x), \pi^2 x\rangle\colon a \otimes b \to a' \otimes b} \qquad \frac{v\colon b \to b'}{\lambda x.\langle\pi^1 x, (v\ \pi^2 x)\rangle\colon a \otimes b \to a \otimes b'}$$

$$\frac{u\colon a' \to a}{\lambda xy.\,(u\ (x\ y))\colon b\backslash a' \to b\backslash a} \qquad \frac{u\colon a' \to a}{\lambda xy.\,(u\ (x\ y))\colon a'/b \to a/b}$$

$$\frac{v\colon b \to b'}{\lambda xy.\,(x\ (v\ y))\colon b'\backslash a \to b\backslash a} \qquad \frac{v\colon b \to b'}{\lambda xy.\,(x\ (v\ y))\colon a/b' \to a/b}$$

A derivation list of a sequent $a \to c$ in K' with semantic annotation is a pair of a lambda term and of a list of formulas, which we write as

$$\lambda x.(f_{n-1}\ (f_{n-2} \dots (f_1\ x)))\colon a_1 \dots a_n$$

such that $a = a_1$ and $c = a_n$ and for all i, $1 \leqslant i < n$, $f_i\colon a_i \to a_{i+1} \in \mathcal{AX}'$.

The construction of Kandulski's normal derivations in Chapter 5 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 6.2. Let D_1 and D_2 be two semantically annotated normal derivations of a sequent $a \to c$. Let t_1 and t_2 be the normal form lambda terms associated to D_1 and D_2, respectively. Then D_1 is equivalent to D_2 if and only if $t_1 \equiv t_2$.

Someone might object that in system K' the symmetric variants of all axioms and rules in which only slashes appear receive the same semantic annotation. Of course, we could have used directional lambda terms as in [Buszkowski, 1987] and [Hepple, 1994]. However, for our purposes this is not necessary. For instance, the trivial derivations $\lambda xy.(y\ x)\colon a \to (b/a)\backslash b$ and $\lambda xy.(y\ x)\colon a \to b/(a\backslash b)$ are not equivalent, as they are not two derivations of the same sequent (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.

Example 6.1. Redundancies:

(6.3) Labeled deduction corresponding to derivation 6.1:

$$\dfrac{\dfrac{\dfrac{\lambda x.(\pi^1 x\ \pi^2 x)\colon a \otimes a\backslash c \to c}{\lambda xy.(\pi^2(x\ y)\ \pi^1(x\ y))\colon (a \otimes a\backslash c)/b \to c/b}}{\lambda x.\langle \lambda y.(\pi^2(\pi^1 x\ y)\ \pi^1(\pi^1 x\ y)), \pi^2 x\rangle\colon (a \otimes a\backslash c)/b \otimes b \to c/b \otimes b}}{\lambda x.(\pi^2(\pi^1 x\ \pi^2 x)\ \pi^1(\pi^1 x\ \pi^2 x))\colon (a \otimes a\backslash c)/b \otimes b \to c}$$

(6.4) Labeled deduction corresponding to derivation 6.2:

$$\dfrac{\lambda x.(\pi^1 x\ \pi^2 x)\colon (a \otimes a\backslash c)/b \otimes b \to a \otimes a\backslash c}{\lambda x.(\pi^2(\pi^1 x\ \pi^2 x)\ \pi^1(\pi^1 x\ \pi^2 x))\colon (a \otimes a\backslash c)/b \otimes b \to c}$$

The term in the conclusion of the two deductions is indeed the same, hence the two deductions are equivalent.

The previous example shows that Kandulski's method of constructing normal derivations is not exempt from redundancies. There is in fact another case in which we may derive the same sequent in two equivalent ways. Consider the following example.

Example 6.2. Further redundancies: the following two deductions are equivalent.

1.
$$\dfrac{\dfrac{\nu\colon a \to a'}{\lambda xy.(\nu\ (x\ y))\colon a/b \to a'/b} \quad \dfrac{u\colon b' \to b}{\lambda xy.(x\ (u\ y))\colon a'/b \to a'/b'}}{\lambda xy.(\nu\ (x\ (u\ y)))\colon a/b \to a'/b'}$$

2.
$$\dfrac{\dfrac{u\colon b' \to b}{\lambda xy.(x\ (u\ y))\colon a/b \to a/b'} \quad \dfrac{\nu\colon a \to a'}{\lambda xy.(\nu\ (x\ y))\colon a/b' \to a'/b'}}{\lambda xy.(\nu\ (x\ (u\ y)))\colon a/b \to a'/b'}$$

The choice of which derivations are redundant is, to some extent, *arbitrary*. In the case of Example 6.2 there seem to be no criterion to chose one deduction rather than the other. However, both these deductions are instances of a more general schema, which is the following:

$$\dfrac{\nu\colon a \to a' \quad u\colon b' \to b}{\lambda xy.(\nu\ (x\ (u\ y)))\colon a/b \to a'/b'}$$

thus by adopting this inference rule in place of $/\mathsf{M}$ and $/\mathsf{M}'$ of Definition 5.3 on page 97, we can avoid this kind of redundancies.

Instead, the deductions in 6.3 and in 6.4 in Example 6.1 exhibit two different structures. Deduction 6.1 follows the *innermost* reduction strategy: the most

embedded formulas are contracted before the most external ones. Deduction 6.1 follows the *outermost* reduction strategy: the most external formulas are contracted before the most embedded ones. We will consider redundant the deductions following the outermost reduction. In other words, whenever two derivations (or deductions) D_1 and D_2 are available and they 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 is easy to control and implement. Indeed we saw already, in Remark 5.4 on page 105 and in the proof of Proposition 5.7, and will see below in more detail, that Definition 5.9 implements the innermost reduction. In the next chapter, we will examine also the benefits of an outermost reduction strategy.

A final, trivial, case of redundancy is that of non-atomic identity axioms which may also be derived through the rules of M^0.

We start by redefining the set $\mathcal{AX}$ of Definition 5.4 so as to eliminate the redundancies of the form discussed before. Let us write $\mathsf{a} \trianglerighteq \mathsf{b}$ for a sequent $\mathsf{a} \to \mathsf{b}$ such that $|\mathsf{a}| > |\mathsf{b}|$ or $\mathsf{a} \equiv \mathsf{b}$ and $\mathsf{a} \trianglelefteq \mathsf{b}$ for a sequent $\mathsf{a} \to \mathsf{b}$ such that $|\mathsf{a}| < |\mathsf{b}|$ or $\mathsf{a} \equiv \mathsf{b}$. Moreover, to avoid repetitions in the following construction, we define $\widetilde{\trianglerighteq} = \trianglelefteq$ and $\widetilde{\trianglelefteq} = \trianglerighteq$, and we use $\diamond$ as a variable over $\trianglerighteq$ and $\trianglelefteq$.

Definition 6.3. Let $\mathcal{AX}^\star$ denote the smallest set such that:

1. $\mathcal{AX}^\star$ contains for every *atom* a the axiom $\mathsf{a} \to \mathsf{a}$.
2. $\mathcal{AX}^\star$ contains $\mathcal{AX}^0$.
3. $\mathcal{AX}^\star$ is closed under the following rules, which we call M^1:

$$\frac{\mathsf{a} \diamond \mathsf{a}' \quad \mathsf{b} \diamond \mathsf{b}'}{\mathsf{a} \otimes \mathsf{b} \diamond \mathsf{a}' \otimes \mathsf{b}'} \otimes\mathsf{M}^1$$

$$\frac{\mathsf{a} \diamond \mathsf{a}' \quad \mathsf{b}' \mathbin{\widetilde{\diamond}} \mathsf{b}}{\mathsf{b}\backslash\mathsf{a} \diamond \mathsf{b}'\backslash\mathsf{a}'} \backslash\mathsf{M}^1$$

$$\frac{\mathsf{a} \diamond \mathsf{a}' \quad \mathsf{b}' \mathbin{\widetilde{\diamond}} \mathsf{b}}{\mathsf{a}/\mathsf{b} \diamond \mathsf{a}'/\mathsf{b}'} /\mathsf{M}^1$$

4. $\mathcal{AX}^\star$ is closed under the following two rules:
   a) if $\mathsf{a} \trianglerighteq \mathsf{b} \in \mathcal{AX}^\star$ and $\mathsf{b} \trianglerighteq \mathsf{c} \in \mathcal{AX}^0$ and $\mathsf{a} \to \mathsf{b}$ is not itself in $\mathcal{AX}^0$ nor is it derived by this same clause, then $\mathsf{a} \to \mathsf{c} \in \mathcal{AX}^\star$.
   b) if $\mathsf{a} \trianglelefteq \mathsf{b} \in \mathcal{AX}^0$ and $\mathsf{b} \trianglelefteq \mathsf{c} \in \mathcal{AX}^\star$ and $\mathsf{b} \to \mathsf{c}$ is not itself in $\mathcal{AX}^0$ nor is it derived by this same clause, then $\mathsf{a} \to \mathsf{c} \in \mathcal{AX}^\star$.

We shall prove the equivalence of $\mathcal{AX}^\star$ and $\mathcal{AX}$. Observe that $\mathcal{AX}^\star$ is more restrictive than $\mathcal{AX}$ in its construction method. In particular, it assumes *atomic*

identity axioms and the monotonous cuts in clause 3 impose special conditions to their premises, which are lacking from the corresponding clause in Definition 5.4. The proof of the following proposition shows that these extra requirements can always be satisfied.

In the proof we make use of the notion of *inner formula*. This is the subformula $\mathtt{z}$ of $\mathtt{x}$ such that $\mathtt{z} \to \mathtt{x} \in \mathcal{AX}^0$ if $\mathtt{z} \to \mathtt{x}$ is expanding or $\mathtt{x} \to \mathtt{z} \in \mathcal{AX}^0$ if $\mathtt{x} \to \mathtt{z}$ is reducing. For instance, $\mathtt{a}$ is the inner formula of $(\mathtt{a} \otimes \mathtt{b})/\mathtt{b}$ or of $\mathtt{b}/(\mathtt{a}\backslash\mathtt{b})$ or of $\mathtt{b} \otimes \mathtt{b}\backslash\mathtt{a}$. By replacement of the inner formula $\mathtt{y}$ for $\mathtt{z}$ in $\mathtt{x}$, we mean the replacement of the inner *occurrence* of $\mathtt{y}$ in $\mathtt{x}$. For instance, the result of the replacement of the inner formula $\mathtt{a}$ for $\mathtt{b}$ in $(\mathtt{a} \otimes \mathtt{a})/\mathtt{a}$ is $(\mathtt{b} \otimes \mathtt{a})/\mathtt{a}$.

Proposition 6.1. The set of sequents $\mathcal{AX}^\star$ is equivalent to the set of sequents $\mathcal{AX}$ in Definition 5.4.

Proof. We prove that each set is included in the other.

$\mathcal{AX}^\star \subseteq \mathcal{AX}$. Clearly, in $\mathcal{AX}$ we can derive everything that is derivable in $\mathcal{AX}^\star$.

$\mathcal{AX} \subseteq \mathcal{AX}^\star$. Induction on the $\mathcal{AX}$ derivation:

1. Non-atomic identity axioms are derivable via M^1.
2. $\mathcal{AX}^0$ is common to $\mathcal{AX}$ and $\mathcal{AX}^\star$.
3. Rules M^1 are a generalization of rules M^0. Thus whatever is obtained by M^0 is derivable by using M^1 with the identity as one of the premises.
4. Assume that $\mathtt{a} \diamond \mathtt{a}'$ and $\mathtt{b}' \mathrel{\widetilde{\diamond}} \mathtt{b}$ are in $\mathcal{AX}$. Then by $/\mathsf{M}$, also $\mathtt{a}/\mathtt{b} \diamond \mathtt{a}'/\mathtt{b}$ and $\mathtt{a}'/\mathtt{b} \diamond \mathtt{a}'/\mathtt{b}'$ are in $\mathcal{AX}$. Then by clause 3 of Definition 5.4, also $\mathtt{a}/\mathtt{b} \diamond \mathtt{a}'/\mathtt{b}'$ is in $\mathcal{AX}$. By IH, $\mathtt{a} \diamond \mathtt{a}'$ and $\mathtt{b}' \mathrel{\widetilde{\diamond}} \mathtt{b}$ are in $\mathcal{AX}^\star$ and $\mathtt{a}/\mathtt{b} \diamond \mathtt{a}'/\mathtt{b}'$ is in $\mathcal{AX}^\star$ by $/\mathsf{M}^1$.
5. One proves the other cases in which the premises of the monotonous cut (in clause 3 of Definition 5.4) are obtained trough rules in M^0 in a similar way.
6. Assume that $\mathtt{a} \trianglelefteq \mathtt{b} \in \mathcal{AX}^0$ and $\mathtt{b} \trianglelefteq \mathtt{c} \in \mathcal{AX}$. There are the following cases:
   a) $\mathtt{b} \trianglelefteq \mathtt{c} \in \mathcal{AX}^0$. Then $\mathtt{b}$ is the inner formula of $\mathtt{c}$. Thus by replacing in $\mathtt{c}$ the inner formula $\mathtt{b}$ for $\mathtt{a}$, we obtain $\mathtt{c}'$, such that $\mathtt{a} \to \mathtt{c}' \in \mathcal{AX}^0$. Instead, $\mathtt{c}' \to \mathtt{c}$ is not in $\mathcal{AX}^0$, although it is in $\mathcal{AX}$. Hence, $\mathtt{c}' \to \mathtt{c} \in \mathcal{AX}^\star$ by IH and $\mathtt{a} \to \mathtt{c}$ is obtained in clause 4b of the construction of $\mathcal{AX}^\star$.
   b) $\mathtt{b} \trianglelefteq \mathtt{c}$ is derived from $\mathtt{b} \trianglelefteq \mathtt{b}' \in \mathcal{AX}^0$ and $\mathtt{b}' \trianglelefteq \mathtt{c} \in \mathcal{AX}$. The previous case shows that there is another derivation of $\mathtt{a} \trianglelefteq \mathtt{b}'$. Then, by IH and clause 4b it follows that $\mathtt{a} \trianglelefteq \mathtt{c} \in \mathcal{AX}^\star$.
   c) $\mathtt{b} \trianglelefteq \mathtt{c}$ is neither in $\mathcal{AX}^0$ nor derived by clause 4b. Then $\mathtt{a} \trianglelefteq \mathtt{c} \in \mathcal{AX}^\star$ by clause 4b of the construction of $\mathcal{AX}^\star$.

7. The case in which $\mathtt{a} \trianglerighteq \mathtt{b} \in \mathcal{AX}$ and $\mathtt{b} \trianglerighteq \mathtt{c} \in \mathcal{AX}^0$ is similar to the previous one and we omit it.

□

As we discussed before, the construction method of $\mathcal{AX}^\star$ is more regimented than that of $\mathcal{AX}$. Let us consider once more Examples 6.2 and 6.1. Clearly the redundancy arising in Example 6.2 does not affect $\mathcal{AX}^\star$ as the only way of obtaining $\mathtt{a}/\mathtt{b} \to \mathtt{a}'/\mathtt{b}'$ is by means of $/\mathsf{M}^1$. Concerning the redundancy in Example 6.1, observe that $(\mathtt{a} \otimes \mathtt{a}\backslash\mathtt{c})/\mathtt{b} \otimes \mathtt{b} \to \mathtt{a} \otimes \mathtt{a}\backslash\mathtt{c}$ and $\mathtt{a} \otimes \mathtt{a}\backslash\mathtt{c} \to \mathtt{c}$ in

$$(\mathtt{a} \otimes \mathtt{a}\backslash\mathtt{c})/\mathtt{b} \otimes \mathtt{b},\ \ \mathtt{a} \otimes \mathtt{a}\backslash\mathtt{c},\ \ \mathtt{c}$$

are both in $\mathcal{AX}^0$, hence clause 4a of Definition 6.3 does not apply. However, $(\mathtt{a}\otimes\mathtt{a}\backslash\mathtt{c})/\mathtt{b}\otimes\mathtt{b} \to \mathtt{c}/\mathtt{b}\otimes\mathtt{b}$ is not in $\mathcal{AX}^0$ (although it is in $\mathcal{AX}^\star$), while $\mathtt{c}/\mathtt{b}\otimes\mathtt{b} \to \mathtt{c}$ is in $\mathcal{AX}^0$. Thus we obtain $(\mathtt{a} \otimes \mathtt{a}\backslash\mathtt{c})/\mathtt{b} \otimes \mathtt{b} \to \mathtt{c}$ from these two premises by clause 4a of the construction.

The final question is whether there are other kinds of redundancies affecting the construction method of $\mathcal{AX}^\star$. In order to prove that this is not the case we make use again of lambda terms. We give here the semantic labeling of the rules M^1 of $\mathcal{AX}^\star$.

$$\frac{\nu\colon \mathtt{a} \to \mathtt{a}' \quad \mathtt{u}\colon \mathtt{b} \to \mathtt{b}'}{\lambda\mathtt{x}.\langle(\nu\ \pi^1\mathtt{x}), (\mathtt{u}\ \pi^2\mathtt{x})\rangle\colon \mathtt{a} \otimes \mathtt{b} \to \mathtt{a}' \otimes \mathtt{b}'}\ \otimes\mathsf{M}^1$$

$$\frac{\nu\colon \mathtt{a} \to \mathtt{a}' \quad \mathtt{u}\colon \mathtt{b}' \to \mathtt{b}}{\lambda\mathtt{x}\mathtt{y}.(\nu\ (\mathtt{x}\ (\mathtt{u}\ \mathtt{y})))\colon \mathtt{b}\backslash\mathtt{a} \to \mathtt{b}'\backslash\mathtt{a}'}\ \backslash\mathsf{M}^1$$

$$\frac{\nu\colon \mathtt{a} \to \mathtt{a}' \quad \mathtt{u}\colon \mathtt{b}' \to \mathtt{b}}{\lambda\mathtt{x}\mathtt{y}.(\nu\ (\mathtt{x}\ (\mathtt{u}\ \mathtt{y})))\colon \mathtt{a}/\mathtt{b} \to \mathtt{a}'/\mathtt{b}'}\ /\mathsf{M}^1$$

Let $\overline{\mathcal{AX}}^\star$ be defined as $\mathcal{AX}^\star$ except for the fact that it is a *multi-set*[4] rather than a set. We refer to the extended $\overline{\mathcal{AX}}^\star$ as to the variant of $\overline{\mathcal{AX}}^\star$ extended with lambda terms and we assume that all lambda terms are in normal form. We state the following final result.

Proposition 6.2. Assume that $\mathtt{t}_1\colon \mathtt{a} \to \mathtt{c}$ and $\mathtt{t}_2\colon \mathtt{a} \to \mathtt{c}$ are in the extended $\overline{\mathcal{AX}}^\star$. Then $\mathtt{t}_1 \neq \mathtt{t}_2$.

Proof. In first place, we observe that the rules in M^1 do not engender spurious ambiguity in themselves, although they may inherit it from their premises. Thus, we shall look only at the transitions in $\mathcal{AX}^0$. However, in this case spurious ambiguity might arise only from monotonous cuts between premises $\mathtt{x} \to \mathtt{y}$ and $\mathtt{y} \to \mathtt{z}$ in $\mathcal{AX}^0$ which are excluded from $\mathcal{AX}^\star$ by construction. □

[4] A multi-set is a set-like object in which order is ignored, but multiplicity is explicitly significant. Therefore, multisets $\{1, 2, 3\}$ and $\{2, 1, 3\}$ are equivalent, but $\{1, 1, 2, 3\}$ and $\{1, 2, 3\}$ differ. One can also think of a multi-set as a list, the order of the elements of which is ignored.

Observe also that in general the normal derivation of a sequent $\mathfrak{a} \to \mathfrak{c}$ consists in a reducing derivation of a sequent $\mathfrak{a} \to \mathfrak{b} \in \mathcal{AX}^\star$ and in the expanding derivation of a sequent $\mathfrak{b} \to \mathfrak{c} \in \mathcal{AX}^\star$ from which $\mathfrak{a} \to \mathfrak{c}$ is obtained by cut. As the construction of the two derivations which give the premises of the cut is free from spurious ambiguity, also the derivation of $\mathfrak{a} \to \mathfrak{c}$ is.

Comparison with the expansion and reduction sets

Let us compare the construction method of $\mathcal{AX}^\star$ and the procedure that we presented in Definition 5.9 on page 103. We shall see that the two systems construct deductions in exactly the same way. Hence that the procedure encoded in Definition 5.9 is free from spurious ambiguity.

Let us begin with an example. We restate Definition 5.9 to be sensitive to multiplicity.

Definition 6.4. Let us define functions e' and r', which are like e and r in Definition 5.9, respectively, except for the fact that they use $[\,]$ instead of $\{\,\}$, that is, list comprehension instead of set comprehension, and $+\!\!+$ instead of $\cup$, that is, list concatenation instead of set union[5].

We consider once more the formula $(\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}\otimes\mathfrak{b}$ that we used to exemplify the kind of redundancies that we want to eliminate.

Example 6.3. If we compute $\mathsf{r}'((\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}\otimes\mathfrak{b})$, we have the following trace of the recursion.

$$\dfrac{\dfrac{\mathsf{r}'(\mathfrak{a}) = [\,\mathfrak{a}\,] \qquad \dfrac{\mathsf{r}'(\mathfrak{c}) = [\,\mathfrak{c}\,] \quad \mathsf{e}'(\mathfrak{a}) = [\,\mathfrak{a}\,]}{\mathsf{r}'(\mathfrak{a}\backslash\mathfrak{c}) = [\,\mathfrak{a}\backslash\mathfrak{c}\,]}}{\mathsf{r}'(\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c}) = [\,\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c},\ \mathfrak{c}\,]} \qquad \mathsf{e}'(\mathfrak{b}) = [\,\mathfrak{b}\,]}{\mathsf{r}'((\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}) = [\,(\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b},\ \mathfrak{c}/\mathfrak{b}\,]} \qquad \mathsf{r}'(\mathfrak{b}) = [\,\mathfrak{b}\,]}{\mathsf{r}'((\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}\otimes\mathfrak{b}) = [\,(\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}\otimes\mathfrak{b},\ \mathfrak{c}/\mathfrak{b}\otimes\mathfrak{b},\ \mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c},\ \mathfrak{c}\,]}$$

We may observe that although both $\mathfrak{c}/\mathfrak{b}\otimes\mathfrak{b}$ and $\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c}$ are in the list $\mathsf{r}'((\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}\otimes\mathfrak{b})$, only *one* occurrence of the formula $\mathfrak{c}$ is in this list. In fact, $\mathfrak{c}$ is obtained at the root by contracting $\mathfrak{c}/\mathfrak{b}\otimes\mathfrak{b}$ (which is also obtained at the root, but by clause 2'a) of Definition 5.9). Instead, $\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c}$ is obtained, again at the root, by contracting $(\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}\otimes\mathfrak{b}$. As in the construction of $\mathcal{AX}^\star$, we prevent the formula $\mathfrak{a}\otimes\mathfrak{a}\backslash\mathfrak{c}$ from being the input for further contraction in Definition 5.9 by computing only one cycle of pattern simplification, that is, by simply mapping the set $\mathtt{mon}$ in clauses b) and c) of Definition 5.9 under pattern contraction rather than computing the *closure* of the set $\mathtt{mon}$ under contraction operations.

[5]Indeed, as the order of the elements is not important, we could use multisets rather than lists. However, it makes no difference for the sake of the argument

Remark 6.1. It is interesting to compare this situation to what might be the case for the *associative* Lambek calculus. If we wish to build normal derivations for this system, we could, for example, capture among the pattern contractions, the expanding pattern $\mathtt{a}/\mathtt{b} \to (\mathtt{a}/\mathtt{c})/(\mathtt{b}/\mathtt{c})$. Thus, if a formula of the form $(\mathtt{a}/\mathtt{c})/(\mathtt{b}/\mathtt{c})$ were in the set $\mathfrak{mon}$, we would generate $\mathtt{a}/\mathtt{b}$, in the same way as we generate $\mathtt{c}$, if $\mathtt{a}/(\mathtt{c}\backslash\mathtt{a})$ is in $\mathfrak{mon}$ in Definition 5.9. Observe, anyway, that in this case $\mathtt{a}/\mathtt{b}$ is not a subformula of $(\mathtt{a}/\mathtt{c})/(\mathtt{b}/\mathtt{c})$, thus it may give rise to a further contraction, if, for eample, $\mathtt{a}/\mathtt{b} \equiv (\mathtt{a}'/\mathtt{x})/(\mathtt{b}'/\mathtt{x})$, and so forth. Therefore, in this case, one cycle of pattern simplification would not be enough and we should compute the *closure* of the set $\mathfrak{mon}$ under the appropriate contraction operations.

We conclude this section with the following result.

Proposition 6.3. The recognition procedure resulting from Proposition 5.8 is free from spurious ambiguity.

Proof. As we have shown before, Definition 5.9 parallels the construction of $\mathcal{AX}^\star$ which is not affected by spurious ambiguity as we proved in Proposition 6.2. In particular, the functions $\mathtt{e}$ and $\mathtt{r}$ in Definition 5.9 exploit the rules M^1 for the recursion, while the way in which the contraction mappings of the set $\mathfrak{mon}$ in clauses b) and c) of Definition 5.9 are applied, has the same effect as the constraints that we imposed in clause 4 of Definition 6.3. □

6.2 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 $\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}) \otimes (\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}))\backslash\mathtt{s} \to \mathtt{s}$, discussed by [Hendriks, 1993] and [de Groote, 1999], among others. This sequent has the following two readings, presented as deductions in ER.

(6.5) Subject wide scope:

$$\dfrac{\dfrac{\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}) \to \mathtt{s}/(\mathtt{n}\backslash\mathtt{s}) \qquad \dfrac{\mathtt{s} \to \mathtt{s} \qquad \dfrac{\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}) \to \mathtt{s}/(\mathtt{n}\backslash\mathtt{s})}{\mathtt{n} \to \mathtt{s}/(\mathtt{n}\backslash\mathtt{s})}}{(\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}))\backslash\mathtt{s} \to \mathtt{n}\backslash\mathtt{s}}}{\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}) \otimes (\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}))\backslash\mathtt{s} \to \mathtt{s}/(\mathtt{n}\backslash\mathtt{s}) \otimes \mathtt{n}\backslash\mathtt{s}}}{\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}) \otimes (\mathtt{s}/(\mathtt{n}\backslash\mathtt{s}))\backslash\mathtt{s} \to \mathtt{s}}$$

(6.6) 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 $\mathcal{AX}^0$ of the basic characteristic laws of NL.

$$\begin{array}{lll} c/a\otimes a\to c & a\otimes a\backslash c\to c & \text{evaluation } (e)\\ a\to(a\otimes c)/c & a\to c\backslash(c\otimes a) & \text{coapplication } (c)\\ a\to c/(a\backslash c) & a\to(c/a)\backslash c & \text{lifting } (l)\end{array}$$

Let us define the following operations, together with their symmetric forms, possibly.

Definition 6.5. For a and b ranging over *atoms*, we define the following operations:

$$\begin{array}{lcl}\phi^x(a)(b)(0) & = & a, \text{ for } x\in\{e,c,l\}\\ \phi^e(a)(b)(i+1) & = & (\phi^e(a)(b)(i))/b\otimes b\\ \phi^c(a)(b)(i+1) & = & ((\phi^c(a)(b)(i))\otimes b)/b\\ \phi^l(a)(b)(i+1) & = & b/((\phi^l(a)(b)(i))\backslash b)\end{array}$$

Each iteration of these operations introduces a pair of connectives. We can simplify the notation if we adopt the following convention.

Notation 6.1. As a and b are arbitrary atoms, and constant in the iteration, we write $\phi^x(n)$, $x\in\{e,c,l\}$, for $\phi^x(a)(b)(n)$.

We may observe the following property.

Remark 6.2. Let $x\in\{c,l\}$ and $n\geqslant 0$. Then each of the following sequents is provable.

$$\begin{array}{lcl}\phi^x(n) & \to & \phi^x(n+1)\\ \phi^x(n+2) & \to & \phi^x(n+1)\end{array}$$

Similar patterns can be proved also for ϕ^e. In order to show this, we uniform the construction by introducing the following abstraction.

Definition 6.6. Let $n\geqslant 0$ and $m>0$. We write $n\overset{\phi^x}{\to}m$ for

$$\begin{array}{lcll}\phi^x(n) & \to & \phi^x(m), & \text{if } x\in\{c,l\}\\ \phi^x(m) & \to & \phi^x(n), & \text{if } x\in\{e\}\end{array}$$

This abstract notation allows us to write $n \xrightarrow{\Phi} m$ for $n \xrightarrow{\Phi^x} m$, the $x \in \{e, l, c\}$ being understood. We call Φ-sequents such kinds of sequents. For example,

(6.7) $2 \xrightarrow{\Phi^c} 3 = ((a \otimes b)/b \otimes b)/b \to (((a \otimes b)/b \otimes b)/b \otimes b)/b$

(6.8) $2 \xrightarrow{\Phi^e} 3 = ((a/b \otimes b)/b \otimes b)/b \otimes b \to (a/b \otimes b)/b \otimes b$

We generalize the properties in Remark 6.2.

Proposition 6.4. Let $n \geqslant 0$. Then each of the following sequents is provable.

$$\begin{array}{rcl} n & \xrightarrow{\Phi} & n+1 \\ n+2 & \xrightarrow{\Phi} & n+1 \end{array}$$

Proof. Clearly, for every $n \geqslant 0$, $n \xrightarrow{\Phi} n$ is provable. We consider only the case of Φ^e, the other cases being similar. Each step in the following deductions is either based on the rules of the system ER from Definition 5.11 or on substitutions of the previous definitions.

1. We first prove that $n \xrightarrow{\Phi} n+1$.

$$\dfrac{\dfrac{\dfrac{\dfrac{n+1 \xrightarrow{\Phi^e} n+1}{\Phi^e(n+1) \to \Phi^e(n+1)}}{\Phi^e(n+1) \to \Phi^e(n)/b \otimes b}}{\Phi^e(n+1) \to \Phi^e(n)}}{n \xrightarrow{\Phi^e} n+1}$$

2. We now prove that $n+2 \xrightarrow{\Phi} n+1$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{n+1 \xrightarrow{\Phi^e} n+1}{\Phi^e(n+1) \to \Phi^e(n+1)}}{\Phi^e(n)/b \otimes b \to \Phi^e(n+1)} \quad b \to b}{(\Phi^e(n)/b \otimes b)/b \to \Phi^e(n+1)/b}}{\Phi^e(n)/b \to \Phi^e(n+1)/b} \qquad b \to b}{\Phi^e(n)/b \otimes b \to \Phi^e(n+1)/b \otimes b}}{\Phi^e(n+1) \to \Phi^e(n+2)}}{n+2 \xrightarrow{\Phi^e} n+1}$$

□

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 exact 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 shall 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 of a given length.

Let us define the following count.

Definition 6.7. Let $n \geqslant 0$ and $m \geqslant 1$.

$$\begin{array}{lcl} \rho(n, m) & = & 1, \text{ if } n \equiv 0 \text{ or } m \equiv 1 \\ \rho(n+1, m+1) & = & \rho(n, m+1) + \rho(n+1, m) \end{array}$$

We now prove that the function ρ counts exactly the number of readings of ϕ-sequents. We write $|a \to c|^{\rho}$ for the number of different proofs of $a \to c$.

Proposition 6.5. Let $n \geqslant 0$ and $m \geqslant 1$. Then

$$|n \xrightarrow{\phi} m|^{\rho} = \rho(n, m)$$

Proof. In the proof, we examine the case of ϕ^c, the other cases being similar.

Case $n \equiv 0$ 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 $m \equiv 1$ 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.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{n-1 \xrightarrow{\phi^c} 1}{\phi^c(n-1) \to (a \otimes b)/b} \quad b \to b}{\phi^c(n-1) \otimes b \to (a \otimes b)/b \otimes b}}{\phi^c(n-1) \otimes b \to a \otimes b} \quad b \to b}{(\phi^c(n-1) \otimes b)/b \to (a \otimes b)/b}}{n \xrightarrow{\phi^c} 1}$$

Case $n > 0$ and $m > 1$. There are two possibilities of obtaining $n \xrightarrow{\phi^c} m$.

1.
$$\dfrac{\dfrac{\dfrac{\dfrac{n \xrightarrow{\phi^c} m-1}{\phi^c(n) \to \phi^c(m-1)} \quad b \to b}{\phi^c(n) \otimes b \to \phi^c(m-1) \otimes b} \quad b \to b}{(\phi^c(n) \otimes b)/b \to (\phi^c(m-1) \otimes b)/b}}{\phi^c(n) \to (\phi^c(m-1) \otimes b)/b}}{n \xrightarrow{\phi^c} m}$$

2.
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{n-1 \xrightarrow{\phi^c} m}{\phi^c(n-1) \to \phi^c(m)}}{\phi^c(n-1) \to (\phi^c(m-1) \otimes b)/b} \quad b \to b}{\phi^c(n-1) \otimes b \to (\phi^c(m-1) \otimes b)/b \otimes b}}{\phi^c(n-1) \otimes b \to \phi^c(m-1) \otimes b} \qquad b \to b}{(\phi^c(n-1) \otimes b)/b \to (\phi^c(m-1) \otimes b)/b}}{n \xrightarrow{\phi^c} m}$$

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 6.5 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 column and m in the topmost row. Notice that this is the well known Pascal's triangle[6].

$\xrightarrow{\phi}$	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.

[6] I wish to thank Michael Moortgat for telling me the name of this beautiful figure.

Remark 6.3. The count $\rho(n, m)$ can be formulated in terms of the *binomial coefficient*.

$$\rho(n, m) = \binom{n + m - 1}{m - 1} = \frac{(n + m - 1)!}{n!(m - 1)!}$$

Coming back to the issue of finding an upper bound to the number of readings 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 $s' \in \Phi_n$, $m \geqslant |s'|^\rho$ (in other words, $\lceil \Phi_n \rceil$ is the highest number of readings that a ϕ-sequents of length n can have). We state the following result.

Proposition 6.6. Let $a \to c$ be given. If $|a \to c| = n$, then $|a \to c|^\rho \leqslant \lceil \Phi_n \rceil$.

Proof. We observe that the monotonicity rules do not engender ambiguity. There remain only the elements of $\mathcal{AX}$ to be considered. ϕ-sequents are instances of sequents in $\mathcal{AX}$. Their shape allows to match literals and connectives inside them, introducing the highest degree of ambiguity. Consider, as an example, the ϕ-sequent $1 \xrightarrow{\phi^c} 2$. This has two readings, namely

$$\frac{((a \otimes b)/b \otimes b)/b \to ((a \otimes b)/b \otimes b)/b}{(a \otimes b)/b \to ((a \otimes b)/b \otimes b)/b}$$

and

$$\dfrac{\dfrac{\dfrac{(a \otimes b)/b \to (a \otimes b)/b}{a \to (a \otimes b)/b} \qquad b \to b}{a \otimes b \to (a \otimes b)/b \otimes b} \qquad b \to b}{(a \otimes b)/b \to ((a \otimes b)/b \otimes b)/b}$$

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 6.7. Let $a \to c$ be given. Let $|a \to c| = n$ and $m = \frac{n}{2} - 1$. Then,

- if $m \equiv 0$, 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 6.5 and 6.6. □

6.3 Conclusion

In this chapter, I have refined Kandulski's notion of normal derivation by eliminating redundancies from the construction. I proved that the procedure in Definition 5.9 and hence the recognition procedure in Proposition 5.8 is a redundancy-free theorem prover for NL.

In exploring the properties of the theorem prover arising from Proposition 5.8, I observed the pleasant progression of Figure 6.1. The link between number of readings of NL sequents and binomial coefficient substantially strengthens the previous results of [van Benthem, 1991] and [Tiede, 1999a]. This result also clarifies [de Groote, 1999] claim about the exponential number of readings of NL sequents.

Chapter 7

Complexity

The non-associative Lambek calculus with product can be parsed in polynomial time as proved in [de Groote, 1999; Buszkowski, 2005; Bulińska, 2006]. The proofs of polynomiality in [Buszkowski, 2005] and [Bulińska, 2006] rely on the *subformula property* which restricts the search space to sequents made of subformulas of the input sequent. The proof in [de Groote, 1999] depends on the subformula property and on structure sharing. We have seen that in general, normal derivations do not respect the subformula property. For example, the derivation in Example 6.1 on page 118 passes through a formula which is not a subformula of the input sequent.

Although the procedure encoded in Definition 5.9 may work fairly well even with complex and highly ambiguous sequents, as it is not affected by spurious ambiguity, we will see in Section 7.1 that it is not polynomial. I will redefine the expansion and reduction functions in such a way that their search space is structured as a graph. However, we will see that even enhanced with structure sharing, this approach does not guarantee polynomiality. As we mentioned before, the main drawback of the expansion and reduction method is that the process of derivation construction does not respect the subformula property.

Then I will present a calculus for NL, closely related to the system ER, which makes it possible to find deductions in accordance with the subformula property and maintains some of the pleasant properties of ER. A charted algorithm based on this system will be described, I will prove its correctness and that it recognizes sequents of NL in polynomial time. I will calculate a polynomial time of $O(n^5)$, where n is the length of the input sequent. Such limit is lower that those calculated in the works discussed before.

Finally, we will see how this polynomial recognition algorithm for two-formula sequents can be applied to parsing multi-formula sequents, that is sequents whose antecedent structure is not given.

7.1 Charted expansion/reduction algorithm

One could easily define the agenda-driven chart-based algorithm implementing Definition 5.9. Let a *polarized formula* be a pair of a boolean and a formula. We define the following operation returning the set of polarized subformulas of a given polarized formula.

Definition 7.1. Polarized subformulas.

$\varphi :: (\mathtt{Bool}, \mathcal{F}) \to \{(\mathtt{Bool}, \mathcal{F})\}$

$$\begin{array}{lcl} \varphi\ (\mathfrak{p}, \mathfrak{a}) & = & \{(\mathfrak{p}, \mathfrak{a})\}, \text{ if } \mathfrak{a} \text{ is an atom.} \\ \varphi\ (\mathfrak{p}, \mathfrak{a} \otimes \mathfrak{b}) & = & \{(\mathfrak{p}, \mathfrak{a} \otimes \mathfrak{b})\} \cup \varphi(\mathfrak{p}, \mathfrak{a}) \cup \varphi(\mathfrak{p}, \mathfrak{b}) \\ \varphi\ (\mathfrak{p}, \mathfrak{a}/\mathfrak{b}) & = & \{(\mathfrak{p}, \mathfrak{a}/\mathfrak{b})\} \cup \varphi(\mathfrak{p}, \mathfrak{a}) \cup \varphi(\overline{\mathfrak{p}}, \mathfrak{b}) \\ \varphi\ (\mathfrak{p}, \mathfrak{b}\backslash\mathfrak{a}) & = & \{(\mathfrak{p}, \mathfrak{b}\backslash\mathfrak{a})\} \cup \varphi(\mathfrak{p}, \mathfrak{a}) \cup \varphi(\overline{\mathfrak{p}}, \mathfrak{b}) \end{array}$$

where $\overline{0} = 1$ and $\overline{1} = 0$.

The main component of the program is given below.

Algorithm 7.1. Agenda-driven chart-based variant of Definition 5.9.

Input: a two-formula sequent $\mathfrak{a} \to \mathfrak{c}$.

Output: a boolean value: 1, if $\vdash_{\mathbf{NL}} \mathfrak{a} \to \mathfrak{c}$, 0 otherwise.

1. `type Agenda` $:= [(\mathtt{Bool}, \mathcal{F})]$. Initialization of the agenda: let $(\mathfrak{y}:\mathfrak{ys})$ be the result of mapping to a sorted list the set $\varphi(0, \mathfrak{a}) \cup \varphi(1, \mathfrak{c})$[1].

2. `type Chart` $:= [((\mathtt{Bool}, \mathcal{F}), \{\mathcal{F}\})]$. Initialization of the chart: $\mathfrak{zs} := [\,]$.

3. `exhaust-agenda` $:: (\mathtt{Chart}, \mathtt{Agenda}) \to (\mathtt{Chart}, \mathtt{Agenda})$

   $$\begin{array}{lll} \texttt{exhaust-agenda} & (\mathfrak{zs}, [\,]) & = (\mathfrak{zs}, [\,]) \\ \texttt{exhaust-agenda} & (\mathfrak{zs}, \mathfrak{y}:\mathfrak{ys}) & = \\ & \multicolumn{2}{l}{\texttt{exhaust-agenda}\ (\mathfrak{y}':\mathfrak{zs}, \mathfrak{ys})} \\ & \multicolumn{2}{l}{\quad \text{where}} \\ & \multicolumn{2}{l}{\qquad \mathfrak{y}' := \texttt{consequences}\ \mathfrak{y}\ \mathfrak{zs}} \end{array}$$

4. Let $((0, \mathfrak{a}), \mathfrak{as}), ((1, \mathfrak{c}), \mathfrak{cs}) \in \mathfrak{zs}$ at the end of step 3. Then,

$$\vdash_{\mathbf{NL}} \mathfrak{a} \to \mathfrak{c} \text{ iff } \mathfrak{as} \cap \mathfrak{cs} \neq \emptyset$$

[1] That is $(\mathfrak{y} : \mathfrak{ys})$ contains *one occurrence* of each $\mathfrak{z} \in \varphi(0, \mathfrak{a}) \cup \varphi(1, \mathfrak{c})$ in complexity increasing order.

Step by step analysis of Algorithm 7.1

Step 1 declares the type and the initial value of the agenda variable ($\mathtt{y{:}ys}$): a sorted list of polarized formulas which will be used to drive the inference process in step 3. Step 2 declares the type and initial value of the chart variable $\mathtt{zs}$. The chart is empty at the beginning. The functions $\mathtt{r}$ and $\mathtt{e}$ in Definition 5.9 can be seen as instances of a function $\mathtt{f}$ taking in input a boolean, a formula and returning a set of formulas, in such a way that $\mathtt{f}\ 1 = \mathtt{e}$ and $\mathtt{f}\ 0 = \mathtt{r}$. We omit the details of the implementation of the function `consequences` of type $(\mathtt{Bool}, \mathcal{F}) \to \mathtt{Chart} \to ((\mathtt{Bool}, \mathcal{F}), \{\mathcal{F}\})$ as they are not particularly relevant and describe informally what it should do in Definition 7.2.

Definition 7.2. Let Algorithm 7.1 be applied to a sequent $\mathtt{x} \to \mathtt{y}$. For each polarized subformula $(\mathtt{p}, \mathtt{a}\sharp\mathtt{b})$ of $\mathtt{x} \to \mathtt{y}$, the function `consequences` in the routine `exhaust-agenda` should perform the following steps:

1. Retrieve from the chart the elements $((\mathtt{p}_1, \mathtt{a}), \mathsf{A})$ and $((\mathtt{p}_2, \mathtt{b}), \mathsf{B})$ where $\mathtt{p}_1$ and $\mathtt{p}_2$ are the appropriate polarities for $(\mathtt{p}, \mathtt{a}\sharp\mathtt{b})$ in input[2].
2. Use the sets of formulas A and B to calculate the set of formulas X inferable for $(\mathtt{p}, \mathtt{a}\sharp\mathtt{b})$. The sets A and B contain the premises. For example, for $(\mathtt{p}, \mathtt{a}\sharp\mathtt{b}) \equiv (1, \mathtt{a}/\mathtt{b})$ we calculate, in a way parallel to Definition 5.9:

$$\begin{aligned} \mathsf{X} \quad = \quad & \text{let } \mathtt{mon} \text{ be } \{\ \mathtt{a}'/\mathtt{b}' \mid \mathtt{a}' \in \mathsf{A}\ \&\ \mathtt{b}' \in \mathsf{B}\ \} \text{ in} \\ & \mathtt{mon} \\ & \cup \\ & \{\ \mathtt{c} \mid (\mathtt{c} \otimes \mathtt{b}')/\mathtt{b}' \in \mathtt{mon}\ \} \\ & \cup \\ & \{\ \mathtt{c} \mid \mathtt{a}'/(\mathtt{c}\backslash\mathtt{a}') \in \mathtt{mon}\ \} \end{aligned}$$

3. Return $((\mathtt{p}, \mathtt{a}\sharp\mathtt{b}), \mathsf{X})$.

Observe that `consequences` is not a *recursive function* as recursion is implemented by `exhaust-agenda`. In fact, when a polarized formula $\mathtt{a}\sharp\mathtt{b}$ (for simplicity, we omit the polarity) is the current input of `consequences`, we know in advance that the polarized formulas $\mathtt{a}$ and $\mathtt{b}$ have already been computed, as they are shorter than the formula $\mathtt{a}\sharp\mathtt{b}$.

Proposition 7.1. If the function `consequences` is implemented according to Definition 7.2, then at each iteration of `exhaust-agenda` in Algorithm 7.1, we have:

1. $((1, \mathtt{x}), \mathsf{Y}) \in \mathtt{zs}$ and $\mathtt{y} \in \mathsf{Y}$ iff $\mathtt{y} \in \mathtt{e}(\mathtt{x})$.
2. $((0, \mathtt{x}), \mathsf{Y}) \in \mathtt{zs}$ and $\mathtt{y} \in \mathsf{Y}$ iff $\mathtt{y} \in \mathtt{r}(\mathtt{x})$.

[2]More precisely: if $\sharp = \otimes$, then $\mathtt{p}_1 = \mathtt{p} = \mathtt{p}_2$, if $\sharp = /$, then $\mathtt{p}_1 = \mathtt{p}$ and $\mathtt{p}_2 = \overline{\mathtt{p}}$, and if $\sharp = \backslash$, then $\mathtt{p}_1 = \overline{\mathtt{p}}$ and $\mathtt{p}_2 = \mathtt{p}$.

Proof. The only difference between the recognition strategy of Definition 5.9 and that of Algorithm 7.1 is that while in the first the search space is organized as a *tree*, in the second it is organized as a *graph*. Sharing is obtained simply by building the agenda as a sorted list from the *set* of polarized subformulas of the input sequent. Thus, multiple occurrences of the same polarized subformula are computed only once. The complexity increasing order of the agenda, guarantees that at each iteration of `exhaust-agenda` the premises needed to derive the consequences of the current input formula are already in the chart. □

As we explained in the introduction, the main drawback of Definition 5.9 and that of Algorithm 7.1 is that they do not respect the subformula property. If we look once more at derivation 6.1 on page 118, in order to prove $(\mathfrak{a} \otimes \mathfrak{a}\backslash\mathfrak{c})/\mathfrak{b}\otimes\mathfrak{b} \to \mathfrak{c}$, we pass through the formula $\mathfrak{c}/\mathfrak{b}\otimes\mathfrak{b}$, which is not a subformula of the input sequent. Observe that in Definition 5.9, the calculation of the size of the set $\mathtt{mon}$ involves the product of the size of the subproblem solutions, see [Le Nir, 2004]. The situation is not different in case of Algorithm 7.1. Since, as we saw in Chapter 6, a formula may give rise to a number of reduced forms which is exponential on its length, this indicates that the procedure is not polynomial. Suppose that we want to calculate the number of inferences for a formula $\mathfrak{a}\sharp\mathfrak{b}$. This involves retrieving from the chart the inferences of $\mathfrak{a}$ and of $\mathfrak{b}$. After we have recovered $(\mathfrak{a}, A)$ and $(\mathfrak{b}, B)$ from the chart, the generation of $(\mathfrak{a}\sharp\mathfrak{b}, X)$ will take $|A| \times |B| + k$ operations of $O(1)$, where k is the number of contracted forms for the $\mathfrak{a}'\sharp\mathfrak{b}'$'s such that $\mathfrak{a}' \in A$ and $\mathfrak{b}' \in B$. Multiplying this reasoning for the number of subformulas of the input sequent gives an exponential growth of space and time.

In our tests of Algorithm 7.1 and Definition 5.9, we could indeed observe such an exponential explosion, although this happened with artificially constructed complex sequents which would not occur in actual linguistic practice. We believe indeed that for linguistic applications the procedure 5.9, eventually implemented as in Algorithm 7.1, would be perfectly adequate.

In the next sections, we show how the recognition method of Definition 5.9 can be generalized in such a way that deductions respecting the subformula property can always be found. This will be the basis for building a polynomial algorithm for NL recognition.

7.2 A calculus for the subformula property

We call $\mathcal{G}$ the following axiomatization of NL.

Definition 7.3. The system $\mathcal{G}$.

- **Identities**:

Axioms	Cut
$\mathfrak{a} \to \mathfrak{a}$	$\dfrac{\mathfrak{a} \to \mathfrak{b} \quad \mathfrak{b} \to \mathfrak{c}}{\mathfrak{a} \to \mathfrak{c}}$

- **Binary Rules**:

Product Rule:
$$\frac{a \to a' \quad b \to b'}{a \otimes b \to a' \otimes b'}$$

Application:
$$\frac{a \to a' \quad b \to a'\backslash c}{a \otimes b \to c} \qquad \frac{a \to c/b' \quad b \to b'}{a \otimes b \to c}$$

Monotonicity:
$$\frac{a' \to a \quad b \to b'}{b'\backslash a' \to b\backslash a} \qquad \frac{a' \to a \quad b \to b'}{a'/b' \to a/b}$$

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

Coapplication:
$$\frac{b \to b' \quad b' \otimes c \to a}{c \to b\backslash a} \qquad \frac{b \to b' \quad c \otimes b' \to a}{c \to a/b}$$

The system $\mathcal{G}$ is closely related to the system 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 ER.

- Application:

$$\frac{a \to a' \quad b \to a'\backslash c}{a \otimes b \to c} \qquad \frac{\dfrac{a \to a' \quad b \to a'\backslash c}{a \otimes b \to a' \otimes a'\backslash c}}{a \otimes b \to c}$$

- Lifting:

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

- Coapplication:

$$\frac{b \to b' \quad b' \otimes c \to a}{c \to b\backslash a} \qquad \frac{\dfrac{b' \otimes c \to a \quad b \to b'}{b'\backslash(b' \otimes c) \to b\backslash a}}{c \to b\backslash a}$$

Furthermore, normal derivations can be constructed in $\mathcal{G}$. For example, the clause for $e(a/b)$ would become as follows.

$$\begin{array}{rcl} e(a/b) & = & \text{let } as \text{ be } e(a) \text{ and } bs \text{ be } r(b) \text{ in} \\ & & \{\, a'/b' \mid a' \in as \;\&\; b' \in bs \,\} \\ & & \cup \\ & & \{\, c \mid c \otimes b' \in as \;\&\; b' \in bs \,\} \\ & & \cup \\ & & \{\, c \mid a' \in as \;\&\; c\backslash a' \in bs \,\} \end{array}$$

However, our primary interest in $\mathcal{G}$ is that it will allow us to always find deductions respecting the subformula property. Let us consider some examples of violation of the subformula property in normal deductions of $\mathcal{G}$.

Example 7.1.

Permutation: Consider the following example, the formula within the box is not a subformula of the conclusion.

$$\dfrac{\dfrac{\dfrac{a \to a \quad a\backslash b \to a\backslash b}{a \otimes a\backslash b \to b} \qquad c \to c}{(a \otimes a\backslash b)/c \to \boxed{b/c}} \qquad c \to c}{(a \otimes a\backslash b)/c \otimes c \to b}$$

In $\mathcal{G}$, there is an equivalent deduction which respects the subformula property:

$$\dfrac{\dfrac{(a \otimes a\backslash b)/c \to (a \otimes a\backslash b)/c \quad c \to c}{(a \otimes a\backslash b)/c \otimes c \to a \otimes a\backslash b} \qquad \dfrac{a \to a \quad a\backslash b \to a\backslash b}{a \otimes a\backslash b \to b}}{(a \otimes a\backslash b)/c \otimes c \to b}$$

Percolation: the cut is replaced with two new cuts of smaller degree, so that the cut formula disappears and the new cut formulas may be subformulas of the root sequent.

$$\dfrac{\dfrac{a \to a \qquad \dfrac{\dfrac{b \to b \qquad \dfrac{a\backslash b \to a\backslash b \quad b \to b}{a \to b/(a\backslash b)}}{(b/(a\backslash b))\backslash b \to a\backslash b}}{a \otimes (b/(a\backslash b))\backslash b \to \boxed{a \otimes a\backslash b}} \qquad \dfrac{\dfrac{a\backslash b \to a\backslash b \quad b \to b}{a \to b/(a\backslash b)} \qquad a\backslash b \to a\backslash b}{\boxed{a \otimes a\backslash b} \to b/(a\backslash b) \otimes a\backslash b}}{a \otimes (b/(a\backslash b))\backslash b \to b/(a\backslash b) \otimes a\backslash b}$$

The variant respecting the subformula property is obtained by moving the cut

upwards.

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

We observe, that both these transformations rely on the use of the cut rule. Furthermore, while they return deductions that respect the subformula property, these new deductions may no longer be normal, in Kandulski's sense. This is the case in the second example. Thus if we want to be able to find deductions respecting the subformula property, we should relax the conditions on normality. We introduce the notion of *quasinormal* deduction.

Definition 7.4. A deduction D in $\mathcal{G}$ is *quasinormal* either if D is normal or if the last step of D is a binary rule of $\mathcal{G}$ applied to the conclusions of quasinormal deductions D_1 and D_2.

According to this definition, normal deductions are a subset of quasinormal deductions. For instance, the deduction respecting the subformula property in the second example in 7.1 is quasinormal (though not normal), while the deduction in Example 6.2 is normal and respects the subformula property.

As in the case of normal deductions, we can see the application each rule as triggered by a polarized subformula of the goal sequent. We call such formula the *trigger* of the inference rule.

Proposition 7.2. Let $\mathfrak{a}\to\mathfrak{b}$ and $\mathfrak{b}\to\mathfrak{c}$, with $\mathfrak{a}\neq\mathfrak{c}$, be deducible in $\mathcal{G}$ with quasinormal deductions. If $|\mathfrak{b}|>|\mathfrak{a}|$ and $|\mathfrak{b}|>|\mathfrak{c}|$, then there is a $\mathfrak{b}'$, $|\mathfrak{b}'|<|\mathfrak{b}|$, such that $\mathfrak{a}\to\mathfrak{b}'$ and $\mathfrak{b}'\to\mathfrak{c}$ are deducible with quasinormal deductions in $\mathcal{G}$.

We will prove now that among all quasinormal deductions of a given sequent we can always find one that respects the subformula property, that is one whose formulas are all subformulas of the goal sequent. Observe that, since the trigger is by definition a subformula of the input sequent, we should only verify that the other formula in the conclusion is also a subformula of the input sequent.

We prove the subformula property for $\mathcal{G}$. We write $\sigma(\mathfrak{x})$ for the set of the subformulas of $\mathfrak{x}$ and $\sigma(\mathfrak{x}\to\mathfrak{y})$ for the set of subformulas of $\mathfrak{x}\to\mathfrak{y}$. Let Σ be a set of formulas, we write $\mathfrak{x}\to_\Sigma\mathfrak{y}$, if $\mathfrak{x}\to\mathfrak{y}$ has a deduction D in $\mathcal{G}$ such that all formulas appearing in D belong to Σ.

Proposition 7.3. Let $\vdash_{\mathcal{G}}\mathfrak{a}\to\mathfrak{c}$ and $\Sigma:=\sigma(\mathfrak{a}\to\mathfrak{c})$. Then $\mathfrak{a}\to_\Sigma\mathfrak{c}$.

Proof. Induction on the quasinormal deduction of $\mathfrak{a}\to\mathfrak{c}$. We work *bottom-up*. This means that in constructing a deduction for $\mathfrak{a}\to\mathfrak{c}$, with $\Sigma=\sigma(\mathfrak{a}\to\mathfrak{c})$, we proceed from premises to conclusion and consider the first introduction of a sequent $\mathfrak{x}\to\mathfrak{y}$, such that either $\mathfrak{x}$ or $\mathfrak{y}$ does not belong to Σ. As the axioms are

obviously in Σ, we may assume, without loss of generality, that the premises contain formulas in Σ. Instead, the conclusion $\mathtt{x} \to \mathtt{y}$ contains a formula which triggers the inference and is therefore in Σ, and another formula which is not in Σ. Such assumption immediately excludes from the analysis the rules of lifting, coapplication and application. Consider lifting for all.

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

Such rule is triggered by the formula $\mathtt{b}\backslash\mathtt{a}$ in the conclusion, which is therefore in Σ. By assumption, the premises are made of formulas in Σ. Therefore, the new formula $\mathtt{c}$ in the conclusion is also in Σ as it occurs as a subformula of one of the premises. Coapplication and application have the same behaviour. The cut rule does not create problems either, as no new formula is introduced in the conclusion. There remain only the monotonicity rules.

Consider the following instance of monotonicity, as triggered by the succedent formula $\mathtt{b}\backslash\mathtt{a}$.

$$\frac{\mathtt{a}' \to \mathtt{a} \quad \mathtt{b} \to \mathtt{b}'}{\mathtt{b}'\backslash\mathtt{a}' \to \mathtt{b}\backslash\mathtt{a}}$$

Then $\mathtt{b}'\backslash\mathtt{a}' \notin \Sigma$. Therefore, $\mathtt{b}'\backslash\mathtt{a}'$ should cancel with a positive occurrence of $\mathtt{b}'\backslash\mathtt{a}'$, derived from an inference of the following kind,

$$\frac{\mathtt{a}'' \to \mathtt{a}' \quad \mathtt{b}' \to \mathtt{b}''}{\mathtt{b}''\backslash\mathtt{a}'' \to \mathtt{b}'\backslash\mathtt{a}'}$$

whose trigger is $\mathtt{b}''\backslash\mathtt{a}''$, to generate $\mathtt{b}''\backslash\mathtt{a}'' \to \mathtt{b}\backslash\mathtt{a}$. We know that $\mathtt{a}' \to \mathtt{a}, \mathtt{b} \to \mathtt{b}', \mathtt{a}'' \to \mathtt{a}'$ and $\mathtt{b}' \to \mathtt{b}''$ are made of formulas in Σ. Thus, we obtain $\mathtt{b}''\backslash\mathtt{a}'' \to \mathtt{b}\backslash\mathtt{a}$, avoiding to pass through $\mathtt{b}'\backslash\mathtt{a}'$, in the following way[3].

$$\frac{\dfrac{\mathtt{a}'' \to \mathtt{a}' \quad \mathtt{a}' \to \mathtt{a}}{\mathtt{a}'' \to \mathtt{a}} \quad \dfrac{\mathtt{b} \to \mathtt{b}' \quad \mathtt{b}' \to \mathtt{b}''}{\mathtt{b} \to \mathtt{b}''}}{\mathtt{b}''\backslash\mathtt{a}'' \to \mathtt{b}\backslash\mathtt{a}}$$

The other cases are similar and we omit them. □

7.3 NL in polynomial time

In the previous section, we proved that in the system $\mathcal{G}$ we can always find derivations respecting the subformula property, that is derivations whose formulas are all subformulas of the root sequent. As we said before, this is the key property which may guarantee polynomiality of the recognition procedure. We wish to remark that derivations respecting the subformula property may

[3]Observe that in this way, the same conclusion can be obtained twice: once as triggered by the antecedent and a second time as triggered by the succedent.

not be *normal* in the sense of the previous chapters. However this does not represent a problem for recognition.

We define now an agenda-driven chart-based algorithm for proving two-formula sequents based on $\mathcal{G}$ and drawing inferences respecting the subformula property.

Algorithm 7.2.

Input: a two-formula sequent $\mathfrak{a} \to \mathfrak{c}$.

Output: a boolean value: 1, if $\vdash_{\mathbf{NL}} \mathfrak{a} \to \mathfrak{c}$, 0 otherwise.

1. `type` $\mathit{Agenda} := [(\mathit{Bool}, \mathcal{F})]$. Initial value of the agenda: let $(\mathfrak{y} : \mathfrak{ys})$ be the result of mapping to a sorted list the set $\varphi(0, \mathfrak{a}) \cup \varphi(1, \mathfrak{c})$.

2. `type` $\mathit{Chart} := \{\mathit{Seq}\}$. Initial value of the chart: $\mathit{zs} := \emptyset$.

3. `type` $\mathit{Subform} := \{\mathcal{F}\}$. $\Sigma := \sigma(\mathfrak{a} \to \mathfrak{c})$, is the set of the subformulas of $\mathfrak{a} \to \mathfrak{c}$.

4. `exhaust-agenda` $:: \mathit{Subform} \to (\mathit{Chart}, \mathit{Agenda}) \to (\mathit{Chart}, \mathit{Agenda})$

   $$\begin{array}{lll} \texttt{exhaust-agenda} & \Sigma\ (\mathit{zs}, [\,]) & = (\mathit{zs}, [\,]) \\ \texttt{exhaust-agenda} & \Sigma\ (\mathit{zs}, (\mathfrak{p}, \mathfrak{y}) : \mathfrak{ys}) & = \\ & \texttt{exhaust-agenda}\ \Sigma\ (\mathit{zs}', \mathfrak{ys}) & \\ & \quad \text{where} & \\ & \qquad \mathfrak{ys}' & := \texttt{infer}\ (\mathfrak{p}, \mathfrak{y})\ \Sigma\ \mathit{zs} \\ & \qquad \mathit{cs} & := \texttt{cut}\ \mathfrak{p}\ \mathfrak{ys}'\ \mathit{zs} \\ & \qquad \mathit{zs}' & := \mathfrak{ys}' \cup \mathit{cs} \cup \mathit{zs} \end{array}$$

5. Recognition: $\vdash_{\mathbf{NL}} \mathfrak{a} \to \mathfrak{c}$ iff $\mathfrak{a} \to \mathfrak{c} \in \mathit{zs}$ after step 4.

At each iteration of step 4, the new inferences are produced by the functions `infer` and `cut`. As we saw in the proof of Proposition 7.3, the subformula property for $\mathcal{G}$ essentially relies on the use of the cut rule. The function `cut` takes a boolean and two sets of sequents as arguments and returns the set of sequents derivable by cut with the elements of the two sets as premises. The polarity determines which set gives which premise.

- `cut` $:: \mathit{Bool} \to \{\mathit{Seq}\} \to \mathit{Chart} \to \{\mathit{Seq}\}$

  $\texttt{cut}\ 0\ \mathfrak{ys}\ \mathit{zs} = \{\ \mathfrak{a} \to \mathfrak{c} \mid \mathfrak{a} \to \mathfrak{b} \in \mathfrak{ys}\ \&\ \mathfrak{b} \to \mathfrak{c} \in \mathit{zs}\ \}$

  $\texttt{cut}\ 1\ \mathfrak{ys}\ \mathit{zs} = \{\ \mathfrak{a} \to \mathfrak{c} \mid \mathfrak{a} \to \mathfrak{b} \in \mathit{zs}\ \&\ \mathfrak{b} \to \mathfrak{c} \in \mathfrak{ys}\ \}$

Let us write the set of sequents generated by each subclause of `infer` as proof trees, for readability. This means that an output like

$$\frac{\mathfrak{a}' \to \mathfrak{c}' \in \mathfrak{as} \quad \ldots}{\mathfrak{a} \to \mathfrak{c}}\ \text{if } \mathfrak{p}$$

should be read as

$$\{\ a \to c \mid a' \to c' \in as\ \&\ \ldots\ \&\ p\ \}$$

The function `infer` takes in input a polarized formula, the set of subformulas and the set of sequents currently in the chart and returns a set of sequents. In a way analogous to Definition 5.9, if the input formula is positive, this will be the succedent of the output sequents, if it is negative, it will be the antecedent. Furthermore, we require the antecedent (resp. succedent) of a positive (resp. negative) instance of `infer` to be in the set of subformulas in the case of the monotonicity inferences. However, unlike in Algorithm 7.1, no particular restriction is imposed to the premises.

- $\texttt{infer} :: (\mathsf{Bool}, \mathcal{F}) \to \mathsf{Subform} \to \mathsf{Chart} \to \{\mathsf{Seq}\}$

$$\texttt{infer}\ (\_, a)\ \_\ \_ = \{\ a \to a\ \}, \text{ if } a \text{ is an atom.}$$

$$\texttt{infer}\ (1, a' \otimes b')\ \Sigma\ zs = \frac{a \to a' \in zs \quad b \to b' \in zs}{a \otimes b \to a' \otimes b'} \text{ if } a \otimes b \in \Sigma$$

$$\texttt{infer}\ (0, a \otimes b)\ \Sigma\ zs = \frac{a \to a' \in zs \quad b \to b' \in zs}{a \otimes b \to a' \otimes b'} \text{ if } a' \otimes b' \in \Sigma$$
$$\cup$$
$$\frac{a \to c/b' \in zs \quad b \to b' \in zs}{a \otimes b \to c}$$
$$\cup$$
$$\frac{a \to a' \in zs \quad b \to a' \backslash c \in zs}{a \otimes b \to c}$$

$$\texttt{infer}\ (1, b \backslash a)\ \Sigma\ zs = \frac{a' \to a \in zs \quad b \to b' \in zs}{b' \backslash a' \to b \backslash a} \text{ if } b' \backslash a' \in \Sigma$$
$$\cup$$
$$\frac{b \to a'/c \in zs \quad a' \to a \in zs}{c \to b \backslash a}$$
$$\cup$$
$$\frac{b \to b' \in zs \quad b' \otimes c \to a \in zs}{c \to b \backslash a}$$

$$\texttt{infer}\ (0, b' \backslash a')\ \Sigma\ zs = \frac{a' \to a \in a \quad b \to b' \in\ zs}{b' \backslash a' \to b \backslash a} \text{ if } b \backslash a \in \Sigma$$

The proof of Proposition 7.3 clarifies why the subformula test is applied only to the monotonicity inferences in Algorithm 7.2: in any other case, we know in advance that the formulas in the conclusion are subformulas of the goal sequent, since one formula is the trigger of the rule and the other occurs as a subformula of one of the premises.

We prove the correctness of Algorithm 7.2 by proving first its soundness and then its completeness.

Proposition 7.4. If Algorithm 7.2 is applied to a sequent $\mathtt{a} \to \mathtt{c}$, then, upon termination, if $\mathtt{x} \to \mathtt{y}$ is in the chart variable $\mathtt{zs}$, then $\vdash_{\mathcal{G}} \mathtt{x} \to \mathtt{y}$.

Proof. This follows immediately from the correspondence between the inference rules of $\mathcal{G}$ and those of Algorithm 7.2, and from the bottom-up approach of Algorithm 7.2 which is consistent with the interpretation of the rules of $\mathcal{G}$. Thus all the sequents in the chart are derivable in $\mathcal{G}$. □

Completeness of Algorithm 7.2 requires that every provable sequent $\mathtt{a} \to \mathtt{c}$ such that $\mathtt{a}$ is a negative subformula and $\mathtt{c}$ is a positive subformula of the goal sequent is in the chart at the end of the computation. Observe that we can be more restrictive and require that $\mathtt{a} \to \mathtt{c}$ is in chart either at the $\mathtt{a}$-iteration or at the $\mathtt{c}$-iteration of `exhaust-agenda`, where by $\mathtt{x}$-*iteration* we mean the iteration triggered by the polarized formula $\mathtt{x}$ of the agenda. Observe however that if $|\mathtt{a}| < |\mathtt{c}|$, then at the $\mathtt{a}$-iteration $\mathtt{a} \to \mathtt{c}$ will *not* be in the chart. Thus, we shall prove that $\mathtt{a} \to \mathtt{c}$ is in the chart at the $\mathtt{c}$-iteration. This forces us to claim the existence of a subformula $\mathtt{b}$ of the goal sequent, $|\mathtt{b}| < |\mathtt{a}|$ or $\mathtt{b} \equiv \mathtt{a}$, such that at the $\mathtt{a}$-iteration $\mathtt{a} \to \mathtt{b}$ is put in the chart and at the $\mathtt{c}$-iteration $\mathtt{b} \to \mathtt{c}$ is found by `infer` and $\mathtt{a} \to \mathtt{c}$ is obtained by `cut`. We call such a formula $\mathtt{b}$ bridge formula. Let us write $\mathtt{x}^+$, for $(1, \mathtt{x})$ (a positive formula) and $\mathtt{x}^-$, for $(0, \mathtt{x})$ (a negative formula).

Definition 7.5. Let $\mathtt{a} \to \mathtt{c}$ be a sequent provable in $\mathcal{G}$. A *bridge formula* for $\mathtt{a} \to \mathtt{c}$ is a formula $\mathtt{b}$, such that $\mathtt{b}^- \in \varphi(\mathtt{a}^-)$ and $\mathtt{b}^+ \in \varphi(\mathtt{c}^+)$ and $\vdash_{\mathcal{G}} \mathtt{a} \to \mathtt{b}$ and $\vdash_{\mathcal{G}} \mathtt{b} \to \mathtt{c}$.

What we are calling bridge formula is similar to what [Roorda, 1991, 1994] calls *interpolant* . However, a bridge formula for a sequent $\mathtt{a} \to \mathtt{c}$ is a subformula of $\mathtt{a} \to \mathtt{c}$, which is not necessarily the case for Roorda's interpolants[4].

Clearly, for each provable sequent $\mathtt{a} \to \mathtt{c}$, there is always at least one bridge formula. Let us give a few examples.

Example 7.2.

For $\mathtt{a} \otimes \mathtt{a}\backslash\mathtt{b} \to \mathtt{b}$, $\mathtt{b}$ is a bridge formula (as $\vdash \mathtt{a} \otimes \mathtt{a}\backslash\mathtt{b} \to \mathtt{b}$ and $\vdash \mathtt{b} \to \mathtt{b}$).

[4]We observe that Roorda's interpolation is more closely related to the procedure in Proposition 5.8: given a sequent $\mathtt{a} \to \mathtt{c}$ the formulas in $\mathtt{r}(\mathtt{a}) \cap \mathtt{e}(\mathtt{c})$ are all interpolants in the sense of Roorda.

For $a \otimes a\backslash b \to c/(b\backslash c)$, b is a bridge formula (as $\vdash a \otimes a\backslash b \to b$ and $\vdash b \to c/(b\backslash c)$).

For $a\otimes(b/(a\backslash b))\backslash b \to b/(a\backslash b)\otimes a\backslash b$, both $a\otimes(b/(a\backslash b))\backslash b$ and $b/(a\backslash b)\otimes a\backslash b$ are bridge formulas.

We now prove the existence of bridge formulas for every derivable sequent.

Proposition 7.5. Let $a \to c$ be given and $\Sigma = \sigma(a \to c)$. Let D be a quasinormal deduction of $a \to c$ such that all formulas appearing in D belong to Σ. Then there is a bridge formula b for $a \to c$.

Proof. Induction on D.

- If D consists of an axiom $a \to a$, then we take a.
- If D is a normal, reducing deduction, then we take c.
- If D is a normal, expanding deduction, then we take a.
- If D is normal, then there is a formula b such that $a \xrightarrow{r} b$ and $b \xrightarrow{e} c$. The, b is a bridge formula for $a \to c$
- If D ends in
$$\frac{a' \to c' \quad c'' \to a''}{a'/a'' \to c'/c''}$$
then, we have a bridge formula b'/b'' for $a'/a'' \to c'/c''$ made out of some bridge formulas b' for $a' \to c'$ and b'' for $c'' \to a''$.
- The other cases obtained trough monotonicity are similar.
- If D ends in lifting
$$\frac{c'' \to a\backslash c^\circ \quad c^\circ \to c'}{a \to c'/c''}$$
then there is an assumption $a \to a'$ among those from which $c'' \to a\backslash c^\circ$ is obtained. By induction hypothesis, we have a bridge formula b for $a \to a'$. Then b is a bridge formula for $a \to c'/c''$.
- The case of application and coapplication are similar.

□

Given polarized formulas x and y in the agenda, we write $x \prec y$, if x occurs in the agenda before y.

Proposition 7.6. Let a sequent $a \to c$ be given. Then, if $\vdash_g x \to y$, $x^-, y^+ \in \varphi(0, a)\cup\varphi(1, c)$, then $x \to y$, is in the chart variable zs either at the x-iteration or at the y-iteration of the procedure `exhaust-agenda`.

Proof. Induction on the deduction of $x \to y$.

1. $\mathtt{x} \to \mathtt{y}$ is the conclusion of monotonicity.

$$\frac{\mathtt{x}' \to \mathtt{y}' \quad \mathtt{y}'' \to \mathtt{x}''}{\mathtt{x}'/\mathtt{x}'' \to \mathtt{y}'/\mathtt{y}''}$$

   There are two possibilities:

   a) $\mathtt{y} \prec \mathtt{x}$, we have the following subcases:
      i. $\mathtt{x}', \mathtt{x}'' \prec \mathtt{y}$, then at the $\mathtt{y}$-iteration $\mathtt{x}' \to \mathtt{y}'$ and $\mathtt{y}'' \to \mathtt{x}''$ are in the chart $\mathtt{zs}$ and the conclusion is obtained by `infer` $\mathtt{y}$ $\mathtt{zs}$.
      ii. $\mathtt{y} \prec \mathtt{x}', \mathtt{x}''$, then there are bridge formulas $\mathtt{z}'$ and $\mathtt{z}''$ such that, at the $\mathtt{y}$-iteration, $\mathtt{z}' \to \mathtt{y}'$ and $\mathtt{y}'' \to \mathtt{z}''$ are in the chart. Thus, at the $\mathtt{x}'$-iterations, we obtain $\mathtt{x}' \to \mathtt{z}'$ and also $\mathtt{x}' \to \mathtt{y}'$ by `cut`, while, at the $\mathtt{x}''$-iterations, we obtain $\mathtt{z}'' \to \mathtt{x}''$ and also $\mathtt{y}'' \to \mathtt{x}''$ by `cut`. We conclude $\mathtt{x}'/\mathtt{x}'' \to \mathtt{y}'/\mathtt{y}''$ at the $\mathtt{x}$-iteration by `infer` $\mathtt{x}$ $\mathtt{zs}$.
      iii. The other cases are similar.
   b) $\mathtt{x} \prec \mathtt{y}$, dual of $\mathtt{y} \prec \mathtt{x}$.

2. The other monotonicity cases are similar.

3. $\mathtt{x} \to \mathtt{y}$ is the conclusion of lifting.

$$\frac{\mathtt{y}'' \to \mathtt{x} \backslash \mathtt{y}^\circ \quad \mathtt{y}^\circ \to \mathtt{y}'}{\mathtt{x} \to \mathtt{y}'/\mathtt{y}''}$$

   We shall consider the following cases.

   a) $\mathtt{y} \prec \mathtt{x}$, that is, $\mathtt{x}$ has not yet been computed. Then, there is a bridge formula $\mathtt{z}$, such that the conclusion of

$$\frac{\mathtt{y}'' \to \mathtt{z} \backslash \mathtt{y}^\circ \quad \mathtt{y}^\circ \to \mathtt{y}'}{\mathtt{z} \to \mathtt{y}'/\mathtt{y}''}$$

      is in the chart $\mathtt{zs}$ at the $\mathtt{y}$-iteration by `infer` $\mathtt{y}$ $\mathtt{zs}$. Thus $\mathtt{x} \to \mathtt{z}$ is in the chart at the $\mathtt{x}$-iteration. We conclude $\mathtt{x} \to \mathtt{y}$ by `cut` at the $\mathtt{x}$-iteration.
   b) $\mathtt{x} \prec \mathtt{y}$, we shall consider the following subcases.
      i. $\mathtt{y}'' \prec \mathtt{x}$, then for some bridge formula $\mathtt{z}$, $\mathtt{y}'' \to \mathtt{z} \backslash \mathtt{y}^\circ$ is in the chart at the $\mathtt{y}''$-iteration and $\mathtt{x} \to \mathtt{z}$ at the $\mathtt{x}$-iteration. We obtain $\mathtt{x} \to \mathtt{y}'/\mathtt{y}''$ at the $\mathtt{y}$-iteration by `cut`.
      ii. Otherwise, $\mathtt{y}'' \to \mathtt{x} \backslash \mathtt{y}^\circ$ and $\mathtt{y}^\circ \to \mathtt{y}'$ are already in the cart at the $\mathtt{y}$-iteration, and the conclusion $\mathtt{x} \to \mathtt{y}$ is obtained by `infer` $\mathtt{y}$ $\mathtt{zs}$.

4. Application and coapplication are similar.

5. $x \to y$ is the conclusion of cut.

$$\frac{x \to z \quad z \to y}{x \to y}$$

Then z is a bridge formula.

□

We examine now the complexity of Algorithm 7.2.

Proposition 7.7. If Algorithm 7.2 is applied to a sequent $a \to c$ of length n, then the function `exhaust-agenda` terminates after executing $O(n^5)$ operations.

Proof. The number of polarized subformulas (size of the agenda) is bounded by n. Due to the subformula property, the size of the chart is bounded by n^2. Thus, at each iteration of `exhaust-agenda`, the application of the rules `cut` and `infer` may involve $O(n^4)$ steps. Since there are $O(n)$ iterations, we conclude that `exhaust-agenda` will compute $O(n^5)$ operations. □

7.4 Connection to parsing

Algorithm 7.2 terminates after computing all possible inferences among polarized subformulas of the input sequent, as we proved in Proposition 7.6. However the procedure can be generalized as to apply to an agenda containing any list of polarized formulas. The result is, again, the set of all derivable sequent made of formulas in the agenda. Let us see how this set may be used in relation to *parsing*, that is to find a derivation of a sequent $a_1, \ldots, a_n \to c$ whose structure is not given.

Consider the following rules.

Definition 7.6. Generalized rules:

Generalized cancellation rules.

$$\frac{\Gamma \to c/a' \quad \Delta \to a}{\Gamma\Delta \to c} \text{ if } \vdash_{NL} a \to a'$$

$$\frac{\Gamma \to a \quad \Delta \to a'\backslash c}{\Gamma\Delta \to c} \text{ if } \vdash_{NL} a \to a'$$

Product rule.

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

Given the rules in 7.6 we can define NL recognition as follows.

Proposition 7.8. Let $a_1, \ldots, a_n \to c$ be a sequent provable in NL. Then, for some c' such that $\vdash_{NL} c' \to c$, the sequent $a_1, \ldots, a_n \to c'$ can be proved only by means of the rules given in Definition 7.6.

The following proposition connects recognition of two formula sequents, as defined in Algorithm 7.2 to parsing.

Proposition 7.9. Let $a_1, \ldots, a_n \to c$ be a sequent provable in NL. Let us apply `exhaust-agenda`, from Algorithm 7.2, to the agenda containing the sorted list obtained from $\varphi(a_0^-) \cup \ldots \cup \varphi(a_n^-) \cup \varphi(c^+)$ and to the empty chart. Let Z be the chart variable after termination of `exhaust-agenda`. Then, for some c' such that $c' \to c \in Z$, the sequent $a_1, \ldots, a_n \to c'$ can be proved through the following rules:

Generalized cancellation rules.

$$\frac{\Gamma \to c/a' \quad \Delta \to a}{\Gamma\Delta \to c} \text{ if } a \to a' \in Z$$

$$\frac{\Gamma \to a \quad \Delta \to a'\backslash c}{\Gamma\Delta \to c} \text{ if } a \to a' \in Z$$

Generalized product rule.

$$\frac{\Gamma \to a' \quad \Delta \to b'}{\Gamma\Delta \to a \otimes b} \begin{cases} \text{if } a' \to a \in Z \\ \text{and } b' \to b \in Z \end{cases}$$

More in general, Algorithm 7.2 can perform a form of preprocessing on the input grammar.

Given a set of sequents Z, let us call $AB_Z^{\otimes}$ the deductive system whose inference rules are the generalized cancellation and product rules of Proposition 7.9.

Proposition 7.10. Let G be a NL grammar $\langle V_t, s, Lex, NL\rangle$. Let us apply the function `exhaust-agenda`, from Algorithm 7.2, to the agenda containing the sorted list obtained from $\varphi(a_0^-) \cup \ldots \cup \varphi(a_n^-)$, where $w \to a_i \in Lex$ for all i, $0 \leqslant i \leqslant n$ and to the empty chart. Let Z be the chart variable after termination of `exhaust-agenda`. Then, we can construct a new grammar $G' = \langle V_t, s, Lex, AB_Z^{\otimes}\rangle$, such that $L_t(G) = L_t(G')$.

We call the set Z in the preceding construction *grammar inference set*, *gis* for short.

Example 7.3. We propose once more grammar A_6.

$$\begin{array}{lcl} n^* & \to & n \,|\, s/(n\backslash s) \,|\, tv\backslash(s/(n\backslash s))\backslash s \,|\, (s/(n\backslash s) \otimes tv)\backslash s \\ tv^* & \to & tv \\ hv^* & \to & (s/(n\backslash s))\backslash s \end{array}$$

The *gis* of A_6 is the following set of sequents:

$$
\begin{array}{l}
\{\ \mathfrak{n} \to \mathfrak{n},\\
\ \ \mathfrak{s} \to \mathfrak{s},\\
\ \ \mathfrak{n}\backslash\mathfrak{s} \to \mathfrak{n}\backslash\mathfrak{s},\\
\ \ (\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n} \to (\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n},\\
\ \ \mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}) \to \mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}),\\
\ \ \mathfrak{n} \to \mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}),\\
\ \ (\mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}))\backslash\mathfrak{s} \to \mathfrak{n}\backslash\mathfrak{s},\\
\ \ (\mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}))\backslash\mathfrak{s} \to (\mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}))\backslash\mathfrak{s},\\
\ \ \mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}) \otimes (\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n} \to \mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}) \otimes (\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n},\\
\ \ (\mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}) \otimes (\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n})\backslash\mathfrak{s} \to (\mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}) \otimes (\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n})\backslash\mathfrak{s},\\
\ \ ((\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n})\backslash(\mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}))\backslash\mathfrak{s} \to ((\mathfrak{n}\backslash\mathfrak{s})/\mathfrak{n})\backslash(\mathfrak{s}/(\mathfrak{n}\backslash\mathfrak{s}))\backslash\mathfrak{s}\ \}
\end{array}
$$

As we know that the inference engine of Algorithm 7.2 generates only sequents made of subformulas of the formulas in the agenda, we may easily calculate the size of the grammar inference set.

Proposition 7.11. Let a NL grammar $\mathsf{G} = \langle V_t, \mathfrak{s}, \mathsf{Lex}, \mathsf{NL}\rangle$ be given. Let Z be the *gis* of G and $\mathfrak{m}$ the size of the set containing all subformulas of the formulas appearing in the lexicon Lex. Then

1. $|\mathsf{Z}| = O(\mathfrak{m}^2)$,
2. Z can be constructed in $O(\mathfrak{m}^5)$ steps.

Proof. The statement in 1 depends on the fact that all sequents in Z are made of formulas which are subformulas of the formulas in Lex. The statement in 2 is a consequence of Proposition 7.7. □

The asymptotic bounds stated in Proposition 7.11 are pleasant properties of the grammar inference set. We may also observe that many elements of the *gis* are identities, whose interaction with the rules is entirely predictable without looking at the *gis* (in case of identity sequents in the side conditions, the generalized rules boil down to the $\mathsf{AB}^{\otimes}$ rules). Thus we may prefer to prune the grammar inference set of all sequents of the form $\mathfrak{x} \to \mathfrak{x}$ and move the identity test among the side conditions of the generalized rules given above. We omit the implementation these simple operations.

Polynomial parsing for NL

All the parsing systems for $\mathsf{AB}^{\otimes}$ grammars developed in Chapter 3 can easily be modified to work with $\mathsf{AB}^{\otimes}$ grammars based on a grammar inference set Z. We formulate here the system $\mathcal{NL}_{\mathtt{Mix}}$.

Let us formulate, first of all, the deductive system $\widetilde{\mathsf{AB}}^{\otimes}_{\mathsf{Z}}$.

Definition 7.7. Let a set of sequents Z be given. We call $\widetilde{\mathsf{AB}}^{\otimes}_{\mathsf{Z}}$ the triple $\langle \mathcal{F}, \widetilde{\mathsf{AX}}^{\star}, \mathtt{Cut}' \rangle$ such that $\widetilde{\mathsf{AX}}^{\star}$ is as in $\widetilde{\mathsf{AB}}^{\otimes}$ and $\mathtt{Cut}'$ is the following rule

$$\frac{\Gamma \to \mathfrak{a} \quad \Delta[\mathfrak{a}'] \to \mathfrak{c}}{\Delta[\Gamma] \to \mathfrak{c}} \begin{cases} \text{if } \mathfrak{a} \to \mathfrak{a}' \in \mathsf{Z} \\ \text{or } \mathfrak{a} = \mathfrak{a}' \end{cases}$$

On the basis of $\widetilde{\mathsf{AB}}^{\otimes}_{\mathsf{Z}}$ we define the parsing system $\mathcal{NL}_{\mathtt{Mix}}$ as follows.

Definition 7.8. Let G be a NL grammar. Let Z be the grammar inference set of G and G' the $\widetilde{\mathsf{AB}}^{\otimes}_{\mathsf{Z}}$ grammar corresponding to G according to the construction in Proposition 7.10 on page 147. The system $\mathcal{NL}_{\mathtt{Mix}}$ is a triple $\langle \mathcal{I}, \mathcal{A}, \mathcal{R} \rangle$ such that $\mathcal{I}$, $\mathcal{A}$ and $\mathcal{R}$ are as in $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$ except for the fact that $\mathcal{R}$ has the following ϵ-scanning and completion rules in place of the ϵ-scanning and completion rules of $\mathcal{AB}^{\otimes}_{\mathtt{Mix}}$.

$$\frac{(\mathfrak{i}, \Delta \triangleright \mathfrak{a}'\, \Gamma \to \mathfrak{c},\, \mathfrak{j})}{(\mathfrak{i}, \Delta\, \mathfrak{a}' \triangleright \Gamma \to \mathfrak{c},\, \mathfrak{j})} \text{ if } \vdash_{\mathsf{G}'} \epsilon \to \mathfrak{a}'$$

$$\frac{(\mathfrak{i}, \Gamma\, \mathfrak{a}' \triangleleft \Delta \to \mathfrak{c},\, \mathfrak{j})}{(\mathfrak{i}, \Gamma \triangleleft \mathfrak{a}'\, \Delta \to \mathfrak{c},\, \mathfrak{j})} \text{ if } \vdash_{\mathsf{G}'} \epsilon \to \mathfrak{a}'$$

$$\frac{(\mathfrak{i}, \Delta \triangleright \mathfrak{a}'\, \Gamma \to \mathfrak{c}, \mathfrak{k}) \quad (\mathfrak{k}, \Lambda \to \mathfrak{a}, \mathfrak{j})}{(\mathfrak{i}, \Delta\, \mathfrak{a}' \triangleright \Gamma \to \mathfrak{c}, \mathfrak{j})} \begin{cases} \text{if } \mathfrak{a} \to \mathfrak{a}' \in \mathsf{Z} \\ \text{or } \mathfrak{a} = \mathfrak{a}' \end{cases}$$

$$\frac{(\mathfrak{i}, \Lambda \to \mathfrak{a}, \mathfrak{k}) \quad (\mathfrak{k}, \Delta\, \mathfrak{a}' \triangleleft \Gamma \to \mathfrak{c}, \mathfrak{j})}{(\mathfrak{i}, \Delta \triangleleft \mathfrak{a}'\, \Gamma \to \mathfrak{c}, \mathfrak{j})} \begin{cases} \text{if } \mathfrak{a} \to \mathfrak{a}' \in \mathsf{Z} \\ \text{or } \mathfrak{a} = \mathfrak{a}' \end{cases}$$

7.5 Conclusion

This chapter presented parsing methods for the non-associative Lambek calculus. The result of polynomial parsability for this logic is not new. However we wish to remark the following benefits of our methods with respect to the results that can be found in the literature.

[De Groote, 1999] proved that two-formula sequents of NL can be recognized in polynomial time. On the other hand, [de Groote, 1999] and [de Groote and Lamarche, 2002] gives only vague specifications on how the recognition procedure should be implemented. A special requirement of de Groote's method is that "the search space is organized as a DAG rather than as a tree". In this respect, we explicitly described how we can define an agenda-driven chart-based procedure (see Chapter 4) for logical parsing. Furthermore, de Groote gives no degree of polynomiality, although [Moot, 2002] states that [de Groote and Lamarche, 2002] "prove that, given a bracketed input, the decision problem for NL can be solved in $O(n^6)$ time". This calculation is based on the number of

possible subformulas and subcontexts available at each application of the *context rules*. Hence our recognition procedure for two-formula sequents is more efficient than one based on contexts.

[Buszkowski, 2005] and [Bulińska, 2006] give a constructive method for proving NL sequents in polynomial time. Our approach presents similarities to these works. In fact, here as there, recognition of a sequent $a_1, \ldots, a_n \to c$ passes through the construction of a set of sequents which allows the application of extended context-free parsing methods. However, our grammar inference set is *smaller* than the set of *basic sequents* employed by Buszkowski and Bulińska as it consists only of *two-formula* sequents made of polarized subformulas of formulas of the lexicon and can be constructed much faster than the $O(n^{18})$ of Buszkowski and the $O(n^{12})$ of Bulinska[5]. Our recognition method guarantees that once the *gis* is constructed, what has to be done once and for all, the complexity of recognition is $O(m^2n^3)$, where m is the size of the lexicon, calculated as in Proposition 7.11 on page 148.

A final remark concerns the extension of our polynomial recognition algorithm to the system NG presented at the end of Section 2.8, that is NL extended with symmetric operators. Clearly, the addition to $\mathcal{G}$ of the dual rules, in Definition 7.9 below, preserves all the properties which were relevant to the design of Algorithm 7.2, namely goal orientation of rule application and subformula property.

Definition 7.9. Dual rules:

$$\frac{a \to a' \quad b \to b'}{a \oplus b \to a' \oplus b'}$$

$$\frac{a' \to a \quad a' \oslash c \to b}{c \to a \oplus b} \qquad \frac{c \odot b' \to a \quad b' \to b}{c \to a \oplus b}$$

$$\frac{a' \to a \quad b \to b'}{b' \oslash a' \to b \oslash a} \qquad \frac{a' \to a \quad b \to b'}{a' \odot b' \to a \odot b}$$

$$\frac{a' \odot c \to b \quad a \to a'}{a \oslash c \to b} \qquad \frac{c \oslash a' \to b \quad a \to a'}{a \odot b \to c}$$

$$\frac{b' \to b \quad a \to b' \oplus c}{b \oslash a \to c} \qquad \frac{b' \to b \quad a \to c \oplus b'}{a \odot b \to c}$$

[5]In fact, Buszkowski and Bulińska consider NL extended with non-logical axioms and empty antecedent, respectively. However, even without such extensions their sets of basic sequents would be bigger than the *gis*. Essentially, this depends on the stronger notion of subformula that we adopted in our construction.

The extension of Algorithm 7.2 to this new system is trivial and we conclude this chapter with the following result.

Proposition 7.12. Two-formula sequents of the system NG can be recognized in polynomial time.

Chapter 8

Conclusion and Further Lines of Research

This book is a study of the logical and computational properties of structure-preserving categorial grammars, namely non-associative Lambek grammars with product and Ajdukiewicz–Bar-Hillel grammars with product. Chapter 3 presents parsing systems for AB grammars with product which are implemented in Chapter 4 as cubic time parsing algorithms. These chapters show two important facts. In the first place, that the product introduction rule can be handled elegantly and efficiently. Secondly that, due to the use of abstract inference schemes, categorial parsing can be more efficient than context-free parsing.

In Chapter 5, we examined a simple procedure for automatic construction of normal deductions in the non-associative Lambek calculus. This procedure is based on two functions which operate by recursively contracting suitable patterns. We have seen that the problem of proving a two formula sequent $a \to c$ boils down to that of finding a formula b such that $b \in r(a)$ and $b \in e(c)$. The procedure extends to the more general problem of proving a 'multi-formula' sequent $a_1 \dots a_n \to c$: if $\vdash_{\mathsf{NL}} a_1 \dots a_n \to c$, then $\vdash_{\mathsf{AB}\otimes} a'_1 \dots a'_n \to c'$, where $a'_i \in r(a_i)$, $1 \leqslant i \leqslant n$ and $c' \in e(c)$.

In Chapter 6, we have addressed the problem of spurious ambiguity, and shown that the procedure in Definition 5.9 is not affected by this problem. As the algorithms for $\mathsf{AB}^{\otimes}$ are not affected by this problem either, we have indirectly designed a general normal form parser for non-associative Lambek grammars.

The expansion and reduction method in Definition 5.9 and in Proposition 5.8 enabled us to determine the degree of ambiguity of NL sequents, which we stated in Proposition 6.7 on page 130.

Chapter 7 has shown how NL parsing can be executed in polynomial time. This result is not new. However, our approach has some advantages over the previous ones. With respect to [de Groote, 1999], it provides a fully explicit

procedure both for two-formula and for multi-formula sequents. With respect to [Buszkowski, 2005], we lowered notably the polynomial bound.

An obvious question is whether the recognition methods developed in this thesis can be extended to other logics and in particular to logics *with* structural rules.

Normal derivations had been proposed in the second half of the 1980s (by Buszkowski [1986] and Kandulski [1988]) as a method for proving context-freeness of NL grammars. Since at that time the problem of context-freeness of the grammars based on the *associative* Lambek calculus was still open, it was also reasonable to try to extend the methods used for the non-associative system to the associative calculus. However, no attempt in this direction can be found in the literature. On one hand, it might be the case that a normalization parallel to the one for NL was not possible for the calculus of [Lambek, 1958]. On the other, that, while possible, it was not enough to prove context-freeness.

Indeed, we believe that it is possible to extend the normalization procedure to logical systems with structural rules. As in the case of NL, the expedient would be to adopt a non-minimal axiomatization, which allows to interpret the derivation process as divided into a reducing component and an expanding component. However, further research should be dedicated to this topic.

Bibliography

Adams, S. (1993). Functional pearls: Efficient sets–a balancing act. *Journal of Functional Programming*, 3(4):553–561.

Ades, A. and Steedman, M. (1982). On the order of words. *Linguistics and Philosophy*, 4:517–558.

Aho, A. and Ullman, J. (1972). *The Theory of Parsing, Translation and Compiling*, volume 1: Parsing. Prentice-Hall, INC.

Ajdukiewicz, K. (1935). Die syntaktische Konnexität. *Studia Philosophica*, 1:1–27.

Andreoli, J.-M. (1992). Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347.

Andreoli, J.-M. (2001). Focussing and proof construction. *Annals of Pure and Applied Logic*, 107(1-3):131–163.

Baldridge, J. (2002). *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh.

Bar-Hillel, Y. (1953). A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58.

Bar-Hillel, Y., Gaifman, C., and Shamir, E. (1964). On categorial and phrase structure grammars. In Bar-Hillel, Y., editor, *Language and Information. Selected Essays on their Theory and Application*, pages 99–115. Addison-Wesley, Reading, MA.

Bernardi, R. (2002). *Reasoning with polarity in categorial type logic*. PhD thesis, UiL-OTS, Utrecht.

Bernardi, R. and Moortgat, M. (2007). Continuation semantics for symmetric categorial grammar. In *Proceedings of the Fourteenth Workshop on Logic, Language, Information and Computation (WoLLIC'2007)*, Rio de Janeiro.

Blackburn, P. and Bos, J. (2003). Computational semantics for natural language. Course notes for NASSLLI 2003, Indiana University.

Blackburn, P., de Rijke, M., and Venema, Y. (2001). *Modal Logic*. Cambridge University Press, Cambridge, England.

Bulińska, M. (2006). P-TIME decidability of NL1 with assumptions. In *FG2006: The 11th Conference on Formal Grammar*, pages 29–38.

Buszkowski, W. (1986). Generative capacity of the nonassociative Lambek calculus. *Bulletin of the Polish Academy of Sciences, Mathematics*, 34:507–516.

Buszkowski, W. (1987). The logic of types. In Srzednicki, J., editor, *Initiatives in Logic*, pages 180–206. Martinus Nijhoff, Dodrecht.

Buszkowski, W. (1988). Gaifman's theorem on categorial grammars revisited. *Studia Logica*, 47:23–33.

Buszkowski, W. (1997). Mathematical linguistics and proof theory. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*, pages 683–736. Elsevier, Amsterdam.

Buszkowski, W. (2005). Lambek calculus with nonlogical axioms. In Casadio, C., Scott, P., and Seely, R., editors, *Language and Grammar: Studies in Mathematical Linguistics and Natural Language*, pages 77–93. CSLI Lecture Notes 168, Stanford.

Carpenter, B. (1996). The turing-completeness of multimodal categorial grammars.

Carpenter, B. and Morrill, G. (2005). Switch graphs for parsing type logical grammars. In *Proceedings of the International Workshop on Parsing Technology, IWPT05*, Vancouver.

Chomsky, N. (1957). *Syntactic Structures*. Mouton and Co., The Hague.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2(2):137–167.

Cormen, T., Leiserson, C., and Rivest, R. (1990). *Introduction to Algorithms*. MIT Press.

Curry, H. B. and Feys, R. (1958). *Combinatory Logic I*. North-Holland, Amsterdam.

Damas, L. and Milner, R. (1982). Principal type-schemes for functional programs. In *POPL*, pages 207–212.

de Groote, F. (1999). The non-associative Lambek calculus with product in polynomial time. In Murray, N. V., editor, *Lecture Notes in Artificial Intelligence*, volume 1617. Springer-Verlag.

de Groote, P. and Lamarche, F. (2002). Classical non-associative Lambek calculus. *Studia Logica*, 71(3):355–388.

Došen, K. (1988). Sequent systems and groupoid models, part 1. *Studia Logica*, 47:353–386.

Došen, K. (1992). A brief survey of frames for the Lambek calculus. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 38:179–187.

Earley, J. (1968). *An Efficient Context-Free Parsing Algorithm*. PhD thesis, Carnegie-Mellon University, Pittsburgh, PA.

Earley, J. (1970). An efficient context-free parsing algorithm. *Comm. ACM*, 13:94–102.

Finkel, A. and Tellier, I. (1996). A polynomial algorithm for the membership problem with categorial grammar. *Theoretical Computer Science*, 164:207–221.

Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell.

Graham, S. L., Harrison, M., and Ruzzo, W. L. (1980). An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2(3):415–462.

Grishin, V. N. (1983). On a generalization of the Ajdukiewicz-Lambek system. *Studies in Nonclassical Logics and Formal Systems*, pages 315–334.

Hankin, C. (2004). *An Introduction to Lambda Calculi for Computer Scientist*. King's College Publications.

Harrison, M. A. (1978). *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Massachussets.

Hendriks, H. (1993). *Studied Flexibility: Categories and Types in Syntax and Semantics*. PhD thesis, ILLC, Amsterdam.

Hendriks, H. (1999). The logic of tune. A proof-theoretic analysis of intonation. In Lecomte, A., editor, *Logical Aspects of Computational Linguistics*, New York. Springer.

Hepple, M. (1994). Discontinuity and the Lambek calculus. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING'94)*, pages 1235–1239.

Hepple, M. (1996). A compilation-chart method for linear categorial deduction. In *Proceedings of COLING-96*, pages 537–542, Copenhagen.

Hepple, M. (1999). 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.

Heylen, D. (1999). *Types and Sorts. Resource logic for feature checking.* PhD thesis, UiL-OTS, Utrecht.

Hindley, J. R. (1997). *Basic Simple Type Theory.* Cambridge University Press.

Howard, W. (1980). The formulas-as-types notion of construction. In *To H.B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*, pages 479–490. Academic Press.

Jones, S. P., editor (2003). *Haskell 98 Language and Libraries: The Revised Report.* Cambridge University Press.

Kandulski, M. (1988). The equivalence of nonassociative Lambek categorial grammars and context-free grammars. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 34:41–52.

Kasami, T. (1965). An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Res. Lab., Bedford Mass.

König, E. (1994). A hypothetical reasoning algorithm for linguistic analysis. *Journal of Logic and Computation*, 4(1):1–19.

Kraak, E. (1995). French object clitics: a multimodal analysis. In Morrill, G. and Oehrle, R. T., editors, *Formal Grammar.*

Kruijff, G.-J. M. and Baldridge, J. (2003). Multi-modal combinatory categorial grammar. In *EACL*, pages 211–218.

Kurtonina, N. (1995). *Frames and labels. A modal analysis of categorial inference.* PhD thesis, UiL-OTS, Utrecht.

Kurtonina, N. and Moortgat, M. (1997). Structural control. In Blackburn, P. and de Rijke, M., editors, *Specifying Syntactic Structures*, pages 75–113. CSLI, Stanford.

Lambek, J. (1958). The mathematic of sentence structure. *American Mathematical Monthly*, 65(3):154–170.

Lambek, J. (1961). On the calculus of syntactic types. In Jacobson, R., editor, *Proceedings of the Twelfth Symposium in Applied Mathematics*, volume XII, pages 166–178.

Lambek, J. (1993). Logic without structural rules (another look at cut elimination). In Schroeder-Heister, P. and Dosen, K., editors, *Substructural Logic*, pages 179–206. Claredon Press, Oxford.

Le Nir, Y. (2003a). From proof trees in lambek calculus to ajdukiewicz bar-hillel elimination binary trees. *Journal of Research on Language and Computation*, 1:3-4:181–201.

Le Nir, Y. (2003b). *Structures des analyses syntaxiques catégorielles. Application à l'inférence grammaticale*. PhD thesis, Université de Rennes 1, Rennes.

Le Nir, Y. (2004). From NL grammars to AB grammars. In Moortgat, M. and Prince, V., editors, *CG2004 Proceedings*, Montpellier-France.

Montague, R. (1970). English as a formal language. In *Linguaggi nella Società e nella Tecnica*, pages 189–224. Edizioni di Comunità, Milan. Reprinted in [**?**].

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

Moortgat, M. (1996). Multimodal linguistic inference. *Journal of Logic, Language and Information*, 5(3/4):349–385.

Moortgat, M. (1997). Categorial type logics. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*, pages 93–177. Elsevier, Amsterdam.

Moortgat, M. (1999). Constants of grammatical reasoning. In G.Bouma, Hinrichs, E., Kruijff, G.-J., and Oehrle, R. T., editors, *Constraints and Resources in Natural Language Syntax and Semantics*, pages 195–219. CSLI, Stanford.

Moortgat, M. (2007). Symmetries in natural language syntax and semantics: the Lambek-Grishin calculus. In *Proceedings of the Fourteenth Workshop on Logic, Language, Information and Computation (WoLLIC'2007)*, Rio de Janeiro.

Moortgat, M. and Oehrle, R. (1997). Proof nets for the grammatical base logic. In Abrusci, V. M. and Casadio, C., editors, *Proceedings of the IV Roma Workshop*, Roma. Bulzoni Editore.

Moot, R. (2002). *Proof Nets for Linguistic Analysis*. PhD thesis, UiL-OTS, Utrecht.

Morrill, G. (1994). *Type Logical Grammar: Categorial Logic of Signs*. Kluwer, Dordrecht.

Morrill, G. (1996). Memoisation of categorial proof nets: Parallelism in categorial processing. In Abrusci, V. M. and Casadio, C., editors, *Proofs and Linguistic Categories,* Proceedings 1996 Roma Workshop, pages 157–169, Bologna. Cooperativa Libraria Universitaria Editrice.

Nederhof, M.-J. and Satta, G. (2004). Tabular parsing. In Martin-Vide, C., Mitrana, V., and Paun, G., editors, *Formal Languages and Applications, Studies in Fuzziness and Soft Computing 148*, pages 529–549. Springer.

Okasaki, C. (1998). *Purely Functional Data Structures*. Cambridge University Press, Cambridge, England.

Okasaki, C. (1999). Red-black trees in a functional setting. *Journal of Functional Programming*, 9(4):471–477.

Pentus, M. (1993). Lambek grammars are context free. In *Proceedings of the 8th Annual IEEE Symposium on Logic in Computer Science*, pages 429–433, Los Alamitos, California. IEEE Computer Society Press.

Pentus, M. (1995). Models for the Lambek calculus. *Annals of Pure and Applied Logic*, 75(1–2):179–213.

Pentus, M. (2003). Lambek calculus is NP-complete. CUNY Ph.D. Program in Computer Science Technical Report TR–2003005, CUNY Graduate Center, New York. http://www.cs.gc.cuny.edu/tr/techreport.php?id=79.

Pentus, M. (2006). Lambek calculus is np-complete. *Theoretical Computer Science*, 357(1-3):186–201.

Pereira, F. C. N. and Shieber, S. N. (1987). *Prolog and Natural-Language Analysis*. Center for the Study of Language and Information, Stanford, California.

Pereira, F. C. N. and Warren, D. H. D. (1983). Parsing as deduction. In *Proceedings of 21st Annual Meeting of the Association for Computational Linguistics*. MIT.

Retoré, C. (2005). The logic of categorial grammars – lecture notes. Research Report 5703, INRIA. 108 pp.

Roorda, D. (1991). *Resource logics: proof-theoretical investigations*. PhD thesis, University of Amsterdam, Amsterdam.

Roorda, D. (1994). Interpolation in fragments of Classical Linear Logic. *Journal of Symbolic Logic*, 59(2):419–444.

Savateev, Y. (2006). The derivability problem for Lambek calculus with one division.

Shieber, S. M., Schabes, Y., and Pereira, F. C. N. (1995). Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

Sikkel, K. (1993). *Parsing Schemata*. PhD thesis, Department of Computer Science, University of Twente, Enschede.

Sikkel, K. (1998). Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 199.

Sikkel, K. and Nijholt, A. (1997). Parsing of context-free languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of formal languages, Vol. II*. Springer Verlag, Berlin.

Steedman, M. (2000a). Information structure and the syntax-phonology interface. *Linguistic Inquiry*, 31(4):649–689.

Steedman, M. (2000b). *The Syntactic Process*. The MIT Press.

Szczerba, M. (1997). Representation theorems for residuated groupoids. In Retoré, C., editor, *Proceedings of the 1st International Conference on Logical Aspects of Computational Linguistics (LACL-96)*, volume 1328 of *LNAI*, pages 426–434, Berlin. Springer.

Tiede, H.-J. (1998). Lambek calculus proofs and tree automata. In Moortgat, M., editor, *LACL*, volume 2014 of *Lecture Notes in Computer Science*, pages 251–265. Springer.

Tiede, H.-J. (1999a). Counting the number of proofs in the commutative Lambek calculus. In Gerbrandy, J., Marx, M., de Rijke, M., and Venema, Y., editors, *JFAK. Essays Dedicated to Johan van Benthem on the Occasion of his 50th Birthday*. Amsterdam University Press, Amsterdam.

Tiede, H.-J. (1999b). *Deductive Systems and Grammars: Proofs as Grammatical Structures*. PhD thesis, Indiana University.

van Benthem, J. (1991). *Language in Action: Categories, Lambdas and Dynamic Logic*. The MIT Press.

van Eijck, J. (2004). Deductive parsing in Haskell.

Vermaat, W. (2005). *The logic of variation. A cross-linguistic account of wh-question formation*. PhD thesis, UiL-OTS, Utrecht.

Vijay-Shanker, K. and Weir, D. J. (1990). Polynomial time parsing of combinatory categorial grammars. In *ACL*, pages 1–8.

Vijay-Shanker, K. and Weir, D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27.

Weir, D. J. and Joshi, A. K. (1988). Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In *Meeting of the Association for Computational Linguistics*, pages 278–285.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time $\mathfrak{n}^3$. *Information and Control*, 10:189–208.

Zielonka, W. (1981). Axiomatizability of Ajdukiewicz-Lambek calculus by means of cancellation schemes. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 27:215–224.

Index

Samenvatting

Dit boek bestudeert de logische en computationele eigenschappen van structuur behoudende categoriale grammatica's.

In het eerste deel van het boek worden zogenaamde "chart parsing" methodes besproken voor niet-associatieve categoriale grammatica's in de stijl van Ajdukiewicz en Bar-Hillel. Deze methodes worden in eerste instantie gentroduceerd als deductieve zinsontleders, daarmee wordt bedoeld dat ze worden gezien als deductieve systemen die gebruik maken van de lineaire volgorde van syntactische categorien. Vervolgens worden deze methodes geherformuleerd als efficinte zinsontledingsalgoritmes. Een belangrijk aspect is de formulering van efficinte methodes om met product formules in het zinsontleedproces om te kunnen gaan.

Het tweede deel van dit boek gaat over categoriale grammatica's in de stijl van Lambek. In hoofdstuk 5 wordt een simpele en elegante methode voor automatische herkenning geformuleerd. In de navolgende hoofdstukken worden de syntactische en semantische eigenschappen van deze methode besproken. Een verrassend resultaat is de relatie tussen het aantal semantische lezingen en de binominale cofficint, die wordt besproken in hoofdstuk 6. De resultaten voor polynominaliteit, in hoofdstuk 7, zijn gebaseerd op expliciete algoritmes die generalisaties maken en eerdere resultaten verbeteren.

De algoritmes in dit boek representeren de eerste volledige toepassing van "chart parsing"-technieken op logische grammatica's. Ze kunnen worden beschouwd als de grondslag voor de toekomstige ontwikkeling van rijkere logische grammatica's.

Curriculum vitae

Matteo Capelletti was born in Cesena, Italy, on the 4th of January, 1977. After scientific high-school, he studied Philosophy at the University of Bologna with a curriculum in history of philosophy and logic. He graduated in 2001 with a thesis on formal methods for natural language analysis.

In 2003, Matteo Capelletti went to the Netherlands, with a Marie Curie grant first and then with a Marco Polo grant, for research on type-logical grammars at UiL OTS, Utrecht University. In 2004, he was enrolled in the international PhD program at Utrecht University. During the PhD, Matteo Capelletti taught several courses on parsing and functional programming for CKI (Cognitive Artificial Intelligence) students at the Department of Philosophy at Utrecht University.