Proof Nets and Labeling for Categorial Grammar Logics

Richard Moot

20 August 1996

Contents

Chapter 1

Introduction

This paper is about proof theoretic and algorithmic aspects of categorial grammars.

In 1958 Lambek introduced his syntactic calculus to 'obtain an effective rule (or algorithm) for distinguishing sentences from nonsentences'. This direct link between grammar formalism and algorithm is one of the attractive points of categorial grammars, and Lambek's algorithm has been the basis for most early implementations of categorial grammar.

The last few years have seen a number of generalisations of the grammar logic of Lambek's original paper which have increased the linguistic coverage of these systems without sacrificing the basic logical nature of the system. These extensions have made the defects of the early implementations especially clear, and increased the need for a more uniform proof theory which allows for a transparent and efficient implementation.

The goal of this paper is to give such a proof theory, labeled proof nets, and see how efficiently implementations of this system will behave in comparison with other alternatives.

This paper is divided in two parts.

I. The goal of the first part is to develop a uniform proof theory for categorial grammar logics. This part is divided in two chapters

In chapter 2, I will give a short introduction to categorial grammar, and some of the recent developments in the field. For our initial presentation of categorial grammar we will use the sequent formulation, and discuss some of the problems associated with it.

In chapter 3, we will look for a better, more uniform proof system. A priori we want this proof theory to have the following properties

1. Soundness. Our proof system should derive *only* theorems. Without a soundness proof, we cannot in any reasonable way call our system a proof theory.
2. Completeness. Our proof system should derive *all* theorems. Some authors choose to sacrifice this imperative for efficiency and/or uniformity. In my opinion an incomplete proof system is only of limited interest.

3. Uniformity. We want our calculus to treat the syntactic and semantic aspects of natural language in a uniform way.
4. Decidability. We can determine in a finite amount of time whether or not a given list of words is a sentence. Some famous systems such as predicate logic or the full fragment of linear logic don't have this property. As our ultimate goal will be to implement the proof theory, we will restrict ourselves to logics which are known to be decidable.
5. Feasibility. We not only want to have a procedure to determine sentencehood in a finite amount of time, we want a procedure which does so in a *reasonable* amount of time. Though our knowledge of the complexity of the various categorial logics is partial, an NP-completeness result for one of the logics we consider makes it unlikely we will find an efficient algorithm.

Natural deduction will have an interpretation well-suited to natural language semantics, but problems associated with the heterogeneous nature of the rules make natural deduction not a first choice for automated deduction.

Finally we will turn to a hybrid logic of labeling and proof nets. The proof net calculus will combine both the semantic interpretation of natural deduction proofs and the rule symmetry of the sequent calculus in one system, and the labels will encode the structural aspects of grammatical wellformedness. Aside from the practical advantage this division of labour gives us, the structural and semantic aspects of natural language are generally assumed to be at different levels of linguistic description, so we are fully justified in making this move.

We will show that proof nets satisfy properties 1-4, and present a simple algorithm for automated proof net deduction. As for property 5, though the worst case complexity of the algorithm does not classify it as feasible, we will show that it is as good as it can be, given the goals we set.

II. The goal of the second part will then be to develop methods to make the (admittedly still naive) algorithm for our grammar logics work a bit better in order to give an algorithm which is also useful in practice.

We do this in two ways;

In chapter 4, we will improve the way we generate our proof nets in order to prevent the generation of unsound proof structures. Though we know the label algebra will 'filter out' these unsound structures, preventing their generation will give us a considerable computational improvement.

In chapter 5, we will use the label algebra to test if we can meet the structural restrictions of the language *during* (as opposed to after) the construction of the proof net. This will in many cases prevent the construction of proof structures we know will not be valid.

An appendix contains the Prolog source belonging to the text.

Chapter 2

Categorial Grammar

This chapter provides a necessarily brief introduction to categorial grammar and some related issues: labeled deduction and linear logic, which will prove to be important to our endeavor. We refer the reader to [Moortgat 96] or [Morrill 94] for a more detailed treatment of the logical framework of categorial grammar and its application to linguistics.

The central tenet of categorial grammar is that a grammar is a *logic*, and parsing a form of logical deduction. This leaves us very little room to manoeuvre or to postulate (non-logical) grammaticality principles. The aim of this chapter will be to show that in spite of this, we can have linguistic coverage comparable to any other modern grammar formalism with the additional advantage that we have built our formalism on firm, logical footing.

2.1 Lambek Categorial Grammar

2.1.1 AB grammars and Context Free Grammars

The earliest work on categorial grammar can be traced back to Ajdukiewicz and Bar-Hillel. The grammars from Ajdukiewicz and Bar-Hillel (mostly called **AB** grammars, for obvious reasons) are defined over a set of types defined as follows

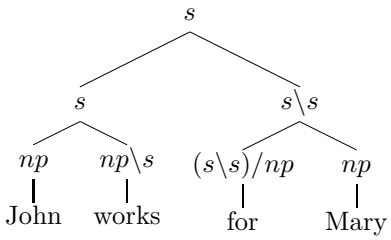
Definition 2.1 (AB Types) *Over a set of atomic types $\mathcal{A}$, the set of types $\mathcal{T}$ is defined inductively as*
$\mathcal{T} ::= \mathcal{A} \mid \mathcal{T}/\mathcal{T} \mid \mathcal{T}\backslash\mathcal{T}$

The set of atomic types is usually small. In this chapter I will work with only three atomic types: n (common noun), np (noun phrase) and s (sentence). **AB** grammars have only two rules, one for each of the type constructors, known in the categorial literature as (left/right) application. A and B denote arbitrary types, and ',' an implicit concatenation operation

$$A/B, B \rightarrow A \qquad\qquad B, B\backslash A \rightarrow A$$

Intuitively an expression of type A/B (resp. $B\backslash A$) combines with an expression B on its right hand side (resp. left hand side) to form an expression of type A.

With the appropriate lexical assignments this allows us to generate, for example, the following phrase structure tree

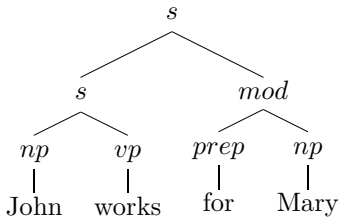


This is, and should be, familiar from the way context free grammars work, with the direction of the arrows systematically reversed. An important difference is that we only have two very general rules in our grammar, and that their use is dictated by the lexical type assignments. Categorial accounts of grammar are *lexicalist*, whereas generative accounts of grammar are rule-based. In a context free grammar all lexical items get assigned an atomic type, and we have a number of specialised rules.

AB grammars have the same recognising power as context free grammars, meaning that we can transform any **AB** grammar into a context free grammar which recognises the same sentences and vice versa. The small example from above is generated from the following context free grammar

$$\begin{array}{rlrl} s & \rightarrow np, vp & prep & \rightarrow \text{for} \\ s & \rightarrow s, mod & np & \rightarrow \text{John} \\ vp & \rightarrow \text{works} & np & \rightarrow \text{Mary} \\ mod & \rightarrow prep, np & & \end{array}$$

which produces a derivation tree isomorphic to the one above



2.1.2 The Lambek Calculus

With a number of examples from logic and linguistics, Lambek [Lambek 58] motivated a number of rules which should intuitively be valid in categorial grammars. Some of the proposed type-shifting principles were

$$\begin{array}{cc} A\backslash(B/C) \rightarrow (A\backslash B)/C & \text{(Associativity)} \\ A/B, B/C \rightarrow A/C & \text{(Composition)} \\ A \rightarrow B/(A\backslash B) & \text{(Lifting)} \end{array}$$

One of the important insights from Lambek was that the categorial slashes were in fact (directional versions of) implication, and the application rules were modus ponens. When we want to claim to have a grammar *logic*, modus ponens cannot exist without some reasoning about hypothetical resources. We cannot have a rule of *use* for a connective without a rule of *proof*, this rule symmetry is

at the heart of any real logic. The rules above are all attempts to capture this reasoning about hypothetical resources.

Lambek therefore gave a sequent formulation of his calculus, which is commonly referred to as **L**. We begin by extending the set of types with an explicit concatenation operator '•', which is the logical counterpart of ','. When we see the divisions '/' and '\' as directional implications, the product '•' is a directional version of conjunction.

Definition 2.2 (Types) *Over a set of atomic types $\mathcal{A}$, the set of types $\mathcal{T}$ is defined inductively as*
$\mathcal{T} ::= \mathcal{A} \mid \mathcal{T}/\mathcal{T} \mid \mathcal{T} \bullet \mathcal{T} \mid \mathcal{T}\backslash\mathcal{T}$

Definition 2.3 *A* **sequent** *is a pair $\langle \Gamma, C \rangle$ written as $\Gamma \Rightarrow C$, where Γ is a list of formulas called the antecedent and C a single formula called the succedent.*

The sequent formulation of the Lambek calculus **L** is as follows.

Identity

$$\frac{}{A \Rightarrow A}[Ax] \qquad \frac{\Gamma, B, \Gamma' \Rightarrow C \quad \Delta \Rightarrow B}{\Gamma, \Delta, \Gamma' \Rightarrow C}[Cut]$$

Logical Rules

$$\frac{\Gamma, A, B, \Delta \Rightarrow C}{\Gamma, A \bullet B, \Delta \Rightarrow C}[L\bullet] \qquad \frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A \bullet B}[R\bullet]$$

$$\frac{\Delta \Rightarrow B \quad \Gamma, A, \Gamma' \Rightarrow C}{\Gamma, A/B, \Delta, \Gamma' \Rightarrow C}[L/] \qquad \frac{\Gamma, B \Rightarrow A}{\Gamma \Rightarrow A/B}[R/]$$

$$\frac{\Delta \Rightarrow B \quad \Gamma, A, \Gamma' \Rightarrow C}{\Gamma, \Delta, B\backslash A, \Gamma' \Rightarrow C}[L\backslash] \qquad \frac{B, \Gamma \Rightarrow A}{\Gamma \Rightarrow B\backslash A}[R\backslash]$$

All rules are restricted to cases where the antecedents are nonempty. Linguistic reasoning must be about actual resources. Without this restriction we would be able to derive not only

$$\begin{array}{cccc} \text{'very} & \text{good} & \text{book'} & \\ (n/n)/(n/n) & n/n & n & \Rightarrow n \end{array}$$

but also, because we have an empty antecedent derivation $\Rightarrow n/n$

$$\begin{array}{ccc} \text{'}*\text{very} & \text{book'} & \\ (n/n)/(n/n) & n & \Rightarrow n \end{array}$$

The various type-shifting rules which have been proposed, now become derived (as opposed to stipulated) rules of inference. I will show below how lifting and composition can be derived from the sequent rules.

$$\dfrac{\dfrac{\dfrac{}{A \Rightarrow A}[Ax] \quad \dfrac{}{B \Rightarrow B}[Ax]}{B/A, A \Rightarrow B}[L/]}{A \Rightarrow (B/A)\backslash B}[R\backslash]$$

$$\dfrac{\dfrac{\dfrac{}{A \Rightarrow A}[Ax] \quad \dfrac{}{B \Rightarrow B}[Ax]}{A/B, B \Rightarrow A}[L/] \quad \dfrac{}{C \Rightarrow C}[Ax]}{\dfrac{A/B, B/C, C \Rightarrow A}{A/B, B/C \Rightarrow A/C}[R/]}[L/]$$

Lambek also proved *cut elimination* for the calculus. By giving an algorithm which could transform any sequent derivation with a number of cut rules into a derivation without use of the cut rule. In other words, removing the cut rule from the calculus does not result in loss of theorems. This has some important consequences.

- For all rules but $[Cut]$, the number of connectives of the premisses is one less than the number of connectives of the conclusion. For the system without cut, we therefore have the *subformula property*; in order to prove a formula we only have to use its subformulas.
- An immediate consequence of the subformula property for **L** is that we have a *procedure* for determining if a sentence is derivable. As applying a rule will decrease the total number of connectives in the sequent, and only a finite number of rules are applicable at any time. The Lambek calculus is decidable.
- We also know that sequents of the form $A \Rightarrow B$ are not provable for arbitrary A and B, as the only rule applicable to this sequent is cut. This means the Lambek calculus is consistent.

The cut free sequent formulation of **L** can be used directly as an algorithm, see for instance [Moortgat 88, Chapter 4], though it suffers from a number of efficiency problems. We will discuss this in more detail in section 2.2.5, where proposed extensions to the Lambek calculus make these efficiency problems even more acute.

2.1.3 Descriptive Limitations

It has long been assumed that the recognising capacity of **L** did not exceed that of context free grammars, though this was proved only recently by [Pentus 93]. There are a number of linguistic phenomena, such as verb raising in Dutch, which are argued to be outside of the domain of context free grammars (see for example [Shieber 85]). When using **L** to give a description of these phenomena we will undergenerate; we will fail to derive valid sentences.

The Lambek calculus also has problems of overgeneration. It derives some non-sentences. We can derive the sentence '∗man which Mary hates John and Tina likes', which is a so-called Island Constraint violation in **L** because the following is valid

'∗man	which	Mary	hates	John	and	Tina	likes'
n	$(n\backslash n)/(s/np)$	np	$(np\backslash s)/np$	np	$s\backslash(s/s)$	np	$(np\backslash s)/np \Rightarrow n$

In the next section we will see how to remedy these problems.

2.2 Extensions

To overcome the descriptive limitations of **L**, several extensions to the basic formalism have been proposed. See [Moortgat 96] for the 'state of the art' of contemporary categorial grammar. Because our ultimate goal will be to give an algorithm for such logics, we will restrict ourselves to those extensions which are known to be decidable.

2.2.1 The Categorial Landscape

In 1961 Lambek introduced the pure residuation logic **NL**. This was based on the principle of residuation, which specified only the relation of the logical constants with respect to eachother.

$$A \to C/B \quad \textit{iff} \quad A \bullet B \to C \quad \textit{iff} \quad B \to A\backslash C \qquad \text{(Res)}$$

and postulates of reflexivity and transitivity of derivability.

The sequent formulation of **NL** is as follows. Sequents for **NL** are pairs $\langle \mathcal{T}, \mathcal{F} \rangle$, where $\mathcal{T}$ is an antecedent term defined as $\mathcal{T} ::= \mathcal{F} \mid (\mathcal{T}, \mathcal{T})$. $\Gamma[\Delta]$ denotes an antecedent term Γ with a distinguished subterm occurrence Δ

Identity

$$\frac{}{A \Rightarrow A}[Ax] \qquad \frac{\Gamma[B] \Rightarrow C \quad \Delta \Rightarrow B}{\Gamma[\Delta] \Rightarrow C}[Cut]$$

Logical Rules

$$\frac{\Gamma[(A,B)] \Rightarrow C}{\Gamma[A \bullet B] \Rightarrow C}[L\bullet] \qquad \frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{(\Gamma, \Delta) \Rightarrow A \bullet B}[R\bullet]$$

$$\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(A/B, \Delta)] \Rightarrow C}[L/] \qquad \frac{(\Gamma, B) \Rightarrow A}{\Gamma \Rightarrow A/B}[R/]$$

$$\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(\Delta, B\backslash A)] \Rightarrow C}[L\backslash] \qquad \frac{(B, \Gamma) \Rightarrow A}{\Gamma \Rightarrow B\backslash A}[R\backslash]$$

In addition to the groups of logical and identity rules, there is a third group of rules called the structural rules. These rules dictate the resource management properties of the logic. For **NL** this rule component is empty.

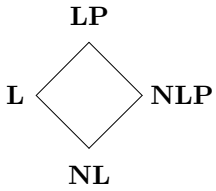
From the point of derivability **NL** is weaker than **L**, in the sense that it derives less theorems. There is for example no derivation of composition in **NL**

$$\frac{\dfrac{???}{((A/B, B/C), C) \Rightarrow A}}{(A/B, B/C) \Rightarrow A/C}[R/]$$

The brackets prevent the $[L/]$ rule from being applicable with the right partitioning. From the base logic **NL** we can get other logics by adding structural postulates of associativity and commutativity (the double line in the associativity postulate indicates the postulate is applicable in both directions)

$$\frac{\Gamma[(\Delta_2, \Delta_1)] \Rightarrow C}{\Gamma[(\Delta_1, \Delta_2)] \Rightarrow C}[Com] \quad \frac{\Gamma[(\Delta_1, (\Delta_2, \Delta_3))] \Rightarrow C}{\Gamma[((\Delta_1, \Delta_2), \Delta_3)] \Rightarrow C}[Ass]$$

This gives us a landscape of logics to work with, depending on which structural options we choose. If we add associativity, the logic we get will be **L** (the previous sequent formulation of **L** is a 'syntactic sugared' version of this logic, with the associativity inferences implicit in the structural punctuation ','). If we add commutativity, we get the nonassociative commutative Lambek calculus **NLP**. Finally, we get the Lambek-Van Benthem calculus **LP** by adding both associativity and commutativity, giving us the following picture



2.2.2 Multimodal Grammars

While the landscape above gives us a number of logics to choose from, none of these by itself is well suited to linguistic analysis. Natural language is multidimensional; different phenomena require different resource management properties.

A general method for combining several categorial logics into one system is given in [Moortgat & Oehrle 93]. We extend the formula language to multiple *families* of connectives, indexing both the logical constants and the structural punctuation. We will call the indices resource management modes or just modes.

Definition 2.4 (Multimodal Formulas) *Over a set of atomic formulas $\mathcal{A}$ and a set I of indices, we have the following set of formulas for all $i \in I$*
$\mathcal{F} ::= \mathcal{A} \mid \mathcal{F}/_i\mathcal{F} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F}\backslash_i\mathcal{F}$

Definition 2.5 (Antecedent Terms) *Over the set of formulas $\mathcal{F}$ and all elements i of the set of indices I, we define the set of antecedent terms $\mathcal{T}$ as follows*
$\mathcal{T} ::= \mathcal{F} \mid \mathcal{T} \circ_i \mathcal{T}$

A sequent will now be a pair $\langle \Gamma, C \rangle$ with Γ an antecedent term and C a multimodal formula. Like before, the notation $\Gamma[\Delta]$ will mean the antecedent term Γ has a distinguished subterm occurrence Δ.

We now redefine the sequent calculus in such a way that the rules respect the resource management modes. In all logical rules the logical connective is coindexed with the structural punctuation. For the rules $[L/_i]$,$[L\backslash_i]$ and $[R\bullet_i]$ this coindexing acts as a condition, allowing the connective to be eliminated only if it appears in the right context.

The rules in the identity group are mode-independent.

Identity

$$\frac{}{A \Rightarrow A}[Ax] \quad \frac{\Gamma[B] \Rightarrow C \quad \Delta \Rightarrow B}{\Gamma[\Delta] \Rightarrow C}[Cut]$$

Logical Rules

$$\frac{\Gamma[(A,B)^i] \Rightarrow C}{\Gamma[A \bullet_i B] \Rightarrow C}[L\bullet_i] \qquad \frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{(\Gamma,\Delta)^i \Rightarrow A \bullet_i B}[R\bullet_i]$$

$$\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(A/_iB,\Delta)^i] \Rightarrow C}[L/_i] \qquad \frac{(\Gamma,B)^i \Rightarrow A}{\Gamma \Rightarrow A/_iB}[R/_i]$$

$$\frac{\Delta \Rightarrow B \quad \Gamma[A] \Rightarrow C}{\Gamma[(\Delta,B\backslash_iA)^i] \Rightarrow C}[L\backslash_i] \qquad \frac{(B,\Gamma)^i \Rightarrow A}{\Gamma \Rightarrow B\backslash_iA}[R\backslash_i]$$

Structural Rules

$$\frac{\Gamma[\Delta_2] \Rightarrow C}{\Gamma[\Delta_1] \Rightarrow C}[SR]$$

We can now combine all logics from the structural landscape in one system, by giving each system its own mode and adding the appropriate mode specific structural rules.

System	Mode	Structural Rules
NL	n	–
L	a	Associativity
NLP	nc	Commutativity
LP	c	Associativity, Commutativity

For $i \in \{a,c\}$ and $j \in \{nc,c\}$ we add the following structural rules

$$\frac{\Gamma[\Delta_1 \circ_i \Delta_2 \circ_i \Delta_3] \Rightarrow C}{\Gamma[\Delta_1 \circ_i \Delta_2 \circ_i \Delta_3] \Rightarrow C}[Ass] \qquad \frac{\Gamma[\Delta_2 \circ_j \Delta_1] \Rightarrow C}{\Gamma[\Delta_1 \circ_j \Delta_2] \Rightarrow C}[Com]$$

We now have a multimodal logic, where all modes are hermetically sealed. When we add *communication* between modes to our logic, we get new possibilities which are unavailable in the individual logics.

Inclusion We can add a postulate like

$$\frac{\Gamma[\Delta_1 \circ_c \Delta_2] \Rightarrow C}{\Gamma[\Delta_1 \circ_a \Delta_2] \Rightarrow C}[Link]$$

indicating that $\Delta_1 \circ_a \Delta_2$ is more informative than $\Delta_1 \circ_c \Delta_2$, as in the former case we also know the order of the two antecedent terms with respect to each other. This would allow us to derive $A/_cB \Rightarrow A/_aB$, but not $A/_aB \Rightarrow A/_cB$.

Interaction We can add postulates which relate different modes to each other, examples of these 'mixed' postulates are

$$\frac{\Gamma[\Delta_1 \circ_w \Delta_3 \circ_a \Delta_2] \Rightarrow C}{\Gamma[\Delta_1 \circ_a \Delta_2 \circ_w \Delta_3] \Rightarrow C}[MxCom]$$

$$\frac{\Gamma[\Delta_1 \circ_a \Delta_2 \circ_w \Delta_3] \Rightarrow C}{\Gamma[\Delta_1 \circ_a \Delta_2 \circ_w \Delta_3] \Rightarrow C}[MxAss]$$

these postulates are used in for example [Moortgat & Oehrle 94] to give an account of Dutch verb raising, something impossible in unimodal **L**.

2.2.3 Unary Operators

By generalising the principle of residuation to unary connectives, we get a logic of pure residuation for families of unary connectives

$$\Diamond_j A \rightarrow B \quad \textit{iff} \quad A \rightarrow \Box_j^{\downarrow} B \qquad \text{(Res)}$$

It is helpful to see the connective '$\Diamond$' as the unary version of '$\bullet$', and the connective '$\Box^{\downarrow}$' as the unary version of '$/$' and '$\backslash$'. The unary connectives will also have their own structural counterpart $\langle . \rangle^j$.

We can see the sequent rules for the unary connectives as generalisations of the sequent rules for the binary connectives.

Definition 2.6 (Multimodal Formulas) *Over a set of atomic formulas $\mathcal{A}$, a set I of binary indices and a set J of unary indices, we have the following set of formulas*
$\mathcal{F} ::= \mathcal{A} \mid \Diamond_j \mathcal{F} \mid \Box_j^{\downarrow} \mathcal{F} \mid \mathcal{F} /_i \mathcal{F} \mid \mathcal{F} \bullet_i \mathcal{F} \mid \mathcal{F} \backslash_i \mathcal{F}$

Definition 2.7 (Antecedent Terms) *Over the set of formulas $\mathcal{F}$, we define the set of antecedent terms $\mathcal{T}$ as follows*
$\mathcal{T} ::= \mathcal{F} \mid \langle \mathcal{T} \rangle^i \mid \mathcal{T} \circ_i \mathcal{T}$

Unary Connectives

$$\frac{\Gamma[\langle A \rangle^i] \Rightarrow C}{\Gamma[\Diamond_i A] \Rightarrow C}[L\Diamond_i] \qquad \frac{\Gamma \Rightarrow C}{\langle \Gamma \rangle^i \Rightarrow \Diamond_i C}[R\Diamond_i]$$

$$\frac{\Gamma[A] \Rightarrow C}{\Gamma[\langle \Box_i A \rangle^i] \Rightarrow C}[L\Box_i^{\downarrow}] \qquad \frac{\langle \Gamma \rangle^i \Rightarrow C}{\Gamma \Rightarrow \Box_i C}[R\Box_i^{\downarrow}]$$

The unary connectives greatly extend the possible structural postulates. We can have postulates which determine the properties of the unary operators

$$\frac{\Gamma[\langle \Delta \rangle^i] \Rightarrow C}{\Gamma[\Delta] \Rightarrow C}[T] \qquad \frac{\Gamma[\langle \Delta \rangle^i] \Rightarrow C}{\Gamma[\langle \langle \Delta \rangle^i \rangle^i] \Rightarrow C}[4]$$

but we can also have interaction between the unary and binary modes of composition with distribution principles

$$\frac{\Gamma[\langle \Delta_1 \rangle^i \circ_j \langle \Delta_2 \rangle^i] \Rightarrow C}{\Gamma[\langle \Delta_1 \circ_j \Delta_2 \rangle^i] \Rightarrow C}[K]$$

$$\frac{\Gamma[\langle \Delta_1 \rangle^i \circ_j \Delta_2] \Rightarrow C}{\Gamma[\langle \Delta_1 \circ_j \Delta_2 \rangle^i] \Rightarrow C}[K1] \qquad \frac{\Gamma[\Delta_1 \circ_j \langle \Delta_2 \rangle^i] \Rightarrow C}{\Gamma[\langle \Delta_1 \circ_j \Delta_2 \rangle^i] \Rightarrow C}[K2]$$

or 'modally licensed' versions of the usual structural rules

$$\frac{\Gamma[\Delta_2 \circ_j \langle\Delta_1\rangle^i] \Rightarrow C}{\Gamma[\langle\Delta_1\rangle^i \circ_j \Delta_2] \Rightarrow C}[ModalCom]$$

Assigning the type $s\backslash(\Box^{\downarrow}s/s)$ to the coordinator 'and' is a way of preventing Island Constraint violations. After the coordinator has combined with its two s arguments, the unary constructor 'projects' an island around the resulting s, which will prevent future extraction.

The unary connectives play much the same role in categorial grammar as the exponentials do in linear logic (see section 2.4). They allow us to take an arbitrary logic in the categorial landscape and embed the other logics via a truth-preserving modal formula translation. See [Kurt. & Moortg. 95] for a detailed treatment.

These embeddings use unary constructors with only very weak properties (only residuation) to either license or restrict the use of structural rules. This raises the question if the postulates $[T]$ and [4], though familiar from modal logic, are necessary or even desirable in the linguistic setting. As these postulates correspond to relatively arbitrary adding and removing structure, we may want to prohibit their use.

2.2.4 Model Theory

Though this paper is primarily concerned with proof theory, there is alway a model theoretic side to a logic to which we will turn now. Restricting ourselves to a (still very large) subset of the possible structural postulates will also give us soundness and completeness with respect to the model.

Formulas will be interpreted in multimodal Kripke frames: tuples of the form $\langle W, \{R_j^2\}_{j\in J}, \{R_i^3\}_{i\in I}\rangle$. The set of worlds W are our linguistic resources and the accessibility relations R_j^2 and R_i^3 model the unary and binary composition of resources. The valuation v is defined as follows. It assigns arbitrary subsets of W to the atomic formulas, and valuates complex formulas as follows

$$\begin{array}{lcl} v(\Diamond_i A) & = & \{x \,|\, \exists y(R_i^2 xy \wedge y \in v(A))\} \\ v(\Box_i^{\downarrow} A) & = & \{y \,|\, \forall y(R_i^2 xy \Rightarrow x \in v(A))\} \\ v(A \bullet_i B) & = & \{x \,|\, \exists yz(R_i^3 xyz \wedge y \in v(A) \wedge z \in v(B))\} \\ v(A/_i B) & = & \{y \,|\, \forall xz((R_i^3 xyz \wedge z \in v(B)) \Rightarrow x \in v(A))\} \\ v(B\backslash_i A) & = & \{z \,|\, \forall xy((R_i^3 xyz \wedge y \in v(B)) \Rightarrow x \in v(A))\} \end{array}$$

Readers familiar with the Kripke semantics for modal logic will recognise the $\Diamond$ interpretation as that of the modal possibility operator, and the $\Box^{\downarrow}$ interpretation as that of the modal necessity operator if with the direction reversed (as suggested by the downarrow superscript). It is important to see that the binary connectives are also modal operators. They can be seen as generalisations of the unary cases.

The valuation above gives us a pure residuation logic for all modes. We translate structural postulates in *restrictions* on the accessibility relations R. A structural postulate $[T]$ for a unary mode j, for example, will restrict the possible accessibility relations R_j^2 to those which satisfy $\forall x(R_j^2 xx)$, i.e. are reflexive.

Definition 2.8 (Sahlqvist Postulates) *A weak Sahlqvist postulate is structural rule of the form*

$$\frac{\Gamma[\Delta_2] \Rightarrow C}{\Gamma[\Delta_1] \Rightarrow C}$$

subject to the following conditions

- *both* Δ_1 *and* Δ_2 *contain only the structural connectives* $. \circ_i .$ *and* $\langle . \rangle^i$ *and structural variables* Γ_i
- *there is no repetition of variables in* Δ_1
- *all variables in* Δ_2 *occur in* Δ_1

Proposition 2.1 ([Kurtonina 95]) *The multimodal sequent calculus is sound and complete for* $\mathbf{NL}\Diamond$ *and an arbitrary set of weak Sahlqvist postulates.*

2.2.5 Discussion: Sequents as a Decision Procedure

None of the proposed extensions affects the cut elimination theorem in any way. This means for the multimodal architecture we described, we can consider the system without the cut rule without loss of theorems, for which the subformula property, decidability (presuming of course the structural rule component is decidable) and consistency hold.

Decidability of the cut free sequent formulation provides us with an effective procedure for deciding whether or not a list of words is grammatical. An important question to ask is if this interpretation of the cut free sequent calculus as a procedure is also feasible. The decision procedure is as follows.

1. We look up the formula corresponding to a word in the lexicon. The nature of natural language makes this step essentially nondeterministic.
2. We try all possible unary and binary bracketings (for the unary bracketings we need to make an estimate as to the maximum number of unary brackets in the antecedent).
3. We try all possible sequences of structural rules. We use some form of memoing to avoid generating the same antecedent term more than once.
4. We try all possible logical rules.
5. We pass the resulting sequents to step 3 of the algorithm, knowing that the total number of connectives in the sequents is one less.

There are a number of problems with the algorithm defined above.

First, the complexity of step 2. For a system with only a single binary mode, there will already be $O(4^n)$ possible bracketings. This is unacceptably expensive for something that is just the initialisation phase of the algorithm.

Second, there is the problem of applying the structural rules at step 3. We can in the worst case convert an antecedent term with n formulas to $O(n!)$ different antecedent terms, and we can do this at each step.

Finally there is nondeterminism about which logical rule we apply at step 4. If we have n non-atomic formulas in the sequent, we have a choice of which of the n connectives we eliminate first. The problem with this is that it is sometimes (but not always) inessential which rule we select. The reason for this

is that, as we will see in section 3.1.2, the rules of the sequent calculus are not truly primitive but contain 'compiled' cut inferences. This is also the reason we can have no satisfactory treatment of natural language semantics in the sequent calculus.

All in all, using this algorithm in practice will be impossible for even very small sentences. $4^n \times n! \times n! \times (n-1)! \times \ldots \times 1!$ will result in over 10^9 steps in the computation for $n = 5$.

Work has been done to eliminate or ameliorate all of these problems. In section 4.1.4, I will discuss a uniform sequent proof system which does not suffer from these defects, though the high price we have to pay for this is incompleteness.

2.3 Labeled Deduction

Part of the problem with the sequent calculus is that we have no division between the logical and structural aspects of the calculus. Labeling will allow us to remedy this.

Gabbay [Gabbay 94] presents a general framework for designing hybrid logics, called labeled deduction. The basic and seemingly innocent move is to use not formulas A, but labeled formulas $\mathbf{x} : A$ as base declarative unit. The logical rules work not only on the formulas, but also on the associated label. The structural rules operate only on the labels.

Labeling will be an important topic in this paper, as the labels will provide us with a division between the logical and structural aspects of natural language. The logical rules will be those of **LP** with the labels acting as conditions or filters on derivability. This should be contrasted to the sequent rules above, which started from **NL** derivability and used the structural rules as licensing other more liberal forms of resource management.

A labeled deductive version of categorial grammar is presented below. This consists of replacing the antecedent by a multiset, and formulas by labeled formulas. Labels are the following

Definition 2.9 (Structure Labels) *Over a countably infinite set* $\mathbf{x}, \mathbf{y}, \mathbf{z}, \ldots$ *of structure variables* $\mathcal{V}$*, we define the set of structure labels inductively as follows*

$\mathcal{L} ::= \mathcal{V} \mid \langle \mathcal{L} \rangle^i \mid \mathcal{L} \circ_i \mathcal{L}$

All antecedent types are assigned a fresh structure variable, and the succedent formula a metavariable Z which will get fully instantiated during the proof. The succedent label will represent the way the antecedent resources are configured. The notation $Z[X]$ will now be interpreted as a label Z with a distinguished occurrence of a sublabel X. Newly introduced structure variables are assumed to be fresh.

Identity

$$\frac{}{\mathbf{x} : A \Rightarrow \mathbf{x} : A}[Ax] \qquad \frac{\Gamma, \mathbf{y} : B \Rightarrow Z[\mathbf{y}] : C \quad \Delta \Rightarrow Y : B}{\Gamma, \Delta \Rightarrow Z[Y] : C}[Cut]$$

Binary Connectives

$$\frac{\Gamma, \mathbf{x} : A, \mathbf{y} : B \Rightarrow Z[\mathbf{x} \circ_i \mathbf{y}] : C}{\Gamma, \mathbf{z} : A \bullet_i B \Rightarrow Z[\mathbf{z}] : C}[L\bullet_i] \qquad \frac{\Gamma \Rightarrow X : A \quad \Delta \Rightarrow Y : B}{\Gamma, \Delta \Rightarrow X \circ_i Y : A \bullet_i B}[R\bullet_i]$$

$$\frac{\Delta \Rightarrow Y : B \quad \Gamma, \mathbf{x} : A \Rightarrow Z[\mathbf{x}] : C}{\Gamma, \Delta, \mathbf{y} : A/_i B \Rightarrow Z[\mathbf{y} \circ_i Y] : C}[L/_i] \qquad \frac{\Gamma, \mathbf{y} : B \Rightarrow X \circ_i \mathbf{y} : A}{\Gamma \Rightarrow X : A/_i B}[R/_i]$$

$$\frac{\Delta \Rightarrow Y : B \quad \Gamma, \mathbf{x} : A \Rightarrow Z[\mathbf{x}] : C}{\Gamma, \Delta, \mathbf{y} : B\backslash_i A \Rightarrow Z[Y \circ_i \mathbf{y}] : C}[L\backslash_i] \qquad \frac{\Gamma, \mathbf{y} : B \Rightarrow \mathbf{y} \circ_i X : A}{\Gamma \Rightarrow X : B\backslash_i A}[R\backslash_i]$$

Unary Connectives

$$\frac{\Gamma, \mathbf{x} : A \Rightarrow Z[\langle \mathbf{x} \rangle^i] : C}{\Gamma, \mathbf{y} : \Diamond_i A \Rightarrow Z[\mathbf{y}] : C}[L\Diamond_i] \qquad \frac{\Gamma \Rightarrow Z : C}{\Gamma \Rightarrow \langle Z \rangle^i : \Diamond_i C}[R\Diamond_i]$$

$$\frac{\Gamma, \mathbf{x} : A \Rightarrow Z[\mathbf{x}] : C}{\Gamma, \mathbf{y} : \Box_i A \Rightarrow Z[\langle \mathbf{y} \rangle^i] : C}[L\Box_i^{\downarrow}] \qquad \frac{\Gamma \Rightarrow \langle Z \rangle^i : C}{\Gamma \Rightarrow Z : \Box_i C}[R\Box_i^{\downarrow}]$$

Structural Rules

$$\frac{\Gamma \Rightarrow Z[X] : C}{\Gamma \Rightarrow Z[Y] : C}[SR]$$

Definition 2.10 *We can define a (reversible) function from succedent labels to antecedent configurations in the following way. Note that the constructors* $\langle . \rangle^i$ *and* $. \circ_i .$ *are overloaded, on the left hand side of the equations they are label constructors, while on the right hand side they are antecedent constructors.*

$\|\mathbf{x}\| = A$ iff $\mathbf{x} : A$ is a formula in the antecedent
$\|\langle X \rangle^i\| = \langle \|X\| \rangle^i$
$\|X \circ_i Y\| = \|X\| \circ_i \|Y\|$

With this translation in hand, we can easily show that the labeled and non-labeled sequent calculus derive the same theorems

Proposition 2.2 $\Gamma \Rightarrow X : C \Longleftrightarrow \Gamma' \Rightarrow C$, *where* $\|X\| = \Gamma'$.

Proof Induction. □

An immediate (algorithmic) advantage the labeled deductive formulation of the calculus has is that it *generates* the structure of the antecedent during the derivation, while for the non-labeled formulation it is not immediately obvious how to do this. Simply summing up all possible bracketings would produce a factor of $O(4^n)$ before we have even started the derivation.

The labels also prevent the derivation of sequents with empty antecedent. For the labeled systems we consider we will shift the burden of detecting empty antecedents to the label algebra. This will prove useful when we consider proof nets in section 3.2 which will have no antecedents in any real sense. There are also some applications of systems with some mode dependent identity element, so we give the grammar designer a choice of whether or not to use them.

A structural rule

$$\frac{\Gamma \Rightarrow Z[\epsilon \circ_i X] : C}{\Gamma \Rightarrow Z[X] : C}[Id]$$

gives us a left identity element for mode i, which allows us to derive the following

$$\frac{\dfrac{\dfrac{}{\mathbf{x} : A \Rightarrow \mathbf{x} : A}[Ax]}{\mathbf{x} : A \Rightarrow \epsilon \circ_i \mathbf{x} : A}[Id]}{\Rightarrow \epsilon : A/_i A}[L/]$$

2.4 Categorial Grammar and Linear Logic

Categorial grammar can be situated in the larger landscape of 'substructural logics'; those logics lacking or restricting the use of the structural rules which apply freely in classical and intuitionistic logic. Another active area of research in this landscape is linear logic. As we will be using some results from linear logic in this paper, I will (very) briefly introduce linear logic and show the intuitionistic, multiplicative fragment of linear logic is essentially **LP**.

Linear logic [Girard 87] is obtained from the sequent formulation of classical logic by dropping the structural rules of contraction and weakening.

$$\frac{\Gamma, A, A \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta}[LC] \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma, A \Rightarrow \Delta}[LW]$$

Contraction states that the number of occurrences of a formulas doesn't matter, whereas weakening states that we can add arbitrary formulas to the antecedent without affecting derivability.

These rules are appropriate when we use logic to model 'eternal, stable truths', but when we want to model *actions* (or state transitions, processes etc.) we want to model that when we enter a new state we 'destroy' or 'update' the old state. This state transition behavior will be the meaning of linear implication '$\multimap$'.

In linear logic we have three kinds of connectives

Multiplicatives '$\multimap$','$\otimes$' and '$\parr$'. Linear implication in classical linear logic is a defined connective. Its definition is $A \multimap B \equiv A^{\perp} \parr B$, with $\perp$ being linear negation. The multiplicative conjunction '$\otimes$' intuitively means we have *both* an A and a B resource.

Additives '$\oplus$' and '&'. A formula $A \oplus B$ or $A \& B$ means we have *either* an A or a B resource. In the case of additive disjunction $A \oplus B$ we don't know which. In the case of additive conjunction $A \& B$ we can choose which.

Exponentials '!' and '?'. The exponentials allow us to represent repeatability. A formula $!A$ means that we have an arbitrary quantity of A formulas. These connectives form the basis for embedding classical and intuitionistic logic into linear logic, but also for an undecidability proof.

In this paper we will only use the multiplicative connectives of linear logic. The additives and exponentials, though they may have some linguistic applications, will complicate our proof theory, and may result in an undecidable logic.

The sequent formulation for the multiplicative fragment of classical linear logic is as follows

Definition 2.11 *A classical linear sequent is a pair* $\langle \Gamma, \Delta \rangle$ *written* $\Gamma \Rightarrow \Delta$ *with* Γ *and* Δ *both multisets of formulas. The implicit meaning of these sequents will be that the conjunction of the formulas in* Γ *implies the disjunction of the formulas in* Δ*.*

$$\frac{}{A \Rightarrow A}[Ax]$$

$$\frac{\Gamma \Rightarrow A, \Delta}{\Gamma, A^{\perp} \Rightarrow \Delta}[L\perp] \quad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow A^{\perp}, \Delta}[R\perp]$$

$$\frac{\Gamma_1, A \Rightarrow \Delta_1 \quad \Gamma_2, B \Rightarrow \Delta_2}{\Gamma_1, \Gamma_2, A \parr B \Rightarrow \Delta_1, \Delta_2}[L\parr] \quad \frac{\Gamma \Rightarrow A, B, \Delta}{\Gamma \Rightarrow A \parr B, \Delta}[R\parr]$$

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \otimes B \Rightarrow \Delta}[L\otimes] \quad \frac{\Gamma_1 \Rightarrow A, \Delta_1 \quad \Gamma_2 \Rightarrow B, \Delta_2}{\Gamma_1, \Gamma_2 \Rightarrow A \otimes B, \Delta_1, \Delta_2}[R\otimes]$$

$$\frac{\Gamma_1, B \Rightarrow \Delta_1 \quad \Gamma_2 \Rightarrow A, \Delta_2}{\Gamma_1, \Gamma_2, A \multimap B \Rightarrow \Delta_1, \Delta_2}[L\multimap] \quad \frac{\Gamma, A \Rightarrow B, \Delta}{\Gamma \Rightarrow A \multimap B, \Delta}[R\multimap]$$

Example 2.1 *The following example is from [Girard e.a. 95]. When modeling chemical processes, we may have an axiom stating* $H_2 \otimes H_2 \otimes O_2 \multimap H_2O \otimes H_2O$*. This will allow us to derive that we can use two molecules of* H_2 *and one molecule of* O_2 *to produce two molecules of* H_2O*, after which we won't have any molecules of* H_2 *or* O_2 *left.*

Therefore we cannot (and wouldn't want to) derive the following in linear logic, though the classical versions of these formulas are derivable.

$H_2O \multimap H_20 \otimes H_2O$

$H_20 \otimes CH_4 \multimap H_2O$

The intuitionistic fragment of linear logic is obtained by restricting the right hand side of the sequent to a single formula occurrence. This means dropping the connective '$\parr$' as its right rule essentially involves two succedent formulas.

When we also drop negation and interpret '$\multimap$' as either '/' or '\' and '$\otimes$' as '$\bullet$' the rules for this fragment are exactly those of **LP**.

When the direction of the slashes is unimportant it is often more convenient to use the linear logic notation for **LP** formulas.

Chapter 3

Proof Systems

In this chapter, we will examine two alternatives to the sequent calculus from the previous chapter. In section 3.1 we will look at natural deduction and the Curry-Howard interpretation of natural deduction proofs with its application to natural language semantics, and in section 3.2 we will look at proof nets which combine many of the 'good' aspects of sequent calculus and natural deduction. Proof nets will provide us with a general, uniform framework for capturing both the form and meaning aspects of natural language and will be our proof system of choice for the remainder of this paper.

3.1 Natural Deduction

Natural deduction proofs are tree-like structures where the conclusion is the root of the tree and the hypotheses are the leaves. Instead of sequent left (resp. right) rules we will have elimination (resp. introduction) rules for our connectives.

Leaves can be either 'active' or 'inactive', some of the rules 'discharging' a single active hypothesis occurrence, for example

$$\frac{\begin{array}{c}[B]^n \\ \vdots \\ A\end{array}}{A/_i B}[/I]^n$$

where the bracketing indicates the hypothesis B is no longer active beyond the rule with which it is coindexed.

If there is a natural deduction proof with undischarged leaves Γ for A, we will write this as $\Gamma \vdash A$ or as a tree like

$$\begin{array}{c}\Gamma \\ \vdots \\ A\end{array}$$

In line with the labeled deduction perspective we are following, the basic units of our natural deduction system will be labeled formulas $X : A$ where X is a structure label.

The full natural deduction calculus looks as follows (as usual structure variables $\mathbf{x}, \mathbf{y}, \ldots$ are chosen fresh).

Hypothesis

$$\mathbf{x} : A$$

Binary Connectives

$$\frac{X : A \bullet_i B \qquad Z[\mathbf{x} \circ_i \mathbf{y}] : C}{Z[X] : C}[\bullet E]^n \qquad \frac{\begin{array}{c}[\mathbf{x} : A]^n \quad [\mathbf{y} : B]^n \\ \vdots \\ X : A \quad Y : B\end{array}}{X \circ_i Y : A \bullet_i B}[\bullet I]$$

$$\frac{X : A/_i B \quad Y : B}{X \circ_i Y : A}[/E] \qquad \frac{\begin{array}{c}[\mathbf{x} : B]^n \\ \vdots \\ X \circ_i \mathbf{x} : A\end{array}}{X : A/_i B}[/I]^n$$

$$\frac{Y : B \quad X : B \backslash_i A}{Y \circ_i X : A}[\backslash E] \qquad \frac{\begin{array}{c}[\mathbf{x} : B]^n \\ \vdots \\ \mathbf{x} \circ_i X : A\end{array}}{X : B \backslash_i A}[\backslash I]^n$$

Unary Connectives

$$\frac{X : \Diamond_i A \quad Z[\langle \mathbf{x} \rangle^i] : C}{Z[X] : C}[\Diamond E]^n \qquad \frac{\begin{array}{c}[\mathbf{x} : A]^n \\ \vdots \\ X : A\end{array}}{\langle X \rangle^i : \Diamond_i A}[\Diamond I]$$

$$\frac{X : \Box_i^{\downarrow} A}{\langle X \rangle^i : X}[\Box^{\downarrow} E] \qquad \frac{\langle X \rangle^i : A}{X : \Box_i^{\downarrow} A}[\Box^{\downarrow} I]$$

Structural Rules

$$\frac{Z[X] : A}{Z[Y] : A}[SR]$$

Example 3.1 *Natural deduction proof of* $\Diamond_i \Box_i^{\downarrow} \Diamond_i \Box_i^{\downarrow} A \vdash \Diamond_i \Box_i^{\downarrow} A$.

$$\frac{\mathbf{x} : \Diamond_i \Box_i^{\downarrow} \Diamond_i \Box_i^{\downarrow} A \qquad \dfrac{\dfrac{[\mathbf{y} : \Box_i^{\downarrow} \Diamond_i \Box_i^{\downarrow} A]^1}{\langle \mathbf{y} \rangle^i : \Diamond_i \Box_i^{\downarrow} A}[\Box^{\downarrow} E]}{\mathbf{x} : \Diamond_i \Box_i^{\downarrow} A}[\Diamond E]^1$$

Example 3.2 *Natural deduction proof of* $np, (np \backslash_a s)/_a np, (s/_a np) \backslash_a s \vdash s$ *'Ted hates everyone' for an associative mode a.*

$$\dfrac{\dfrac{\dfrac{\mathbf{ted} : np \qquad \dfrac{\mathbf{hates} : (np \backslash_a s)/_a np \quad [\mathbf{x} : np]^1}{\mathbf{hates} \circ_a \mathbf{x} : np \backslash_a s}[/E]}{\mathbf{ted} \circ_a \mathbf{hates} \circ_a \mathbf{x} : s}[\backslash E]}{\mathbf{ted} \circ_a \mathbf{hates} \circ_a \mathbf{x} : s}[Ass]}{\mathbf{ted} \circ_a \mathbf{hates} : s/_a np}[/I]^1 \qquad \mathbf{everyone} : (s/_a np) \backslash_a s}{\mathbf{ted} \circ_a \mathbf{hates} \circ_a \mathbf{everyone} : s}[\backslash E]$$

If we allow hypotheses with arbitrary structure labels, we preserve validity as indicated by the following lemma

Lemma 3.1 (Substitution Lemma) *If* $\Gamma, \mathbf{x} : A \vdash Z[\mathbf{x}] : C$ *then* $\Gamma, X : A \vdash Z[X] : C$

Proof Induction. □

3.1.1 Sequent Calculus and Natural Deduction

The aim of this section is to show that the labeled sequent calculus of section 2.3 and the natural deduction system of section 3.1 derive the same theorems. We will translate sequent right rules to natural deduction introduction rules and sequent left rules to natural deduction elimination rules.

Theorem 3.2 $\Gamma \Rightarrow X : A \Longleftrightarrow \Gamma \vdash X : A$

Proof This proof is pretty basic, so we will only prove the unary cases. The binary cases are a straightforward extension of this proof

($\Rightarrow$) Induction on the length of the sequent proof

$n = 1$ The sequent proof is an axiom $\mathbf{x} : A \Rightarrow \mathbf{x} : A$. It corresponds to the natural deduction $\mathbf{x} : A$.

$n > 1$ We look at the last rule of the proof.

[*Cut*] By induction hypothesis we have a natural deduction proof $\Delta \vdash Y : B$ and $\Gamma, \mathbf{y} : B \vdash Z[\mathbf{y}] : C$. Application of lemma 3.1 gives us a proof of $\Gamma, Y : B \vdash Z[Y] : C$. We can compose these proofs in the following way

$$\frac{\Gamma, \mathbf{y} : B \Rightarrow Z[\mathbf{y}] : C \quad \Delta \Rightarrow Y : B}{\Gamma, \Delta \Rightarrow Z[Y] : C}[Cut] \quad \rightsquigarrow \quad \begin{array}{c} \Delta \\ \vdots \\ \Gamma \quad Y : B \\ \vdots \\ Z[Y] : C \end{array}$$

[$L\Diamond$] By induction hypothesis we already have a natural deduction proof of $\Gamma, A \Rightarrow C$ so we translate this rule as follows

$$\frac{\Gamma, \mathbf{x} : A \Rightarrow Z[\langle \mathbf{x} \rangle^i] : C}{\Gamma, \mathbf{y} : \Diamond_i A \Rightarrow Z[\mathbf{y}] : C}[L\Diamond_i] \rightsquigarrow \frac{\mathbf{y} : \Diamond_i A \quad \begin{array}{c} \Gamma \quad [\mathbf{x} : A]^k \\ \vdots \\ Z[\langle \mathbf{x} \rangle^i] : C \end{array}}{Z[\mathbf{y}] : C}[\Diamond E]^k$$

[$L\Box^{\downarrow}$] We translate this rule with application of the substitution lemma 3.1 and induction hypothesis as follows

$$\frac{\Gamma, \mathbf{x} : A \Rightarrow Z[\mathbf{x}] : C}{\Gamma, \mathbf{y} : \Box_i A \Rightarrow Z[\langle \mathbf{y} \rangle^i] : C}[L\Box_i^{\downarrow}]^k \quad \rightsquigarrow \quad \begin{array}{c} \Gamma \quad \dfrac{\mathbf{y} : \Box^{\downarrow} A}{\langle \mathbf{y} \rangle^i : A}[\Box^{\downarrow} E] \\ \vdots \\ Z[\langle \mathbf{y} \rangle^i] : C \end{array}$$

[$R\Box^{\downarrow}$],[$R\Diamond$],[SR] Trivial.

($\Leftarrow$) Induction on the length of the natural deduction proof

$n = 1$ Trivial.

$n > 1$ We look at the last rule of the proof

$[\Diamond E]$ We have the following translation

$$\dfrac{\begin{array}{cc}\Delta & \Gamma \quad [\mathbf{x} : A]^k \\ \vdots & \vdots \\ X : \Diamond_i A & Z[\langle \mathbf{x} \rangle^i] : C\end{array}}{Z[X] : C}[\Diamond E]^k \quad \rightsquigarrow$$

$$\dfrac{\dfrac{\Gamma, \mathbf{x} : A \Rightarrow Z[\langle \mathbf{x} \rangle^i] : C}{\Gamma, \mathbf{y} : \Diamond_i A \Rightarrow Z[\mathbf{y}] : C}[L\Diamond] \quad \Delta \Rightarrow X : \Diamond_i A}{\Gamma, \Delta \Rightarrow Z[X] : C}[Cut]$$

$[\Box^{\downarrow} E]$ We have the following translation

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots \\ X : \Box^{\downarrow} A\end{array}}{\langle X \rangle^i : A}[\Box^{\downarrow} E] \quad \rightsquigarrow$$

$$\dfrac{\dfrac{\dfrac{}{\mathbf{x} : A \Rightarrow \mathbf{x} : A}[Ax]}{\mathbf{y} : \Box_i^{\downarrow} A \Rightarrow \langle \mathbf{y} \rangle^i : A}[L\Box^{\downarrow}] \quad \Gamma \Rightarrow X : \Box_i^{\downarrow} A}{\Gamma \Rightarrow \langle X \rangle^i : A}[Cut]$$

$[\Diamond I], [\Box^{\downarrow} I], [SR]$ Trivial.

□

3.1.2 The Curry-Howard Isomorphism

The main interest in natural deduction lies in the Curry-Howard isomorphism; by interpreting functional types as implicational formulas and pairing types as conjunctive formulas, we can see constructing a term t of a type A in the simply typed lambda calculus corresponds to finding a natural deduction proof Π of the formula corresponding to A.

We will make this a bit more precise below

Definition 3.1 (Types) *The types are defined inductively as follows*

1. *we have a finite number $T_1, \ldots, T_n$ of atomic types.*
2. *if T and U are types, $T \to U$ is a type (function space).*
3. *if T and U are types, $T \times U$ is a type (Cartesian product).*

Atomic formulas correspond to atomic types, the implication '$\multimap$' will correspond to '$\to$', and the product '$\otimes$' will correspond to '$\times$'.

We can now define the set of linear lambda terms as follows

Definition 3.2 (Linear Lambda Terms) *The linear lambda terms are defined inductively as follows*

1. *for each type T we have a (countably infinite) supply $x, y, z, \ldots$ of variables of that type.*

2. *if t is a term of type $T \to U$ and u is a term of type T then (tu) is a term of type U.*

3. *if t is a term of type U and x is a term of type T which* occurs exactly once *in t, then $\lambda x.t$ is a term of type $T \to U$.*

4. *if u is a term of type $U \times V$ and t is a term of type T in which a term x of type U and a term y of type V occur exactly once then t with $\pi^1 u$ substituted for x and $\pi^2 u$ substituted for y is also a term of type T .*

5. *if t is a term of type T and u is a term of type U then $\langle u, v\rangle$ is a term of type $T \times U$.*

An alternative way to define the set of linear lambda terms is given in [Abramsky 93].

Just as the set of **LP** proofs is a proper subset of of the set of intuitionistic proofs, the set of linear lambda terms is a proper subset of the set of lambda terms. We get the usual definition of lambda terms by dropping the 'occurs exactly once' restriction of case 3 and case 4. Case 4 without this restriction is equivalent to the more familiar

4. *if u is a term of type $U \times V$ then $\pi^1 u$ is a term of type U and $\pi^2 u$ is a term of type V.*

The isomorphism is between $\mathbf{LP}\Diamond$ proofs and linear lambda terms. As we can see the structure labels as restrictions on our base $\mathbf{LP}\Diamond$ logic we lose derivations when we add the structure labels. This means some lambda terms will have no derivation associated to them. We can have two views on this.

From a *logical* point of view we may want to recover the isomorphism for logics with stricter resource management than $\mathbf{LP}\Diamond$. This is done in for example [Abrusci 96].

From a *linguistic* point of view, which is the view we will adopt in this paper, we may want to keep Curry-Howard as a *correspondence*. We can see $\mathbf{LP}\Diamond$ as our semantic composition language, and the structure labels as syntactic constraints on derivability. This way $\mathbf{LP}\Diamond$ is a bridge between the syntactic and semantic aspects of natural language.

We will make the isomorphism fully explicit by presenting a 'semantically' labeled version of the natural deduction calculus from the previous section, where the term operations mirror the rules. The unary connectives will have their own term constructors.

x, y, z denote fresh variables and t, u, v denote arbitrary lambda term labels.

Hypothesis

$x : A$

Binary Connectives

$$\frac{\begin{array}{cc}[x:A]^n & [y:B]^n \\ & \vdots \\ u:A\otimes B & t:C\end{array}}{t[x:=\pi^1 u, y:=\pi^2 u]:C}[\otimes E]^n \qquad \frac{t:A \quad u:B}{\langle t,u\rangle : A\otimes B}[\otimes I]$$

$$\frac{t:A\multimap B \quad u:A}{(tu):B}[\multimap E] \qquad \frac{\begin{array}{c}[x:A]^n \\ \vdots \\ t:B\end{array}}{\lambda x.t : A\multimap B}[\multimap I]^n$$

Unary Connectives

$$\frac{\begin{array}{cc}[y:A]^n & \\ \vdots & \\ u:\Diamond_i A & t:C\end{array}}{t[y:=^{\cup} u]:C}[\Diamond E]^n \qquad \frac{x:A}{{}^{\cap}x:\Diamond_i A}[\Diamond I]$$

$$\frac{t:\Box_i^{\downarrow} A}{{}^{\vee}t:A}[\Box^{\downarrow} E] \qquad \frac{t:A}{{}^{\wedge}t:\Box_i^{\downarrow} A}[\Box^{\downarrow} I]$$

For the sequent calculus we don't have such a 1-1 correspondence between proofs and terms, for example the following two sequent proofs

$$\frac{\dfrac{B\Rightarrow B \quad C\Rightarrow C}{B, B\multimap C\Rightarrow C}[L\multimap] \quad A\Rightarrow A}{A, A\multimap B, B\multimap C\Rightarrow C}[L\multimap] \qquad \frac{\dfrac{A\Rightarrow A \quad B\Rightarrow B}{A, A\multimap B\Rightarrow B}[L\multimap] \quad C\Rightarrow C}{A, A\multimap B, B\multimap C\Rightarrow C}[L\multimap]$$

translate to the same natural deduction proof

$$\frac{\dfrac{A \quad A\multimap B}{B}[\multimap E] \quad B\multimap C}{C}[\multimap E]$$

The rules of the sequent calculus are not truly primitive but combinations of the 'true' rules with cut inferences (the translation from natural deduction proofs to sequent proofs makes this especially clear). As a consequence it allows for rule permutations which are inessential to the proof.

3.1.3 Normalisation

For the linear lambda terms we have the following set of equivalences, a beta and eta equivalence for each type. The equivalences for the implicational types have the restriction of t being free for u resp. x. We can always meet this restriction by renaming variables when conflicts occur.

$$\begin{array}{cc} (\lambda x.t)u =_\beta t[x := u] & \lambda x.(tx) =_\eta t \\ \pi^i\langle t_1, t_2\rangle =_\beta t_i & \langle \pi^1 t, \pi^2 t\rangle =_\eta t \\ {}^\cup({}^\cap t) =_\beta t & {}^\cap({}^\cup t) =_\eta t \\ {}^\vee({}^\wedge t) =_\beta t & {}^\wedge({}^\vee t) =_\eta t \end{array}$$

Though equivalences, we will usually apply them from left to right as asymmetric *conversions*, in which case we will call the left hand side of the equivalence a *redex* and the right hand side its *contractum*.

Definition 3.3 (Reduction) *We will say a term t* **converts** *to a term u if it can be obtained by replacing a subterm which is a* redex *by its* contractum.

We will say a term t **reduces** *to a term u ($t \rightsquigarrow u$) if it can be obtained by zero or more conversion steps from t.*

Definition 3.4 (Normal form) *A term is* **beta normal** *or just* **normal** *iff it does not contain any beta redexes.*

If a term t reduces to a term u and u is normal, we will call u a **normal form** *of t.*

Proposition 3.3 *For every term t there exists a normal form u such that $t \rightsquigarrow u$, moreover if $t \rightsquigarrow u'$ and u' is normal then we have $u \equiv u'$.*

We will not prove this proposition here, but refer the interested reader to [Girard e.a. 89].

By the Curry-Howard isomorphism, the conversions on terms correspond to conversions on proofs, for example beta conversion for the implicational types corresponds to replacing the proof

$$\dfrac{\dfrac{\begin{array}{c}[x:B]^k \\ \vdots\,\Pi \\ t:A\end{array}}{\lambda x.t : B \multimap A}[\multimap I]^k \qquad \begin{array}{c}\vdots\,\Sigma \\ u:B\end{array}}{(\lambda x.t)u : A}[\multimap E]$$

by a (simpler) proof, where instead of hypothesising a proof of B, we use proof Σ of B directly

$$\begin{array}{c}\vdots\,\Sigma \\ u:B \\ \vdots\,\Pi \\ t[x := u] : A\end{array}$$

We will call a natural deduction proof *normal* iff its corresponding lambda term is normal. If we translate a cut-free sequent proof by the translation given in the previous section, the natural deduction we get will be normal. Similarly, if we translate a sequent proof where the axiom rule has been restricted to atomic formulas, which is possible for all logics we consider, the resulting natural deduction proof will be eta expanded (i.e. all eta conversions have been applied from right to left, where possible).

3.1.4 Linguistic Importance of the Isomorphism

As we said before, the lambda term language will provide us with a direct link to natural language semantics. In the linguistic setting the proofs as terms interpretation becomes a *proofs as readings* interpretation, in the sense that a different proof of a sentence will give a different reading of that sentence. The semantic labeling on the natural deduction proofs will give us the meaning of a whole sentence as a function composed out of its parts. For the sequent calculus, as it has no Curry-Howard interpretation, a different proof may give us the same reading which makes it unsuitable for our intended theory of semantics.

Giving lexical items a meaning recipe in the language of first order predicate logic enriched with the lambda abstractor and other term constructors, allows us to give our lexical entries Montague-style meaning recipes.

The lambda term language however is sufficiently expressive to accommodate a variety of semantic theories. See for example [Muskens 95] for an integration between categorial grammar and discourse representation theory.

Example 3.3 *Natural deduction proof of* $np, (np\backslash_a s)/_a np, (s/_a np)\backslash_a s \vdash s$ *'Ted hates everyone', this time with semantic labeling.*

Lexicon
$ted : np$
$hates : (np\backslash_a s)/_a np$
$\lambda f.\forall y.(fy) : (s/_a np)\backslash_a s$

$$\dfrac{ted : np \qquad \dfrac{\dfrac{hates : (np\backslash_a s)/_a np \quad [x : np]^1}{(hates\ x) : np\backslash_a s}[/E] \qquad \lambda f.\forall y.(fy) : (s/_a np)\backslash_a s}{((hates\ x)\ ted) : s}[\backslash E]}{\lambda x.((hates\ x)\ ted) : s/_a np}[/I]^1}{(\lambda f.\forall y.(fy))\lambda x.((hates\ x)\ ted)) : s}[\backslash E]$$

Which reduces in two beta steps to $\forall y.((hates\ y)\ ted)$.

3.1.5 Discussion: Problems With Natural Deduction

The normalisation theorem for natural deduction proofs allows us to prove that the subformula property and decidability also hold for natural deduction proofs (see [Prawitz 71]). This means that when we restrict ourselves to normal proofs, we have a decision procedure for the natural deduction calculus.

The natural deduction calculus as presented above suffers from a number of problems, to which there are no easy solutions.

One problem is that because our natural deductions are trees, we can have only a single conclusion. This makes the natural deduction rules asymmetrical, especially when compared to the sequent rules.

Another problem is that the rules $[\bullet_i E]$ and $[\Diamond_i E]$ make use of a 'context' formula C. This is a concession to the sequent calculus, and the same problem as with the intuitionistic rule $[\vee E]$. These rules will require cumbersome permutation conversions in order to make the normalisation proof work.

We can conclude that the natural deduction calculus is not very uniform, only for different reasons than the sequent calculus. In natural deduction the nonuniformity is in the rules, while in the sequent calculus the nonuniformity is in the inability to give a good account of natural language semantics.

Proof nets, in the next section, will provide a solution to all these problems.

3.2 Proof Nets

Girard [Girard 87] describes a proof system for linear logic which, like sequent calculus, is decidable and has symmetric and uniform rules, and which, like natural deduction, has a Curry-Howard interpretation.

As we can see **LP** as the fragment of intuitionistic linear logic with only the connectives '$\multimap$' and '$\otimes$', we can adapt the proof net approach to the categorial setting, as shown in [Roorda 89].

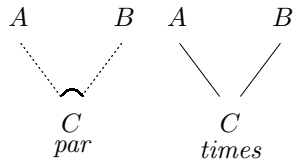
In this section we will do the following. In section 3.2.1 we will present the inductive definition of proof nets for **LP**, which allows us to generate the set $\mathcal{PN}$ of proof nets which is sound and complete with respect to the sequent formulation of **LP** but gives us no easy decision procedure. In section 3.2.2 we give the definition of proof structures, which are less restrictive than proof nets and therefore need an extra soundness criterion. We will give two such criteria in the form of conditions on labels assigned to the formulas in the proof structures.

In section 3.2.3 we will add labeling at the *semantic* level to the proof structures. The lambda term labeling will allow us to state conditions on the term assigned to the proof structure to get soundness and completeness with respect to **LP**. It will also gives us meaning recipes for our proof nets just like we had for natural deduction.

In section 3.2.4 we will add labeling at the *syntactic* level to the proof structures. This structural labeling will allow us to embed arbitrary multimodal logics in our basic **LP** proof net architecture, just like we did with the labeled deductive sequent calculus. We will prove this labeling is sound and complete with respect to any set of structural rules for a multimodal grammar and present a decision procedure for labeled proof structures.

3.2.1 Inductive definition

A proof net is a labeled graph, with two kinds of binary links which we will call *par* links and *times* links.



The par links connect premises of the same proof net, while the times links connect premises of different proof nets. The formulas A and B of a link are its premisses, while the formula C is its conclusion.

Axiom and *cut* links are special kinds of links, corresponding to the rules in the sequent identity group. An axiom link is a link with no premisses and two conclusions, while a cut link has two premisses and no conclusions.

In a proof net each formula is the conclusion of exactly one link, and the premiss of at most one link. We will call formulas which are not the premiss of any link the conclusions of the proof net.

Before we give the inductive definition of proof nets we need an auxiliary notion of the *polarity* of a formula, which can be positive or negative. Negative

formulas will correspond to sequent antecedent formulas, and positive formulas will correspond to sequent succedent formulas.

Definition 3.5 (Polarity) *The* **polarity** *of a formula is defined as follows.*

For sequents $A_0, \ldots, A_n \Rightarrow B$ *the* A_i *have a negative polarity and* B *has a positive polarity.*

For formulas $A \bullet B, A/C, C\backslash A, \Diamond A, \Box^{\downarrow} A$ *the subformulas* A *and* B *have the same polarity as the formula itself, while the* C *subformulas have the reverse polarity.*

We will write $\overset{+}{A}$ for positive formulas A, and $\overset{-}{A}$ for negative formulas A. We will often just call them succedent and antecedent formulas.

In [Girard 87] the 'pure' fragment of the proof net system is defined for the connectives '$\parr$' and '$\otimes$' (for the other connectives it has somewhat less good properties), so it will help to keep in mind that in (classical) linear logic we have the following equivalences.

$$\begin{array}{rcl} (A \otimes B)^{\perp} & \Leftrightarrow & A^{\perp} \parr B^{\perp} \\ A \multimap B & \Leftrightarrow & A^{\perp} \parr B \\ (A \multimap B)^{\perp} & \Leftrightarrow & A \otimes B^{\perp} \end{array}$$

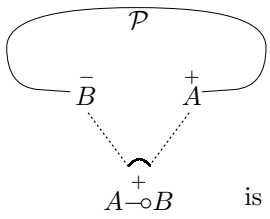
The set of proof nets $\mathcal{PN}$ is defined inductively as follows

(Ax)

$\overset{-}{A}$ $\overset{+}{A}$ is a proof net.

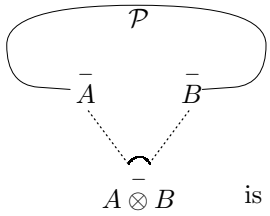
(Par) A par link connects two conclusions from the same proof net to give a new proof net in one of the following ways

$[Par\multimap]$ if $\mathcal{P}$ is a proof net with conclusions $\overset{-}{B}$ and $\overset{+}{A}$, then



is a proof net.

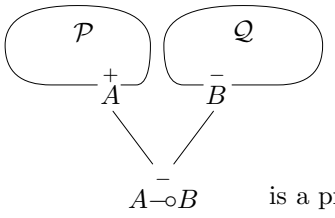
$[Par\otimes]$ if $\mathcal{P}$ is a proof net with conclusions $\overset{-}{A}$ and $\overset{-}{B}$, then



is a proof net.

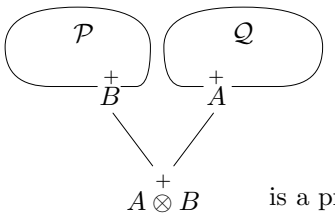
(Times) A times link connects two conclusions of different proof nets to combine them in a single proof net in one of the following ways

$[Times{\multimap}]$ if $\mathcal{P}$ is a proof net with conclusions $\overset{+}{A}$ and $\mathcal{Q}$ is a proof net with conclusion $\overset{-}{B}$, then



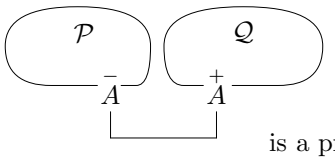
is a proof net.

$[Times{\otimes}]$ if $\mathcal{P}$ is a proof net with conclusion $\overset{+}{A}$ and $\mathcal{Q}$ is a proof net with conclusion $\overset{+}{B}$, then



is a proof net

(Cut) if $\mathcal{P}$ is a proof net with conclusion $\overset{-}{A}$ and $\mathcal{Q}$ is a proof net with conclusion $\overset{+}{A}$, then

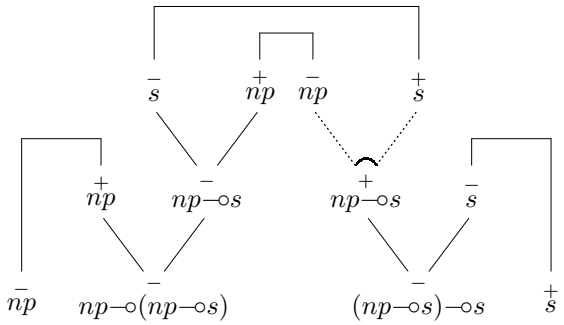


is a proof net.

The proof net calculus is sound and complete with respect to the sequent formulation of **LP** proofs. For completeness we can see that the inductive definition follows the sequent rules very closely. The soundness proof is a bit more involved as it requires induction on the number of vertices in the graph and a way to split up a proof net for the times link. We just state the theorem and refer the reader to [Roorda 89] for the proof.

Theorem 3.4 *$A_0, \ldots, A_n \Rightarrow C$ is derivable in* **LP** *iff we can construct a proof net with conclusions $\overset{-}{A_0}, \ldots, \overset{-}{A_n}, \overset{+}{C}$*

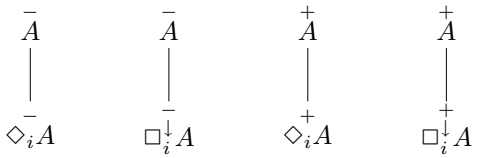
Example 3.4 *For the derivation of $np, np{\multimap}(np{\multimap}s), (np{\multimap}s){\multimap}s \Rightarrow s$ we can construct the following proof net*



We can eliminate the cut rule from the proof nets and we can restrict ourselves to atomic instances of the axiom rule, see [Girard e.a. 89, appendix A]. In the following we will restrict ourselves to cut-free proof nets with atomic axiom links.

Links for the Unary Connectives

For the unary connectives we can make no distinction between par and times links as they were used to keep track of whether the premisses of the rule came from the same or different proofs, and for the unary vocabulary we will have only one proof in both cases. This means we can extend a proof net $\mathcal{P}$ with conclusion $\pm A$ in one of the following ways



Using these links will allow us to derive $\Diamond A \Leftrightarrow A \Leftrightarrow \Box^{\downarrow} A$ for all formulas A, which in a sense trivialises the contribution of the unary connectives at the logical level.

This is not the only possibility to extend the proof net calculus to the unary vocabulary. As we may only need residuation and distribution for these connectives, we could modify the calculus to take this into account. I believe, however, that such a setup necessarily needs 'concessions' to the sequent calculus which complicates the proof net machinery considerably.

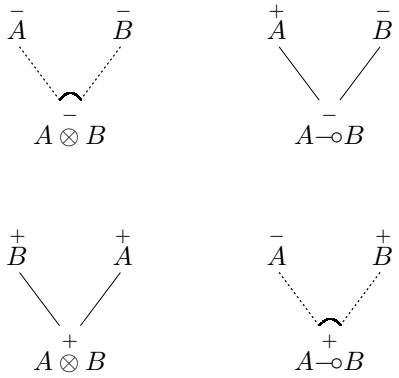
We will therefore use the links above and shift the stricter interpretation of the unary connectives entirely to the labels (see section 3.2.4). This is analogous to what we did for the labeled sequent calculus, where the logical rules for the unary connectives also allowed us to prove $\Diamond A \Leftrightarrow A \Leftrightarrow \Box^{\downarrow} A$ for all formulas A and stricter interpretation of the unary connectives was only by conditions on the labels.

3.2.2 Proof Structures

We can view the sequent calculus from two perspectives. On the one hand it is an inductive definition telling us how to generate new sequent proofs from

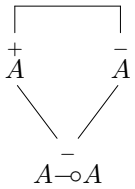
old ones. On the other hand we can 'run it backwards' as a procedure to decide whether or not a given sequent has a proof. For proof nets we want to have something like that second perspective which is why we introduce *proof structures* in this section.

A proof structure, like a proof net, is a labeled graph. In a proof structure we start with the conclusions, only one of which can have a positive polarity, and decompose them in the following way until we reach the atomic formulas. There is no cut link for proof structures.



Finally we connect each atomic formula to a formula of opposite polarity by an axiom link. Like in proof nets, each formula is the conclusion of exactly one link, and the premiss of at most one link.

Though the set of proof nets $\mathcal{PN}$ is clearly a subset of the set of proof structures $\mathcal{PS}$, not all proof structures are proof nets. The following proof structure for instance cannot be generated from the proof net definition



and indeed the corresponding sequent $(A{\multimap}A \Rightarrow)$ is not derivable. While constructing a proof structure corresponding to a sequent is easy, we need an extra criterion to identify the proof nets among the proof structures and to determine if the corresponding sequent is derivable. [Roorda 89] gives several such criteria, including the acyclicity and connectedness criterion which will concern us in section 4.1. In the next two sections we will look at lambda term labeling and structural labeling as soundness criteria.

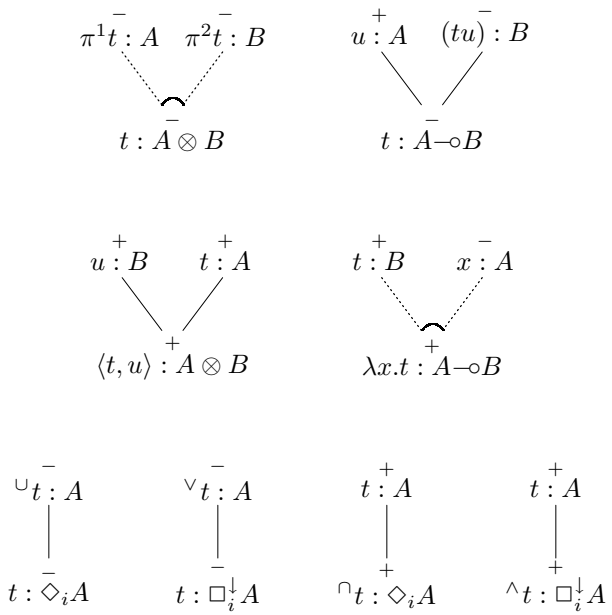
3.2.3 Lambda Term Labeling

A simple way of determining if a proof structure is **LP** valid is to label it with lambda terms, and to check if the lambda term assigned to the succedent type can be generated from the definition of linear lambda terms on page 21. This

is a bit unsatisfactory, as we have seen that the lambda terms correspond to natural deduction proofs. This would make the natural deduction proofs the 'real' proof objects, and the proof structures just a way of enumerating a priori plausible candidates for natural deduction proofs.

We are interested in the lambda terms however because they form our interface to natural language semantics. We will present the lambda term labeling for proof structures, and use it to show a Curry-Howard isomorphism holds for proof nets as well.

Initially all negative conclusions of the proof structures are assigned a fresh variable, and the positive conclusion a metavariable. Decomposition is as follows.



When we link two atomic formulas by an axiom link we unify the labels assigned to them. The unification of two terms is undefined if one is a subterm of the other. In [Roorda 89] the following theorem is proved.

Theorem 3.5 *A proof structure is a proof net iff the lambda term t assigned to the positive conclusion of the proof net satisfies the following conditions*

PN(1) the proof structure has precisely one positive conclusion.

PN(2) all unifications of the terms can be performed.

PN(3) if t contains a subterm $\lambda x.u$ then x occurs in u and x does not occur outside u.

PN(4) every variable assigned to a negative conclusion of the proof structure occurs in t.

PN(5) every subterm of t 'counts for one' in t.

The definition of counts for one is a definition of what counts as one occurrence of a subterm. It takes care of the fact that we want to count the two occurrences of the subformula u in $\langle \pi^1 u, \pi^2 u \rangle$ as one.

Theorem 3.6 *There is a 1-1 correspondence between linear lambda terms and proof nets.*

Proof We have a proof that there is a natural deduction proof iff there is a sequent proof (theorem 3.2), and a proof that there is a sequent proof iff there is a proof net (theorem 3.4). What we still have to prove is that different proof nets correspond to different lambda terms, and that different lambda terms correspond to different proof nets. Obviously a proof net can get assigned only one lambda term. We can also use the links above to *generate* a proof net from a lambda term, and see that in every case different lambda terms get assigned different links. □

3.2.4 Structural Labeling

Moortgat [Moortgat 96] proposes a different kind of labeling to address the full expressivity of the logic. It should be contrasted to earlier proposals for structural labeling like [Moortgat 90a] and [Morrill 95] which suffered from incompleteness problems.

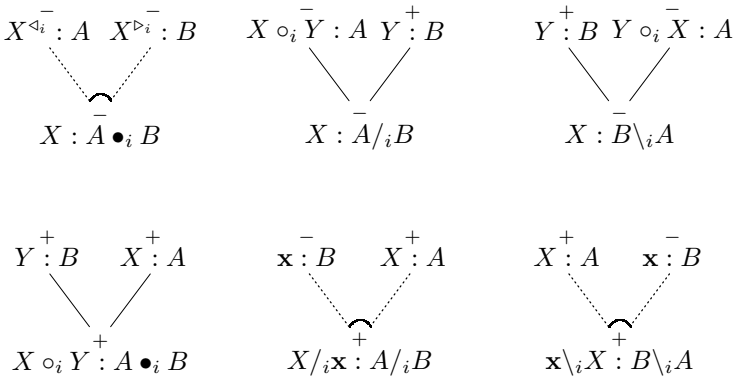
Though there are similarities with the lambda term labeling of the previous section, the labeling in this section is at a different level of linguistic description: at the structural (syntactic) level while the lambda term labeling was at the semantic level.

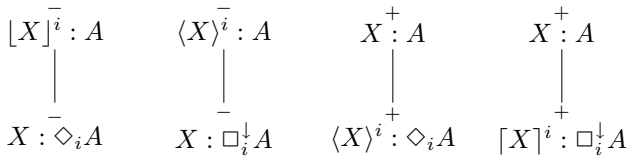
Definition 3.6 *Over a countably infinite set $\mathcal{V}$ of structure variables, we define the set of structure labels $\mathcal{L}$ as follows*

$\mathcal{L} ::= \mathcal{V} \mid \langle \mathcal{L} \rangle^i \mid \mathcal{L} \circ_i \mathcal{L} \mid \lfloor \mathcal{L} \rfloor^i \mid \lceil \mathcal{L} \rceil^i \mid \mathcal{V} \backslash_i \mathcal{L} \mid \mathcal{L} /_i \mathcal{V} \mid \mathcal{L}^{\triangleleft_i} \mid \mathcal{L}^{\triangleright_i}$

We call the constructors $\lfloor . \rfloor^i, \lceil . \rceil^i, . \backslash_i ., ./_i ., .^{\triangleleft_i}$ and $.^{\triangleright_i}$ *auxiliary* constructors. Their purpose will be to check the sublinear constraints.

The negative formulas are initially assigned a structural variable label, the positive formula a metavariable Z then decomposed as follows.





From the decomposed formulas we generate a proof structure as usual by connecting the atomic formulas by axiom links.

We can see that the labels assigned to the antecedent formulas start as variables, and grow during decomposition. The labels assigned to the succedent formulas start as a metavariable which gets partially instantiated during the formula decomposition. This means unification is one way only, and is a definite advantage over other labeling strategies.

Adapting the proof of theorem 3.5 to the structural labeling is unproblematic, but what we want from this labeling is not soundness and completeness with respect to **LP** but with respect to arbitrary multimodal logics. We do this by defining a set of conversions on the labels, which will check the sublinear constraints on derivability. We have one conversion for each connective.

Residuation conversions

$$
\begin{array}{cc}
X^{\triangleleft_i} \circ_i X^{\triangleright_i} \to X & [\mathrm{Res}\bullet_i] \\
X \circ_i Y/_i Y \to X & [\mathrm{Res}/_i] \\
Y\backslash_i Y \circ_i X \to X & [\mathrm{Res}\backslash_i] \\
\langle \lfloor X \rfloor^i \rangle^i \to X & [\mathrm{Res}\Diamond_i] \\
\lceil \langle X \rangle^i \rceil^i \to X & [\mathrm{Res}\Box_i^{\downarrow}]
\end{array}
$$

Further, there will be one structural label conversion for each structural rule in the sequent calculus.

A structural rule

$$\frac{\Gamma \Rightarrow Z[X] : C}{\Gamma \Rightarrow Z[Y] : C}[SR]$$

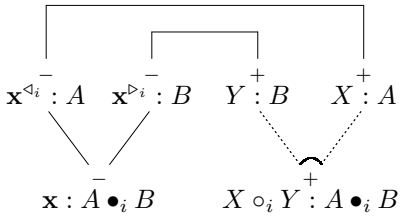
will translate to a label conversion $X \to Y$.

We call the label on the right hand side of the conversion a *redex* and the label on the left hand side its *contractum*. We will write $\twoheadrightarrow$ (reduces to) for the transitive, reflexive closure of $\to$.

A structure label is *normal* if it does not contain any auxiliary constructors, i.e. contains only structure variables and the constructors $\langle . \rangle^i$ and $. \circ_i .$. The normal labels are precisely those labels we had for the labeled sequent calculus. We will call a structure label which does not reduce to a normal form *irreducible*.

This is the same terminology as with the lambda conversions, but we trust no confusion will be possible. It is also important to note a few differences with the lambda conversions. First, not every label has a normal form. Some of the auxiliaries may turn out to be irreducible and this will mean some of the sublinear conditions can not be met. Second, in the presence of structural rules a normal form can be further converted (it is not unique). And third the residuation conversions are not always equivalences applied in one direction, but can be real inclusions.

Example 3.5 *We said earlier that we can restrict ourselves to proof nets with only atomic instances of the axiom rule. The label reductions should therefore allow us to produce a normal label for those. For the product formula, we can do so as follows*

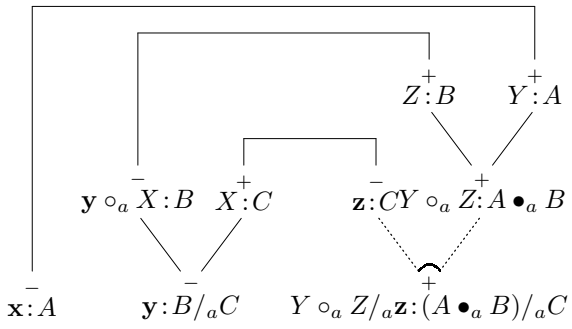


Using the non-atomic axiom link would produce a label $\mathbf{x}$ *for the succedent type immediately. The unifications* $X = \mathbf{x}^{\triangleleft_i}$ *and* $Y = \mathbf{x}^{\triangleright_i}$ *will give us a succedent label* $\mathbf{x}^{\triangleleft_i} \circ_i \mathbf{x}^{\triangleright_i}$*. We can now use the residuation conversion for the product formula to this label to get* $\mathbf{x}^{\triangleleft_i} \circ_i \mathbf{x}^{\triangleright_i} \rightarrow \mathbf{x}$*.*

We already noted restriction to atomic instances of the axiom rule would produce eta expanded proofs. It is therefore no coincidence this reduction is similar to the eta equivalence for lambda terms $\langle \pi^1 t, \pi^2 t \rangle =_\eta t$*.*

Example 3.6 *We show how the labeling, together with the set of reductions allows us to derive* $A, B/_a C \Rightarrow (A \bullet_a B)/_a C$ *if we have an associativity postulate for mode* a*. Using the labeling from [Morrill 95], we will fail to derive this theorem.*

The (only) labeled proof structure for this sequent looks as follows



The unifications $X = \mathbf{z}$*,* $Y = \mathbf{x}$ *and* $Z = \mathbf{y} \circ_a X$ *give us a succedent label* $\mathbf{x} \circ_a \mathbf{y} \circ_a \mathbf{z}/_a \mathbf{z}$*.*

In **NL** *it has no derivation. None of the residuation conversions can be applied (it is not of the right form for the Res/ conversion), meaning that without any structural postulates for* a *we cannot reduce the succedent label into a normal label.*

In **L***, however, we have a proof for this sequent. This would mean that with the associativity postulates, we should be able to reduce the succedent label to a normal label. We can indeed generate the following reduction sequence*

$$\begin{array}{ll} \mathbf{x} \circ_a \mathbf{y} \circ_a \mathbf{z}/_a \mathbf{z} & \rightarrow_{Ass} \\ \mathbf{x} \circ_a \mathbf{y} \circ_a \mathbf{z}/_a \mathbf{z} & \rightarrow_{Res/} \\ \quad \mathbf{x} \circ_a \mathbf{y} & \end{array}$$

Soundness and Completeness of the Structural Labeling

The remainder of this section we will prove the labeled proof nets, together with the label conversions are sound and complete with respect to the labeled sequent calculus with an arbitrary structural rule component.

Theorem 3.7 *$X_1 : A_1, \ldots, X_n : A_n \Rightarrow Z : C$ iff there is a proof net with conclusions $X_1 \overset{-}{:} A_1, \ldots, X_n \overset{-}{:} A_n, Z' \overset{+}{:} C$ with $Z' \twoheadrightarrow Z$ and Z normal.*

Proof We will prove that for two reformulations of the labeled sequent calculus which according to lemma 3.8 on page 36 are equivalent we have that one is equivalent to the normal labeled sequent formulation and the other to the labeled proof nets. By transitivity we therefore have that the labeled proof nets and the labeled sequent calculus derive the same theorems.

For the first sequent calculus $\Rightarrow_1$ we will allow the antecedent types arbitrary labels instead of only variables. This will remove the need for substitutions with the left rules for the divisions. We also replace the structural rules by conditions on the labels. The reformulation does not allow structural rules to be performed just before $[R\bullet_i]$, $[L/_i]$, $[L\backslash_i]$, $[R\diamond_i]$ or $[L\Box_i^{\downarrow}]$ but this does not affect derivability, as we can always apply the structural rule *after* one of these rules. We can then reformulate the labeled sequent calculus from section 2.3 on page 14 as follows. The notation $Z' \twoheadrightarrow Z : C$ meaning label Z' of C has to be reduced to Z by a number of structural conversions before we can apply the rule.

Identity

$$\frac{}{X : A \Rightarrow_1 X : A}[Ax]$$

Binary Connectives

$$\frac{\Gamma, \mathbf{x} : A, \mathbf{y} : B \Rightarrow_1 Z' \twoheadrightarrow Z[\mathbf{x} \circ_i \mathbf{y}] : C}{\Gamma, X : A \bullet_i B \Rightarrow_1 Z[X] : C}[L\bullet_i] \qquad \frac{\Gamma \Rightarrow_1 X : A \quad \Delta \Rightarrow_1 Y : B}{\Gamma, \Delta \Rightarrow_1 X \circ_i Y : A \bullet_i B}[R\bullet_i]$$

$$\frac{\Delta \Rightarrow_1 Y : B \quad \Gamma, X \circ_i Y : A \Rightarrow_1 Z : C}{\Gamma, \Delta, X : A/_i B \Rightarrow_1 Z : C}[L/_i] \qquad \frac{\Gamma, \mathbf{y} : B \Rightarrow_1 Z \twoheadrightarrow X \circ_i \mathbf{y} : A}{\Gamma \Rightarrow_1 X : A/_i B}[R/_i]$$

$$\frac{\Delta \Rightarrow_1 Y : B \quad \Gamma, Y \circ_i X : A \Rightarrow_1 Z : C}{\Gamma, \Delta, X : B\backslash_i A \Rightarrow_1 Z : C}[L\backslash_i] \qquad \frac{\Gamma, \mathbf{y} : B \Rightarrow_1 Z \twoheadrightarrow \mathbf{y} \circ_i X : A}{\Gamma \Rightarrow_1 X : B\backslash_i A}[R\backslash_i]$$

Unary Connectives

$$\frac{\Gamma, \mathbf{x} : A \Rightarrow_1 Z' \twoheadrightarrow Z[\langle \mathbf{x} \rangle^i] : C}{\Gamma, Y : \diamond_i A \Rightarrow_1 Z[Y] : C}[L\diamond_i] \qquad \frac{\Gamma \Rightarrow_1 Z : C}{\Gamma \Rightarrow_1 \langle Z \rangle^i : \diamond_i C}[R\diamond_i]$$

$$\frac{\Gamma, \langle X \rangle^i : A \Rightarrow_1 Z : C}{\Gamma, X : \Box_i A \Rightarrow_1 Z : C}[L\Box_i^{\downarrow}] \qquad \frac{\Gamma \Rightarrow_1 Z' \twoheadrightarrow \langle Z \rangle^i : C}{\Gamma \Rightarrow_1 Z : \Box_i C}[R\Box_i^{\downarrow}]$$

Showing $\Gamma \Rightarrow Z : C$ iff $\Gamma \Rightarrow_1 Z' : C$ and $Z' \twoheadrightarrow Z$ will be an easy induction proof. The extra series of conversions is necessary because the proof $\Gamma \Rightarrow Z : C$ may end in a series of structural rules.

For the second sequent calculus $\Rightarrow_2$ we first observe that the standard rules for $[L\bullet_i]$, $[R/_i]$, $[R\backslash_i]$, $[L\Diamond_i]$, $[R\Box_i^{\downarrow}]$ and $[SR]$ involve *pattern matching* on the succedent label. The rule $[L\bullet_i]$ for example is only applicable if the succedent label contains a sublabel where the two fresh variables assigned to the formulas A and B occur in the right position. By extending the set of labels to the ones we had for the labeled proof nets, and adding the same reductions we get a labeled sequent calculus which is equivalent to the labeled proof net calculus.

We will use the auxiliary constructors to check these conditions on the succedent label. We move the structural rule component of the calculus to the label conditions. The sequent calculus with the new labeling will then look like this.

Identity

$$\frac{}{X : A \Rightarrow_2 X : A}[Ax]$$

Binary Connectives

$$\frac{\Gamma, X^{\triangleleft_i} : A, X^{\triangleright_i} : B \Rightarrow_2 Z : C}{\Gamma, X : A \bullet_i B \Rightarrow_2 Z : C}[L\bullet_i] \qquad \frac{\Gamma \Rightarrow_2 X : A \quad \Delta \Rightarrow_2 Y : B}{\Gamma, \Delta \Rightarrow_2 X \circ_i Y : A \bullet_i B}[R\bullet_i]$$

$$\frac{\Delta \Rightarrow_2 Y : B \quad \Gamma, X \circ_i Y : A \Rightarrow_2 Z : C}{\Gamma, \Delta, X : A/_i B \Rightarrow_2 Z : C}[L/_i] \qquad \frac{\Gamma, \mathbf{y} : B \Rightarrow_2 X : A}{\Gamma \Rightarrow_2 X/_i \mathbf{y} : A/_i B}[R/_i]$$

$$\frac{\Delta \Rightarrow_2 Y : B \quad \Gamma, Y \circ_i X : A \Rightarrow_2 Z : C}{\Gamma, \Delta, X : B\backslash_i A \Rightarrow_2 Z : C}[L\backslash_i] \qquad \frac{\Gamma, \mathbf{y} : B \Rightarrow_2 X : A}{\Gamma \Rightarrow_2 \mathbf{y}\backslash_i X : B\backslash_i A}[R\backslash_i]$$

Unary Connectives

$$\frac{\Gamma, \lfloor X \rfloor^i : A \Rightarrow_2 Z : C}{\Gamma, X : \Diamond_i A \Rightarrow_2 Z : C}[L\Diamond_i] \qquad \frac{\Gamma \Rightarrow_2 Z : C}{\Gamma \Rightarrow_2 \langle Z \rangle^i : \Diamond_i C}[R\Diamond_i]$$

$$\frac{\Gamma, \langle X \rangle^i : A \Rightarrow_2 Z : C}{\Gamma, X : \Box_i A \Rightarrow_2 Z : C}[L\Box_i^{\downarrow}] \qquad \frac{\Gamma \Rightarrow_2 Z : C}{\Gamma \Rightarrow_2 \lceil Z \rceil^i : \Box_i C}[R\Box_i^{\downarrow}]$$

We can observe the following about these rules

- The constructors $.^{\triangleleft_i}$, $.^{\triangleright_i}$ and $\lfloor . \rfloor^i$ function as 'fresh constant generators'.
- The succedent label Z has every label occurring in the proof as a sublabel (proof net condition 4 for the structure labels).
- The rules $[R/_i]$ and $[R\backslash_i]$ guarantee that for subterms $\mathbf{y}\backslash_i X$ and $X/_i \mathbf{y}$ $\mathbf{y}$ occurs in X (proof net condition 3 for the structure labels)
- All **LP** theorems are provable for some succedent label Z.
- The labels are propagated in exactly the same way as in the proof structure decomposition. This means the succedent label Z will be the same in both proof systems.

By these observations, we have that both systems derive all **LP** theorems and both systems have the same succedent label and set of reductions applicable to it. So clearly they must be equivalent □

We still have to prove the lemma that $\Rightarrow_1$ and $\Rightarrow_2$ derive the same theorems. We will first repeat the label reductions, together with the conditions on which sequent rule they check to make the proof a bit easier to follow

Residuation conversions

$X^{\triangleleft_i} \circ_i X^{\triangleright_i} \to X$	condition on $[L\bullet_i]$
$X \circ_i Y/_i Y \to X$	condition on $[R/_i]$
$Y\backslash_i Y \circ_i X \to X$	condition on $[R\backslash_i]$
$\langle \lfloor X \rfloor^i \rangle^i \to X$	condition on $[L\Diamond_i]$
$\lceil \langle X \rangle^i \rceil^i \to X$	condition on $[R\Box_i^{\downarrow}]$
$X \to Y$	corresponding structural rule

Lemma 3.8 $\Gamma \Rightarrow_1 Z : C \iff \Gamma \Rightarrow_2 Z' : C$ *and* $Z' \twoheadrightarrow Z$ *with* Z *normal.*

Proof

($\Rightarrow$) Induction on the length of the proof. We construct a sequence of reductions $Z_0 \to \ldots \to Z_n$ such that Z_n is normal, and the succedent label of the original proof.

$n = 1$ Trivial, we have the same proof and an empty reduction sequence.

$n > 1$ We look at the last rule of the proof, and assume the induction hypothesis holds for all premisses of the rule.

$[L\bullet_i]$ We get the new proof by replacing all occurrences of $\mathbf{x}$ in the proof by $\mathbf{z}^{\triangleleft_i}$ and all occurrences of $\mathbf{y}$ by $\mathbf{z}^{\triangleright_i}$. Appending the reductions applied before this rule to our previous reduction sequence will give us a reduction sequence to a label Z_n with as only redex an occurrence of $\mathbf{z}^{\triangleleft_i} \circ_i \mathbf{z}^{\triangleright_i}$ which reduces in one step to a normal label Z_{n+1}.

$[R\bullet_i]$ We can just append the sequences $X_0 \to \ldots \to X_n$ and $Y_0 \to \ldots \to Y_n$ to get the reduction sequence $X_0 \circ_i Y_0 \to \ldots \to X_n \circ_i Y_0 \to \ldots \to X_n \circ_i Y_n$, and because by induction hypothesis both X_n and Y_n are normal the resulting label is normal.

$[R/_i]$ By induction hypothesis we have a reduction sequence $Z_0 \to \ldots \to Z_n$ to a normal label Z_n to which we can apply the sequence of conversions used before this rule to get a label Z_{n+k}, which for the old rule to be applicable must be of the form $Z_{n+k} = X \circ_i \mathbf{y}$. Applying rule $[R/_i]$ will give us a succedent label $X \circ_i \mathbf{y}/_i \mathbf{y}$ which reduces in one step to label Z_{n+k+1}, which is normal.

$[R\backslash_i]$ Symmetric.

$[L/_i], [L\backslash_i]$ By induction hypothesis we have two reduction sequences $Y_0 \to \ldots \to Y_n$ to a normal label Y_n and $Z_0 \to \ldots \to Z_n$ to a normal label Z_n. The formula A is assigned a label $X \circ_i Y_n$ (resp. $Y_n \circ_i X$), so Y_n is a sublabel of Z_0. We replace all occurrences of Y_n in the proof by Y_0 (otherwise the rule cannot be applied with the new labeling). Then we have a succedent label $Z_0[Y_0]$ to which we can apply the reduction sequence $Z_0[Y_0] \to \ldots \to Z_0[Y_n] \to \ldots \to Z_n[Y_n]$, which is normal.

$[L\Diamond_i]$ We already have a proof with succedent label Z_0 which reduces to $Z_n = Z[\langle X\rangle^i]$. After applying any structural rules, we replace all occurrences of X in the proof by $\lceil X\rceil^i$, we then have $Z_{n+k} = Z[\langle\lceil X\rceil^i\rangle^i]$ and $Z_{n+k+1} = Z[X]$.

$[R\Box_i^{\downarrow}]$ By induction hypothesis we have a reduction sequence $Z_0 \to \ldots \to Z_n$ and for the rule to be applicable a sequence of structural rules $Z_n \to \ldots \to Z_{n+k} = \langle Z'\rangle^i$. Applying rule $[R\Box_i^{\downarrow}]$ gives us $Z_{n+k} = \lfloor\langle Z'\rangle^i\rfloor^i$ where Z_{n+k+1} is Z'.

$[R\Diamond_i], [L\Box_i^{\downarrow}]$ Trivial.

($\Leftarrow$) For the opposite direction we first transform the original proof of $\Gamma \Rightarrow_2 Z' : C$ by a series of rule permutations into an equivalent proof where the rules $[\bullet_i]$ and $[\Diamond_i]$ are applied in the 'right' position. We can find the right position for these rules by looking at the reduction sequence $Z' \twoheadrightarrow Z$; if it is the last residuation conversion, we move it to the bottom of the proof, if not we move it to right before the next conversion of which the $\bullet_i$ or $\Diamond_i$ redex is a subterm. We also move all left rules which apply to a formula of which the product formula is a subformula down with the product formula. Compare this to uniform sequent proofs or the calculus $\mathbf{L}*$ in [Hendriks 93], where it is proved that this rule permutation is always possible.

We now assign, by induction on the new $\Rightarrow_2$ proof, a normal label to every formula in order to produce a $\Rightarrow_1$ proof. We do this by applying a *subsequence* of the reduction sequence to the labels of the old proof. By a subsequence of a reduction sequence $Z' \twoheadrightarrow Z$ we will mean all conversions applying to a sublabel of Z' in the sequence until some point, in the same order as in the original sequence.

It is important to note at this point that the auxiliary constructors function as 'boundaries', meaning that every conversion is applied either fully inside or fully outside of these constructors.

$[Ax]$ We have an axiom $X : A \Rightarrow_2 X : A$. If X is normal, we use the same label for the new proof, otherwise we use the applicable conversions from the reduction sequence until we reach a label containing only the auxiliary constructors $.^{\triangleleft_i}$, $.^{\triangleright_i}$ and $\lfloor.\rfloor^i$ which we consistently replace by fresh variables, producing a normal label X' and a proof $X' : A \Rightarrow_1 X' : A$

$[L\bullet_i]$ We have assigned a normal label Z containing $\mathbf{x}$ and $\mathbf{y}$ to the premiss of the rule. The proof transformation guarantees that if we can at this point apply a structural rule to the label Z' of the old proof, we can also apply it here (this rule is 'bounded' either by the following conversion or because it is the final logical rule in the proof). So we have a reduction sequence $Z \twoheadrightarrow X[\mathbf{x} \circ_i \mathbf{y}]$ containing only structural rules, meaning the premiss is of the right form for the rule in the $\Rightarrow_1$ system. The conclusion of the rule is obtained by a subsequence which *extends* the previous subsequence, and therefore of the right form for the induction.

$[L\Diamond_i]$ Analogous.

$[R/_i]$ The subsequence we assign to the conclusion of the rule extends the subsequence assigned to the premiss of the rule by a series of structural

rules followed by the residuation conversion for $./_i.$. This means we can apply the same rule in the $\Rightarrow_1$ proof.

$[R\backslash_i], [R\Box_i^{\downarrow}]$ Analogous.

$[L/_i], [L\backslash_i]$ We have applied the same subsequence of reductions to the Y label in both branches of the proof, so we can apply the rule.

$[L\Box_i^{\downarrow}], [R\bullet_i], [R\Diamond_i]$ Trivial, the rules are the same in both systems. □

3.2.5 Decision Procedure

For a sequent $\Gamma \Rightarrow A$ we now have a three step decision procedure.

1. All formulas in Γ are assigned negative polarity and a fresh structure label. A is assigned positive polarity and a fresh metavariable Z. All formulas are decomposed by applying the links on page 31 until we reach atomic formulas.
2. We link the atomic formulas with axiom links, unifying the labels.
3. We search for a reduction sequence $Z \twoheadrightarrow Z'$ with Z' a normal label.

Step 1 is deterministic and can be done in $O(n)$ time for a sequent with n connectives. Compared to the other steps in the algorithm, the computational cost of this step can be neglected.

In general for $2n$ atomic formulas, there will be $O(n!)$ linkings possible at step 2, which is computationally very bad. In the current framework however, it is the best possible (complete) strategy, as there are $n!$ natural deduction proofs of

$$x_1 : A/_i A, \ldots, x_n : A/_i A, y : A \vdash t : A$$

in **LP** (one for each permutation of the x_i).

The complexity of the label reductions at step 3 is a bit unclear to me at the moment, because of the large variety of structural postulates possible. It will at least be $O(n!)$, because again in **LP** where we have both associativity and commutativity we may need to consider all permutations of the structural variables.

The next part of this paper will be devoted to ways to reduce the number of linkings for large classes of proof nets. While we will always have an $O(n!)$ worst case algorithm, we can often decide early on if the proof structure we are constructing will *not* be a proof net. This will result in an algorithm which is in practice quite manageable. As suggested by steps two and three in our algorithm, we will do this in two different ways.

- We can look at the *graph* to decide if the proof is **LP** valid. All the logics we consider preserve **LP** validity, so if the proof is invalid in **LP** it will be invalid in all the logics we consider. We will follow this approach in chapter 4, where we look at some graph-theoretic properties of proof nets.
- We can look at the *labels* while we are still constructing the proof net to decide whether it will be possible to reduce the succedent label to a normal label. We will follow this approach in chapter 5.

Chapter 4

Graph Conditions

In this chapter we will look at strategies for determining early that the proof structure we are constructing will not be a proof net.

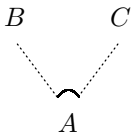
We will see that for two of the logics in the categorial landscape we can formulate graph theoretic soundness criteria. For **LP** all valid proof nets will be acyclic and connected, and all **L** valid proof nets have in addition to being acyclic and connected only planar axiom links.

While we saw in the previous chapter that the label reductions alone provided soundness and completeness, the label reductions, in general, cannot be performed in polynomial time, whereas the graph conditions can be checked quite efficiently (we will give an $O(n^2)$ algorithm for testing acyclicity and connectedness, and an $O(1)$ algorithm for testing planarity). We can use these conditions after each axiom link to test whether the (partial) proof structure satisfies the soundness criteria and so reduce the total search space.

4.1 Acyclicity and Connectedness

4.1.1 Observations of Proof Structures

An *observation* of a proof structure is an (ordinary) graph, we get by replacing all par links

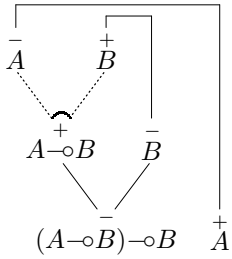


by one of the following (unary) links

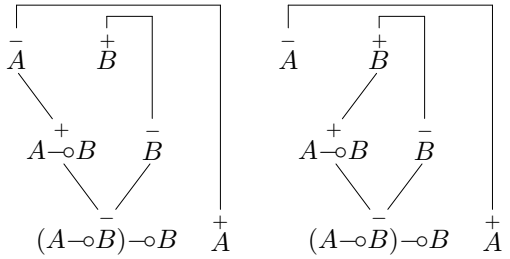
For **LP** the following theorem holds.

Theorem 4.1 (Roorda) *A proof structure $\mathcal{S}$ is a proof net iff all its observations are acyclic and connected (ACC).*

The following proof structure, for example, is not a proof net



as is has two observations



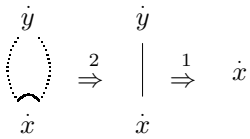
where the observation on the right is both cyclic and nonconnected.

We will call a proof structure acyclic and connected iff all its observations are.

4.1.2 Graph Reductions

For a proof net with n par links, there will be 2^n different observations, so naive application of the ACC criterion will give us a very inefficient algorithm. [Danos 90] gives us a better method for checking whether a proof structure is acyclic and connected. Starting with the graph of the proof structure, we apply the following reductions, until none of them is applicable.

Graph Reductions



The reductions are subject to the following conditions

($\overset{1}{\Rightarrow}$) only if $x \neq y$

($\overset{2}{\Rightarrow}$) only if the two edges come from the same link.

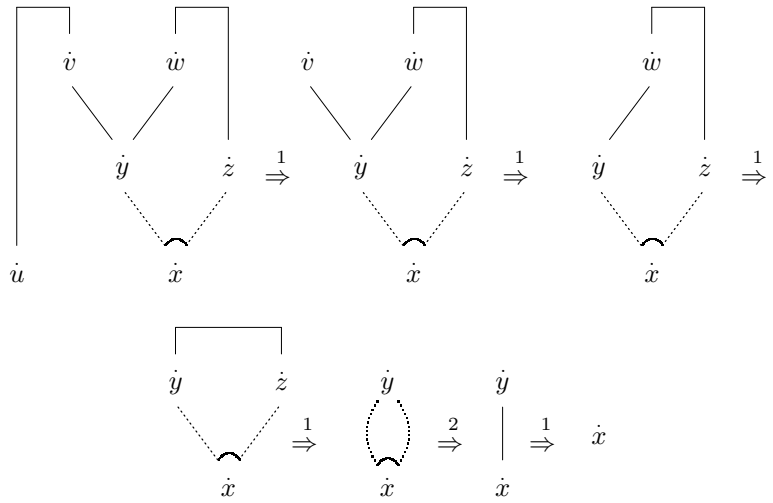
It is important to note the edges of a par link are paired, as suggested by the arc connecting them. This means that when multiple par links have the same vertex as a base, which can happen after applying some reductions, we keep track of which pairs belong together.

It is immediate that whenever it is possible to apply more than one reduction to a graph, the results will converge as the conflicting reductions will either 1)

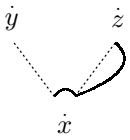
produce isomorphic graphs immediately, or 2) the order in which the reductions are applied can be reversed.

Reduction of a proof net will result in the trivial acyclic connected graph: a single vertex.

Example 4.1 *We can reduce the following proof net as follows*



Reduction of a proof structure which is not a proof net will result in a graph which is not a single vertex. The non proof net on the facing page will reduce in 4 steps to



which can be further reduced by a single 1 reduction after which no reductions are possible. This means we can never apply the 2 reduction to get rid of the par link.

Theorem 4.2 *A proof structure $\mathcal{S}$ is a proof net iff its graph reduces to a single point by applying reductions 1 and 2 above.*

Proof

($\Rightarrow$) We prove that for all proof nets its graph can be reduced to a single point. We do this by following the inductive proof net definition.

(Axiom) The proof net consists just of an axiom link, which we can reduce by a 1 reduction to a point.

(Unary) We add a unary link to a proof net which reduces to a point. To the resulting graph we can apply a 1 reduction, resulting in a point.

(Times) We have two proof nets reducing to a point. After adding the times link we can just apply the 1 reduction two times, and the resulting graph consists of a single point.

(Par) We have a proof net which reduces to a single point. Adding the par link will give us a 2 redex. Applying a 2 reduction followed by a 1 reduction gives us a single point.

(Cut) We have two proof nets reducing to a point. We can apply a 1 reduction to the cut link, which gives us a single point.

($\Leftarrow$) It is easy to see that whenever we apply one of the reductions to reduce a proof structure $\mathcal{S}$ to a proof structure $\mathcal{S}'$ then $\mathcal{S}$ is acyclic and connected iff $\mathcal{S}'$ is. A single vertex is acyclic and connected, so if $\mathcal{S}$ reduces in a number of steps to a single vertex it is acyclic and connected. Application of theorem 4.1 gives us that $\mathcal{S}$ is also a proof net. □

Beginning with a proof structure with n par links and m times links, we can first reduce all times links in $O(m)$ time. Then we can find a par link to which reduction 2 and 1 can be applied in at most $O(n)$ time. If such a par link cannot be found, we fail because the proof structure is nonconnected. Reducing all par links will then take $n + (n-1) + \ldots + 1 (= \frac{1}{2}n(n+1))$ time. The maximum time for determining a proof structure is a proof net will then be $O(\frac{1}{2}n(n+1) + m) = O(n^2)$.

4.1.3 Incremental Graph Reductions

Another important advantage of the graph reduction strategy described above, is that we can incrementally reduce the proof structure we are generating. After each axiom link, we reduce the proof structure as far as possible and check whether we have cyclic or nonconnected parts.

1. Starting with the graph of decomposed formulas (without the axiom links) we assign to each vertex a *multiset* of atomic formulas at that vertex. At this point the leaves will have a singleton multiset assigned to them, and all other vertices the empty multiset.

2. We apply all 1 reductions. The multiset assigned to the result of the reduction will be the *union* of the multisets of the reduced nodes. From this point there are only par links in the graph.

3. We (nondeterministically) remove a two atomic formulas of opposite polarity from the multisets of two different vertices, unify the structure labels and add an axiom link to the graph. We make sure the axiom link does not produce a cycle, and apply a 1 reduction to it.

4. We apply a combination of reduction 2 and 1 to all 2 redexes in the graph. This can result in new 2 redexes, so we repeat this step until no 2 redexes remain.

5. We check for connectedness, and repeat from step 3 until we have a single vertex with an empty multiset of atomic formulas.

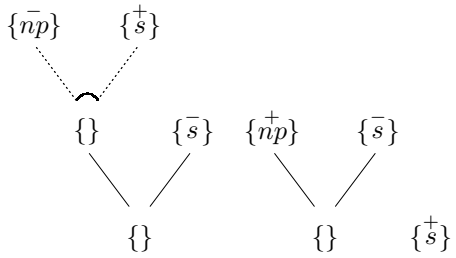
We can check for cycles and nonconnectedness in the following way

- When a vertex, which is not the base of a par link and not the only vertex in the graph, has an empty set of atomic formulas we know the proof structure will not be connected.
- When we apply an axiom link between two formulas where one is a descendant of the other, the resulting proof structure will be cyclic as when all par links between the atoms have been reduced we will have produced a cycle.

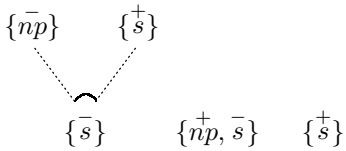
The algorithm described above still leaves us a degree of freedom in the way we select the literals at step 3. In chapter 5 I will discuss two such strategies, which are motivated by the properties of the labels.

As an illustration, we show how this algorithm gives a derivation of the simple theorem $s/(np\backslash s), np\backslash s \Rightarrow s$.

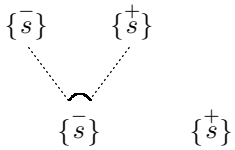
After step 1 we have the following graph



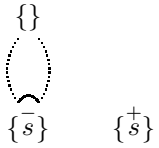
Reduction of all times links, and multiset union give us



at step 2. Step 3 is nondeterministic, but we can see that linking the atomic s formulas of the par link is ruled out by both the cyclicity and connectedness check. We decide to link both np formulas, which after reduction and set union gives us



There are no 2 redexes to reduce, and it is still possible to make the graph connected, so we repeat from step 3. Should we decide to link the left s of the par link to the succedent s, the result would be nonconnected. We link it to the right s of the par link instead, which results (after reducing the axiom link and taking the union of the sets) in



where we have a single 2 redex. Reducing it will produce

$$\{\bar{s}\} \qquad \{\overset{+}{s}\}$$

which is connected as both vertices still have a nonempty atom set. Back at step 3 only one pair can be selected, and reduction gives us the single vertex with empty atom set we want.

This small example already shows how a number of linkings which would never have resulted in a proof net have been prevented. In fact, after we selected the first literal, the rest of the algorithm performed deterministically.

4.1.4 A Comparison With Sequent-Based Methods

Some results have been presented [Hodas & Miller 94], [Morrill 94a], where procedural restrictions on the order of the application of the rules in sequent proofs are used to reduce the search space dramatically. In this section I will compare these proposals with the graph reductions from the previous section.

Uniform Sequent Proofs

Hodas and Miller [Hodas & Miller 94] observe that the fragment of intuitionistic linear logic without the $[L\otimes]$ rule

$$\frac{}{A \Rightarrow A}[Ax]$$

$$\frac{\Delta \Rightarrow A \quad \Gamma, B \Rightarrow C}{\Gamma, \Delta, A{\multimap}B \Rightarrow C}[L{\multimap}] \qquad \frac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A{\multimap}B}[R{\multimap}]$$

$$\frac{\Gamma \Rightarrow A \quad \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow A \otimes B}[R\otimes]$$

has the property that sequent proofs for it can be constructed in a uniform manner.

Hodas and Miller give the following algorithm

1. If the succedent is not an atomic formula, we apply a $[R\otimes]$ or $[R{\multimap}]$. This is possible because in the absence of $[L\otimes]$ we can always transform a sequent proof into a proof where we 'wait' with the application of the $[L{\multimap}]$ rules.

2. If the succedent is an atomic formula, we (nondeterministically) select an antecedent formula and apply $[L{\multimap}]$ rules to it until we have an atomic formula. Then we apply the axiom rule. This is possible because we can always reverse the order in which two $[L{\multimap}]$ rules are applied to different formulas.

There is still the problem of *partitioning* the antecedent for the $[L{\multimap}]$ and $[R\otimes]$ rules. That is, we need to determine which formulas in the antecedent are in Γ and which are in Δ. The solution [Hodas & Miller 94] give for this problem is to pass all formulas to one branch of the proof, keeping track of which are used at each step and passing the unused formulas to the other branch where

they must all be used. Readers familiar with logic programming techniques will recognise the data structure we use for the antecedent as *difference multisets*. The antecedents will have the form $\Gamma - \Delta$ with $\Delta \subseteq \Gamma$, which will be interpreted as an antecedent containing all formulas in Γ not contained in Δ. The sequent calculus with difference multiset looks like this. If we keep in mind that for difference multisets $\Gamma - \Delta' \cup \Delta' - \Delta = \Gamma - \Delta$ holds, it is easy to see equivalence holds between the normal and difference multiset formulation of the calculus.

$$\frac{}{\Gamma \cup \{A\} - \Gamma \Rightarrow A}[Ax]$$

$$\frac{\Gamma - \Delta' \Rightarrow A \quad \Delta' \cup \{B\} - \Delta \Rightarrow C}{\Gamma \cup \{A \multimap B\} - \Delta \Rightarrow C}[L\multimap] \qquad \frac{\Gamma \cup \{A\} - \Delta \Rightarrow B}{\Gamma - \Delta \Rightarrow A \multimap B}[R\multimap]$$

$$\frac{\Gamma - \Delta' \Rightarrow A \quad \Delta' - \Delta \Rightarrow B}{\Gamma - \Delta \Rightarrow A \otimes B}[R\otimes]$$

In section 5.2 I will show how the graph reductions with a goal driven literal selection strategy will apply the axiom links in much the same way as the uniform sequent rules reach the axioms.

The reason the uniform sequent proof strategy does not work for the full logic is that if we allow the $[L\otimes]$ rule we cannot always transform a proof into a proof where the $[R\otimes]$ rule is applied before the $[L\otimes]$ rule. Even worse is that in the full logic we sometimes need to apply a $[L\multimap]$ rule before a $[R\otimes]$ rule, as in the following proof.

$$\dfrac{\dfrac{}{A \Rightarrow A}[Ax] \qquad \dfrac{\dfrac{\dfrac{}{B \Rightarrow B}[Ax] \quad \dfrac{}{C \Rightarrow C}[Ax]}{B, C \Rightarrow B \otimes C}[R\otimes]}{B \otimes C \Rightarrow B \otimes C}[L\otimes]}{A, A \multimap (B \otimes C) \Rightarrow B \otimes C}[L\multimap]$$

Applying the $[R\otimes]$ rule first would result in a sequent where one of the leaves would be either $A \Rightarrow B$ or $A \Rightarrow C$, and would not result in a proof. This means the uniform approach has incompleteness for the full logic, as it fails to derive valid theorems.

Some attempts have been made to extend this approach to the full logic (see e.g. [Janssen 95]), but these attempts have never been fully convincing.

Sequents or Proof Nets?

The sequent normalisation approach has the advantage that it excludes a large number of derivations 'by construction' (i.e. without computation), whereas when we are using the graph reduction approach we have to do some computation after each step.

The graph reduction strategy, however, is completely free with respect to the selection of the literals where the sequent methods force us to use a goal driven literal selection strategy. Though this literal selection strategy may have some advantages in the case of grammars with a relatively simple structural component (for example when we only have an associative mode), it may be impossible to reduce them 'eagerly' as I will show in section 5.2.1.

Another serious defect of the sequent approach is its inability to give a satisfactory coverage of the full logic. Though the $[L\otimes]$ rule is from a linguistic point of view perhaps relatively unimportant, completeness should be high on the agenda of any automated proof system.

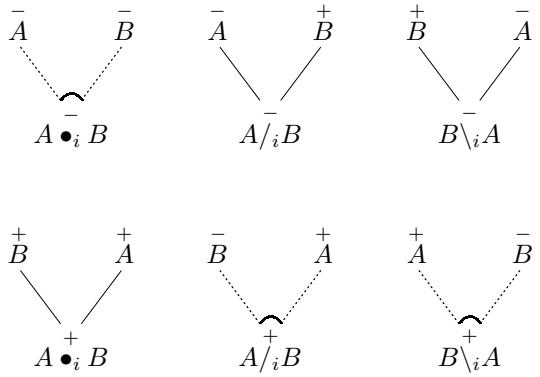
It is a bit hard to do any real comparison between the number of linkings each strategy makes, as in both cases there is a degree of freedom. In the uniform sequent proof approach we have a choice of which branch of the sequent proof to explore first, whereas in the graph reduction approach, we have the choice of which literals to select. In both cases we can be lucky by choosing the more restrictive alternative. I have nevertheless performed a benchmark test on a number of representative sentences, and compared the number of axiom links tried with both of the approaches. In almost all cases the graph reduction strategy performed slightly to significantly better (16% average).

We can conclude that the graph reduction strategy is to be preferred as it is both cleaner and faster, and gives us the freedom to use a literal selection strategy of our choice.

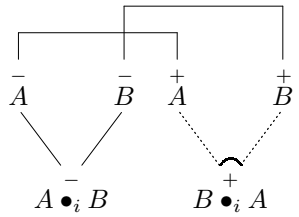
4.2 Planarity

For most natural languages there is only a small number of phenomena which require some form of commutativity. This means that we could use the Lambek calculus **L** for a large fragment of most natural languages.

Roorda [Roorda 89] gives a graph theoretic criterion which distinguishes **L** proofs from **LP** proofs. When during the formula decomposition we keep track of the order of both premisses of a binary link with respect to each other, all **L** valid proofs have *planar* (non-crossing) axiom links. We can do this by making the unfolding of antecedent and succedent formulas symmetric, as below



The minimal **LP** theorem which has no counterpart in **L** is $A\otimes B \Rightarrow B\otimes A$, and indeed it only has a nonplanar linking.



Theorem 4.3 (Planarity) *A proof net is valid in* **L** *iff all its axiom links are planar.*

We refer to [Roorda 89] for the proof of this theorem.

More generally, we can state that when all modes are *continuous* the axiom links must be planar.

Definition 4.1 *A mode i is* **continuous** *iff for all structural postulates $X \rightarrow Y$ where X has a subterm $V \circ_i W$, we have that in Y*

1) no metavariable sublabel of W precedes a metavariable sublabel of V, i.e. we preserve the order of V with respect to W.

2) there is no label Z which is not a sublabel of either V or W and which has a metavariable sublabel of V to its left and a metavariable sublabel of W to its right, i.e. no material is inserted between V and W.

According to this definition all modes involved in the following postulates are discontinuous

$$\begin{array}{rl} X \circ_c Y \rightarrow Y \circ_c X & [Com] \\ X \circ_a Y \circ_a Z \rightarrow X \circ_n Z \circ_w Y & [Wrap] \\ X \circ_n Z \circ_w Y \rightarrow X \circ_a Y \circ_a Z & [Wrap] \\ X \circ_a Y \circ_w Z \rightarrow X \circ_w Z \circ_a Y & [MxCom] \\ X \circ_w Z \circ_a Y \rightarrow X \circ_a Y \circ_w Z & [MxCom] \end{array}$$

4.2.1 String Labeling

As we already noted at the beginning of this section, planarity only holds for *parts* of natural language. There are a number of linguistic phenomena which are best analysed as discontinuous (see [Morrill 94] for a description of a number of these phenomena and their treatment in categorial grammar).

This means that for realistic grammar fragments we have to allow for partial planarity, where some of the linkings must be planar and others can violate this constraint. It is unclear to me how this can be done by constraints on the graph.

There is however an alternative to the graph theoretic planarity constraint. In [Benthem 91] it is noted that **L** has relational algebraic models, where formulas are interpreted as sets of ordered pairs. The pairs correspond to (abstract) string positions. In [Morrill 94a] this kind of labeling is already used for uniform sequent proofs.

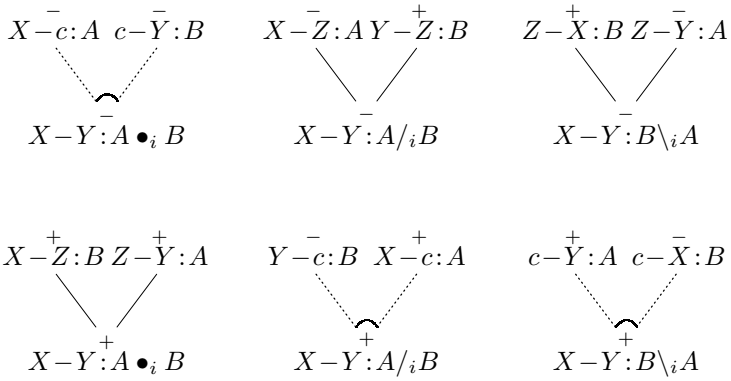
A pair of string labels $X - Y$ will correspond to the string beginning at position X and ending at position Y. This means we can append two of these pairs $X-Y$ and $Y-Z$ in constant time to $X-Z$. We can perhaps best illustrate how the string labeling works by showing the natural deduction rules for one of the implications.

$$\frac{X \cdots\cdots Y \cdots\cdots Z \quad X-Y:A/B \quad Y-Z:B}{X-Z:A}[/E] \qquad \frac{X \cdots\cdots Y \cdots\cdots c \quad [Y-c:B] \quad \vdots \quad X-c:A}{X-Y:A/B}[/I]$$

The dotted line above the rules shows the string and the relevant positions on it. The c position in the introduction rule is a hypothetical string position, which cannot be used below this rule. It is discharged by the rule just like the B formula itself.

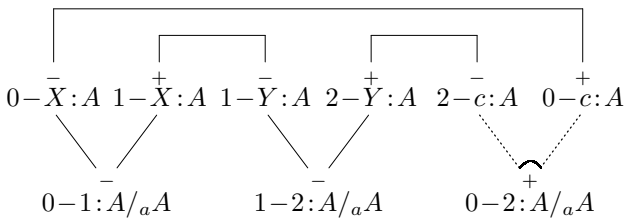
We can adapt this labeling to proof nets and use it for all modes of composition which are continuous. For the other modes we assign the subformulas a fresh pair of string positions, to indicate we have no way of telling what part of the string they correspond to.

We start with labeling the initial formulas $0-1:A_0,\ldots,n-(n+1):A_n \Rightarrow 0-(n+1):C$; the antecedent formulas are given adjoining string positions, and the succedent formula the entire string. Decomposition is as follows.

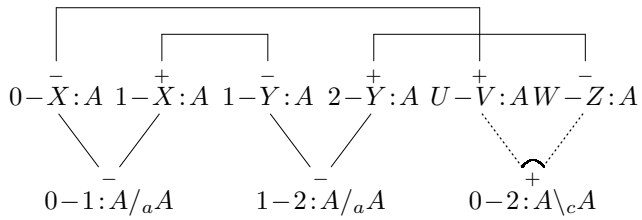


The string labels are unified with each axiom link. As each label is either a constant or a variable, this unification will only take $O(1)$ time.

Example 4.2 *Using the string labeling makes the derivation of $A/_aA, A/_aA \Rightarrow A/_aA$ fully deterministic; for any of the label pairs there is only one label pair with which it can be unified.*



For a discontinuous mode c however, we can still derive the $A/_aA, A/_aA \Rightarrow A\backslash_cA$ which is underivable in **L**



Though the string labeling is very restrictive, its usefulness is limited to modes which are continuous according to definition 4.1. This entails that adding a single mixed commutativity postulate to an otherwise continuous grammar fragment can destroy the continuity for both modes participating in the postulate.

Chapter 5

Label Reductions

This chapter we will look at the labeling algebra, and how it can be used to determine early we will not be able to reduce our succedent label to a normal label.

For the labeling algebra we have two basic options.

- We can wait with the label reductions until after all axioms have been linked. In this case the succedent label will be fully instantiated, and we will only have to reduce it once. We will call this *lazy* reduction. As the label reductions can be computationally very expensive this is sometimes the best way. It can also mean we miss chances to detect early failure when the label algebra is very restrictive.

- We can also after each axiom link test if the label conditions can be satisfied, and fail if they cannot. We will call this *eager* reduction. Eager label reductions can often prevent us from performing axiom links we know will not result in a succedent label which can be normalised.

Whereas we had efficient algorithms for the graph criteria in the previous chapter, the complexity of the label algebra is $O(n!)$ which is the same order of complexity as for constructing the rest of the proof structure. An important theme throughout this chapter will therefore be to make sure we actually have some gain over lazy evaluation when we use eager evaluation of the labels.

After some initial discussion on term rewrite systems, and the properties of the label algebra in section 5.1, I will discuss the possibility of eagerly reducing the labels for two literal selection strategies.

In the goal driven literal selection strategy in section 5.2, we start with the succedent (goal) formula and link it to an antecedent formula. Using this strategy we will have the full succedent label, but our knowledge of it will be partial.

In the data driven literal selection strategy in section 5.3, we start with an antecedent formula which has a ground label, and link it to a succedent formula. The conditions on the labels will guarantee this label is a part of the succedent label, and we can use this information to fail early if we can show every label with this sublabel is irreducible.

5.1 Properties of the Label Algebra

We can see the label calculus as a *term rewrite system* (TRS), if one with not very good properties. An introduction to term rewrite systems can be found in [Klop 92]. I will only briefly review some basic terminology which will be relevant to our discussion in this chapter.

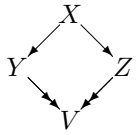
A term rewrite system is a relation '$\rightarrow$' over a set of terms, with its transitive, reflexive closure '$\twoheadrightarrow$'.

Term rewrite systems are mostly used to model a set of *equations* (statements of the form $X = Y$), whereas we use statements which are inclusions of the form $X \leq Y$. When we model a theory of equations, we have a choice of whether to use them as conversions of the form $X \rightarrow Y$ or $Y \rightarrow X$, depending on whether we consider X or Y to be 'simpler', which is not always easy to decide. An inclusion postulate like $X \circ_a Y \rightarrow X \circ_n Y$ can be used only as a conversion in one direction, as we really want to distinguish between the modes involved, which limits the possibilities.

Most work in the literature on term rewrite systems is about systems for which the following two properties hold.

Definition 5.1 (Church-Rosser) *a conversion relation $\rightarrow$ is said to be* Church-Rosser *or* confluent *if whenever we have $X \rightarrow Y$ and $X \rightarrow Z$ there is a V such that $Y \twoheadrightarrow V$ and $Z \twoheadrightarrow V$.*

As a picture



Definition 5.2 (Strong Normalisation) *If every reduction sequence in the TRS is finite the TRS is strongly normalising (SN).*

The properties SN and CR together give us that every term has a unique normal form.

The lambda term equivalences from section 3.1.3 can be used as a term rewrite system (we did this by giving the equivalences an orientation). It is both confluent and strongly normalising, as proved in [Girard e.a. 89].

The nondeterministic nature of the label conversion rules will mean that we will not have confluence as example 5.1 in the next section shows. Some work has been done on nondeterministic term rewrite systems, see for example [Kaplan 88].

For our label reductions we will have no strong normalisation either, as we allow rules of the form $X \circ_i Y \rightarrow Y \circ_i X$, where the right hand side of the rule matches the left hand side of the same conversion. This will lead to an infinite reduction sequence with a 'stupid' reduction strategy. We will show below that we have *weak normalisation* (i.e. some reduction strategy always terminates) for all label conversions satisfying the following conditions

- All conversions are *linear*. The metavariables occurring on the right hand side of the conversion occur in exactly the same quantity on the left hand side of the conversion, and no metavariable occurs more than once. This excludes for example $X \rightarrow X \circ_i Y$ and $X \circ_i X \rightarrow X$, which is justified as conversions of this kind correspond to contraction and weakening.

- The number of symbols on the right hand side of the conversion is less than or equal to the number of symbols on the left hand side of the conversions. We will discuss this condition in section 5.1.2

The table below gives a summary of the differences between TRS's and our label conversions.

TRS	**Labels**
$X = Y$	$X \leq Y$
CR	NonDet
SN	WN

5.1.1 Search Strategy

Reducing a succedent label to a normal label can be seen as searching a (possibly cyclic) connected graph, where the vertices are labels and the edges conversions. A label can only be converted in a finite number of ways, and, by our restrictions on the conversions, its size cannot increase, so we know the graph is finite.

To ensure weak normalisation, we only have to remove the cyclic paths from the search space. A standard technique to prevent getting lost in a cycle is keeping track of the nodes already visited in a *closed set*, and only visiting a node if it is not in this set. Using a closed set, we will only visit each vertex once, and because the graph is finite, computation will terminate.

Technically, using a closed set transforms a graph into a tree by removing non-tree edges from the graph. This gives our rewrite relation a bit of a procedural flavor, as we can now have a term X which converts to a term Y without an edge connecting them.

A second point is that we would like to do the least number of reductions possible. In logic programming depth first search is the standard search method. For our label conversions this has some unpleasant consequences. Depth first search is sensitive to the *order* of the conversions, as it will always select the first conversion applicable. This means it may add inessential conversions to the reduction sequence, where we are only interested in the 'important' ones.

To get around this we use breadth first search. This will guarantee that the first solution we find has the fewest number of reductions possible. For a good introduction to search techniques, and a description of the breadth first search with closed set used here see [O'Keefe 90, Chapter 2].

Example 5.1 *We will give a derivation of composition in a mixed system with both an associative mode of composition a and a nonassociative mode of composition n. We have two structural rules for associativity and one for inclusion*

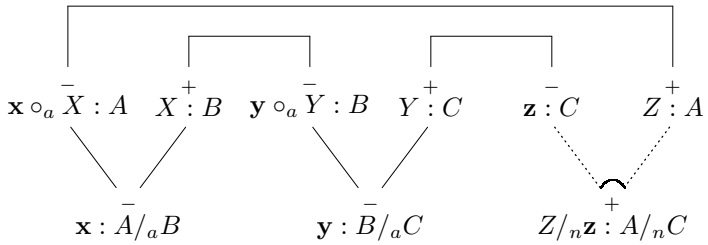
$$\frac{\Gamma \Rightarrow Z[X_1 \circ_a X_2] : C}{\Gamma \Rightarrow Z[X_1 \circ_n X_2] : C}[\text{Link a, n}]$$

$$\frac{\Gamma \Rightarrow Z[X_1 \circ_a X_2 \circ_a X_3] : C}{\Gamma \Rightarrow Z[X_1 \circ_a X_2 \circ_a X_3] : C}[\text{Ass1}] \quad \frac{\Gamma \Rightarrow Z[X_1 \circ_a X_2 \circ_a X_3] : C}{\Gamma \Rightarrow Z[X_1 \circ_a X_2 \circ_a X_3] : C}[\text{Ass2}]$$

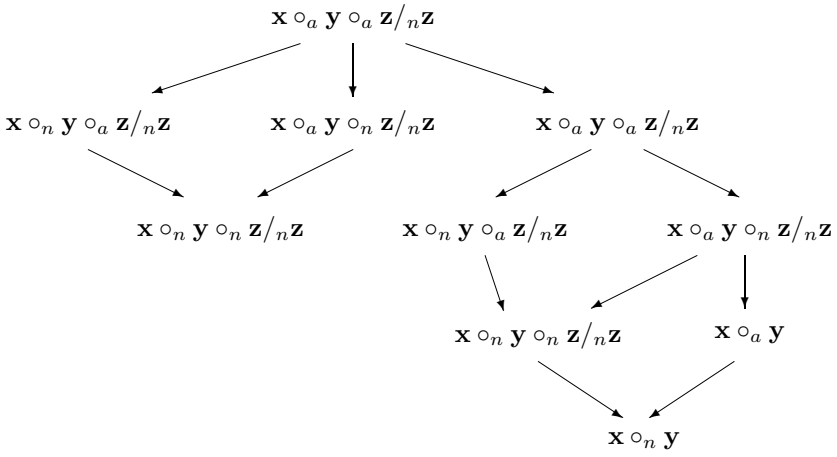
which translate to the following label rewrite rules

$$\begin{array}{ll} X \circ_a Y \rightarrow X \circ_n Y & [Link] \\ X \circ_a Y \circ_a Z \rightarrow X \circ_a Y \circ_a Z & [Ass1] \\ X \circ_a Y \circ_a Z \rightarrow X \circ_a Y \circ_a Z & [Ass2] \end{array}$$

The sequent $A/_aB, B/_aC \Rightarrow A/_nC$ *gives us the proof net below*



From the goal label $\mathbf{x} \circ_a \mathbf{y} \circ_a \mathbf{z}/_n\mathbf{z}$ *we generate the search space as follows*[1]



First we see that careless generation of the search space would have resulted in an infinite branch, with alternate application of the associativity postulates. The closed set prevents the label $\mathbf{x} \circ_a \mathbf{y} \circ_a \mathbf{z}/_n\mathbf{z}$ from reappearing in the third generation.

A second thing to note is that even this simple graph is not Church-Rosser at several points, the labels $\mathbf{x} \circ_n \mathbf{y} \circ_n \mathbf{z}/_n\mathbf{z}$ and $\mathbf{x} \circ_n \mathbf{y}$ for example can neither be further reduced and are not equal.

We can also see that there are two normal labels in the search space nl. $\mathbf{x} \circ_a \mathbf{y}$ and $\mathbf{x} \circ_n \mathbf{y}$, the first being more general than the second. Breadth first search will guarantee we will find this label first, whereas depth first search could miss the $\mathbf{x} \circ_a \mathbf{y}$ solution by selecting the other path to $\mathbf{x} \circ_n \mathbf{y}$.

5.1.2 Identity Elements and Modal Postulate 4

In the previous section we restricted ourselves to postulates where the number of symbols on the right hand side was less than or equal to the number of symbols

[1]Strictly speaking this picture is slightly incorrect as some of the nodes in the picture have more than one parent. We abstract here from the (procedural) decision of what the parent of these nodes is.

on the left hand side. Doing so allowed us to show that generating a normal form for a label can be seen as searching a finite graph.

Some structural postulates proposed in the literature don't have this property, typical examples being the following

$$\begin{array}{rl} X \to \epsilon \circ_i X & \text{Left Identity} \\ X \to X \circ_i \epsilon & \text{Right Identity} \\ \langle X \rangle^i \to \langle \langle X \rangle^i \rangle^i & \text{Modal Postulate 4} \end{array}$$

Using these we can generate reduction sequences of ever expanding labels like $\langle X \rangle^i \to \langle \langle X \rangle^i \rangle^i \to \langle \langle \langle X \rangle^i \rangle^i \rangle^i \to \ldots$. While using breadth first search guarantees that if there is a solution we will find it, if there is no solution we can get lost in an infinite branch, as the graph may not have a finite number of vertices.

If we want to allow these postulates, a solution to dealing with them would be to 'compile them away', i.e. to modify the total set of postulates in such a ways as to obtain an equivalent set of postulates where all search graphs are finite. For the identity element for example, we can take the following set of postulates

$$\begin{array}{rl} X \to \epsilon \circ_i X & \text{Left Identity} \\ \epsilon \circ_i X \to X & \text{Left Identity} \\ X \to X \circ_i \epsilon & \text{Right Identity} \\ X \circ_i \epsilon \to X & \text{Right Identity} \end{array}$$

and replace it by

$$\begin{array}{rl} \mathbf{x} \backslash_i \mathbf{x} \to \epsilon & \text{Left division} \\ \mathbf{x} /_i \mathbf{x} \to \epsilon & \text{Right division} \\ \epsilon \circ_i X \to X & \text{Left Identity} \\ X \circ_i \epsilon \to X & \text{Right Identity} \end{array}$$

The extra postulates for the divisions are actually a specialisation of the usual reductions with ϵ substituted for X.

At the moment it is unclear whether we can replace any set of such postulates by an equivalent set which does not compromise decidability, as with this kind of completion there is always the possibility that the completion procedure itself will not terminate.

Because of the algorithmic problems and the absence of any clear linguistic motivation for these postulates, we will not allow conversions which increase the number of symbols in the remainder of this chapter.

5.2 Goal Driven Label Reductions

In the goal driven literal selection strategy we will select a literal with a metavariable label which is a direct sublabel of the goal label. From the sequent perspective this corresponds to backward chaining from the conclusion of the sequent to the axioms.

We can see that this selection strategy is complete, as when the goal label has no metavariables in it (i.e. is ground) and there are still atomic formulas left in the graph, the proof structure must violate proof net condition PN(4):

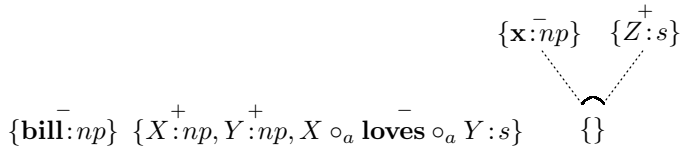
not all variables assigned to a negative formula are a sublabel of the succedent label.

The goal driven literal selection strategy has as its advantage that at any time we have the full succedent label at our disposal, which after each axiom link will become a bit more instantiated until after the final link it is ground.

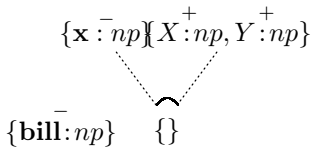
Having the full succedent label we are constructing allows us to see in many cases that the label we are constructing cannot reduce to a normal label. To do so however, we need to make guesses about the structure of the unknown part of the label. This severely reduces the attractiveness of this approach, as when the structural postulates become more complex checking whether the label conditions can be met can actually become computationally more expensive than constructing the rest of the proof net.

We also need to recompute the reduction sequence at each step, which is another important source of inefficiency.

Example 5.2 *Goal driven literal selection for* $np, (np\backslash_a s)/_a np, \Rightarrow s/_a np$. *The reduced graph is as follows*



The goal label $Z_1 = Z/_a \mathbf{x}$ *forces us to select the literal* $Z^{+}{:}s$, *which can be linked only to one literal, resulting in*



Our new goal label is $Z_2 = X \circ_a \mathbf{loves} \circ_a Y/_a \mathbf{x}$, *where we have a choice of which positive np literal to select. Assume we choose to link* $Y : np$ *to* $\mathbf{bill} : np$. *We can now see the succedent label* $Z_3 = X \circ_a \mathbf{loves} \circ_a \mathbf{bill}/_a \mathbf{x}$ *can never reduce to a normal label without some form of commutativity. We link it to* $\mathbf{x} : np$ *instead. The new succedent label can, under associativity of a, reduce to a normal label. Graph reduction gives us*

$$\{\mathbf{bill}^{-}{:}np\}\,\{X^{+}{:}np\}$$

with only one link possible. Finally, we can reduce the now fully instantiated succedent label

$$\begin{array}{rl} \mathbf{bill} \circ_a \mathbf{loves} \circ_a \mathbf{x}/_a \mathbf{x} & \rightarrow_{Ass} \\ \mathbf{bill} \circ_a \mathbf{loves} \circ_a \mathbf{x}/_a \mathbf{x} & \rightarrow_{Res/} \\ \mathbf{bill} \circ_a \mathbf{loves} & \end{array}$$

We can also see how this strategy closely follows the uniform proof of the corresponding sequent, using the rules on page 45

$$\dfrac{\dfrac{\dfrac{\{np,s,np\}-\{np,np\}\Rightarrow s \quad \{np,np\}-\{np\}\Rightarrow np}{\{np,np{\multimap}s,np\}-\{np\}\Rightarrow s}[L{\multimap}] \quad \{np\}-\{\}\Rightarrow np}{\{np,np{\multimap}(np{\multimap}s),np\}-\{\}\Rightarrow s}[L{\multimap}]}{\{np,np{\multimap}(np{\multimap}s)\}-\{\}\Rightarrow np{\multimap}s}[R{\multimap}]$$

Roughly, applying a series of right rules until we reach an atomic formula corresponds to selecting the goal literal, and applying a series of $[L{\multimap}]$ rules to linking the goal literal to selecting a negative formula.

5.2.1 Reductions With Metavariables

When we want to apply the label reductions eagerly we are faced with the fact that the succedent label is only partially instantiated. When we want to generate a search graph for this partial label, we need a way to keep this graph finite. To a metavariable Z, we can apply *any* conversion which can possibly give us new metavariables, to which we can again apply any conversion, an so on. In this section we will try to bound the search space for these cases.

Maximum Label Size

When we start with the axiom links, we can already determine the size of the succedent label after all labels have been unified. We can use this information to put a depth bound on the reductions. That is, we allow a conversion only if the size of the contractum is smaller than the maximum size of the succedent label, thus preventing we get lost in an infinite loop.

For associative, commutative modes of composition this simplistic way of trying to prevent generation of infinite search graphs is actually more expensive than constructing the rest of the proof net and lazy reduction of the label, as it will use $n!$ reductions to determine the proof structure is not a proof net.

Wildcard reductions

We can do a bit better than that by adding a special 'wildcard' constant $*$ to our language, which can take the place of any label. Any metavariable in the succedent label will be instantiated to this special constant. The wildcard constant has its own set of residuation conversions, which we obtain by substituting the constant for every proper subterm of the usual residuation conversions.

Wildcard conversions

$* \circ_i X^{\triangleright_i} \rightarrow X$	$[Product*]$
$X^{\triangleleft_i} \circ_i * \rightarrow X$	$[Product*]$
$X \circ_i */_i Y \rightarrow X$	$[RightDivision*]$
$*/_i Y \rightarrow *$	$[RightDivision*]$
$Y\backslash_i * \circ_i X \rightarrow X$	$[LeftDivision*]$
$Y\backslash_i * \rightarrow *$	$[LeftDivision*]$
$\langle * \rangle^i \rightarrow *$	$[Diamond*]$
$\lceil * \rceil^i \rightarrow *$	$[Box*]$

This will work fine as long as the structural rule component of the grammar allows it. In a grammar fragment where the only structural rules are associativ-

ity and commutativity for some modes just adding these rules may be enough. It will allow us to reduce the label Z_2 from the example above as follows

$$\begin{array}{ll} * \circ_a \mathbf{loves} \circ_a */_a \mathbf{x} & \rightarrow_{Ass} \\ * \circ_a \mathbf{loves} \circ_a */_a \mathbf{x} & \rightarrow_{RightDivision*} \\ \quad * \circ_a \mathbf{loves} & \end{array}$$

but will fail to find a reduction sequence for Z_3, which is the behavior we want.

For structural rules which are more complex, we may need to transform the entire structural rule component in order to ensure completeness.

The following structural rules for example are relatively common in categorial grammars (see e.g. [Moortgat & Oehrle 94])

$$\begin{array}{ll} X \circ_i Y \circ_j Z \rightarrow X \circ_i Y \circ_j Z & [MxAss1] \\ X \circ_i Y \circ_j Z \rightarrow X \circ_i Y \circ_j Z & [MxAss2] \\ X \circ_i Y \circ_j Z \rightarrow X \circ_j Z \circ_i Y & [MxCom1] \\ X \circ_j Z \circ_i Y \rightarrow X \circ_i Y \circ_j Z & [MxCom2] \end{array}$$

but substituting $*$ for every proper subterm of these rules will produce the following set of extra reductions

$$\begin{array}{ll} * \circ_j Z \rightarrow * \circ_i * \circ_j Z & [MA1*] \\ X \circ_i * \rightarrow X \circ_i * \circ_j * & [MA2*] \\ * \circ_j Z \rightarrow * \circ_j Z \circ_i * & [MC1*] \\ * \circ_i Y \rightarrow * \circ_i Y \circ_j * & [MC2*] \end{array}$$

which leaves us with the same problem we had in section 5.1.2: the right hand side of the conversion is more complex than the left hand side and it matches the same conversion again. The case for these postulates is more delicate, as they are used for a lot of linguistic applications, and we would expect our algorithm to be able to handle them.

In this particular case we may be able to replace the $*$ reductions above by

$$\begin{array}{ll} * \circ_j Z/_i Y \rightarrow * \circ_j Z & [/_i *] \\ Y \backslash_i * \circ_j Z \rightarrow * \circ_j Z & [\backslash_i *] \\ X \circ_i */_j Z \rightarrow X \circ_i * & [/_j *] \\ * \circ_i Y/_j Z \rightarrow * \circ_i Y & [/_j *] \end{array}$$

but when we add the postulates to an arbitrary grammar fragment, there may be complex interactions between these and other postulates, and no way of producing an equivalent set of reductions with the right termination properties.

All in all we can conclude that eager evaluation of the labels with a goal driven selection strategy is only possible under special circumstances. If we want our algorithm for this selection strategy to be sufficiently general, lazy evaluation of the labels seems to be the only option.

5.3 Data Driven Label Reductions

In the data driven literal selection strategy we select a literal with a label which is *ground*, i.e. contains no metavariables. From the sequent perspective this corresponds to forward chaining from the axioms of the sequent to the conclusion.

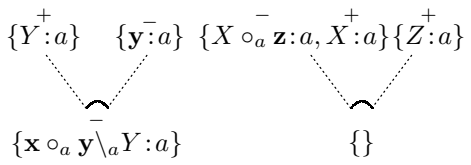
We can see this literal selection strategy is complete, because if at any time during the construction of the proof structure there is no literal with a ground label, the goal label will not be ground after all other axiom links have been performed and this will mean the proof structure is not a proof net.

Because we select only ground literals, the reduction graphs we construct will be finite, so we don't need mechanisms in addition to the closed set to guarantee termination.

Another advantage of this approach over the goal driven strategy which may not be immediately obvious is that ground literals are always the only atomic formula at a vertex. This will restrict the number of axiom links we can make from this atom without making the proof structure nonconnected.

A drawback of this approach is that there may be information in the proof structure from which we could infer the succedent label cannot be reduced to a normal label, but which is unavailable to us until this label is ground.

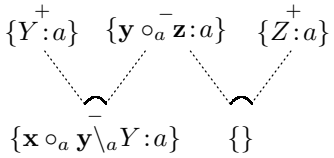
Example 5.3 *Data driven literal selection for* $a/_a(a\backslash_a a) \Rightarrow a/_a(a\backslash_a a)$. *The reduced graph is as follows.*



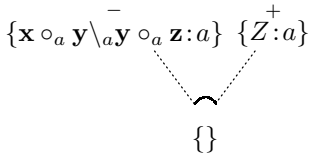
The only literal with a ground label in the graph is $\mathbf{y} : a$, *and we can link it to three positive literals. Linking it to* $Z : a$ *will give us no new ground literal, and indeed the resulting graph after reducing this link is nonconnected.*

Linking the literal to $Y : a$ *will fail on the label conditions. The only ground label in the graph after this link has been reduced will be* $\mathbf{x} \circ_a \mathbf{y}\backslash_a \mathbf{y}$ *to which no conversions are applicable. As this label is a sublabel of the succedent label, we can never remove the constructor* $.\backslash_a .$ *from the succedent label either.*

The only valid link is then to the literal $X : a$, *which will result in the following graph*



The only ground literal in the new graph is $\mathbf{y} \circ_a \mathbf{z} : a$. *Linking it to* $Z : a$ *would again produce no new ground literal, and a nonconnected graph. We link it to* $Y : a$ *instead, and check this link is valid by reducing the label* $\mathbf{x} \circ_a \mathbf{y}\backslash_a \mathbf{y} \circ_a \mathbf{z}$ *in one residuation conversion to* $\mathbf{x} \circ_a \mathbf{z}$. *We now have*



The final axiom link will give us a succedent label $\mathbf{x} \circ_a \mathbf{y}\backslash_a \mathbf{y} \circ_a \mathbf{z}/_a \mathbf{z}$ *which reduces in two residuation conversions to* $\mathbf{x}$.

5.3.1 Eager Label Reductions

Using the data driven literal selection strategy, we have only a *part* of the succedent label available, which means we have to consider the possibility that an irreducible label can be reduced to a normal form when it occurs as a sublabel of a larger label.

This is the case for the auxiliary constructors $.^{\triangleleft_i}$, $.^{\triangleright_i}$ and $\lfloor . \rfloor^i$. When we select a literal with a ground label of the form $\mathbf{x}^{\triangleleft_i}$, for example when deriving the identity for a product formula (like in example 3.5 on page 33) we cannot reduce it to a normal form (no reductions are applicable to it) but the full succedent label for this proof net $\mathbf{x}^{\triangleleft_i} \circ_i \mathbf{x}^{\triangleright_i}$ *can* be normalised. This is a drawback compared to the goal driven reductions, where we had the full succedent label and therefore could reduce all constructors eagerly.

We can apply eager evaluation to the constructors $.\backslash_i .$, $./_i .$ and $\lceil . \rceil^i$, however. When we look at the residuation conversions for these auxiliary constructors

$$\begin{array}{cc} X \circ_i Y /_i Y \to X & [\mathrm{Res}/_i] \\ Y \backslash_i Y \circ_i X \to X & [\mathrm{Res}\backslash_i] \\ \lceil \langle X \rangle^i \rceil^i \to X & [\mathrm{Res}\Box_i^{\downarrow}] \end{array}$$

we see that in order to produce a redex of one of these forms, we only have to consider reductions to sublabels of these constructors. If we cannot produce such a redex, we can fail knowing that the same constructor will be irreducible in the full succedent label.

We will call these constructors *divisions* (the constructor $\lceil . \rceil^i$ can be seen as a unary 'division' or division by a constant), and a label without these constructors *division free*. Every sublabel of a normal label must reduce to a label which is division free.

A problem with this kind of eager evaluation is that, like with the eager reductions for the goal driven strategy, it involves recomputation. After we unify it with a metavariable the ground label will occur somewhere else in the proof structure as a sublabel of one of the remaining labels. This means we have to reduce the same label again and again. Given the worst case complexity of the label algebra, this is not in general a viable strategy. What we would want is to *reduce* the label to a division free label (instead of just checking if this is possible) and unify the reduced label to the metavariable. This would prevent the redex from reoccurring elsewhere, and would only perform label reductions which would have to be performed to the full succedent label anyway, if we would be using lazy evaluation. In the next section we will look at restrictions to the label algebra which will make this incremental way of reducing the labels possible.

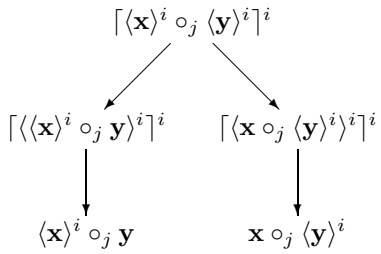
5.3.2 Incremental Label Reductions

If we want to apply the reductions incrementally, we want to guarantee that we reduce the label to a division free form which is more general than or equivalent to all other division free forms of that label. This is not possible for every set of structural postulates.

Example 5.4 *We cannot apply incremental reduction to the following set of postulates.*

$$\langle X\rangle^i \circ_j Y \to \langle X \circ_j Y\rangle^i \quad [K1]$$
$$X \circ_j \langle Y\rangle^i \to \langle X \circ_j Y\rangle^i \quad [K2]$$

as (part of) the search space for the label $\lceil\langle \mathbf{x}\rangle^i \circ_j \langle \mathbf{y}\rangle^i\rceil^i$ *is as follows*



The two branches in the search space produce different division free labels, and we have neither $\langle \mathbf{x}\rangle^i \circ_j \mathbf{y} \twoheadrightarrow \mathbf{x} \circ_j \langle \mathbf{y}\rangle^i$ *nor* $\mathbf{x} \circ_j \langle \mathbf{y}\rangle^i \twoheadrightarrow \langle \mathbf{x}\rangle^i \circ_j \mathbf{y}$.

Both labels do *reduce to a common normal form* $\langle \mathbf{x} \circ_j \mathbf{y}\rangle^i$*, but if we place the label in a larger context (for example selecting the left branch in the search space could later produce* $\langle \mathbf{x}\rangle^i \circ_j \mathbf{y}/_j\mathbf{y}$ *where any other solution would be irreducible.*

Though the reductions converge, eager reduction would force us to make a decision which has consequences for other reductions.

Proving a Label Can Be Reduced Eagerly

A way to ensure a ground label X allows eager reduction is to generate the full search space for the label, filter out those labels which are not division free, and test if there is a division free label Y of X such that for all other division free labels Z of X we have $Y \twoheadrightarrow Z$. If this is the case we can safely reduce X to Y.

The label in the example above would fail this criterion, and therefore be evaluated lazily. Using this strategy, we would have recomputation only for labels without a most general division free form.

From an algorithmic point of view this is still very unattractive, as we could have up to $O(n!)$ division free forms for a label and in order to show we cannot reduce the label eagerly we would have to try for each of these division free forms to reach all $n!$ other division free forms. When this computation fails, we still don't know if we are constructing a valid proof net or not and have to redo this entire computation elsewhere during the proof.

In the worst case scenario for this method we would do $O(n!)$ computations after each axiom link, and would not be able to fail early a single time.

Proving a Division Can Be Reduced Eagerly

A better method would be to perform some test on the structural postulates in advance, which would tell us for which modes the first label we find using breadth first search is also the most general.

How can we prove a division of a certain mode has this property, given the set of reductions? A first problem is how we can generate all relevant divergences in the search space.

When we can convert a label X to two different labels Y and Z, there can be three possibilities

1. The redexes for the conversions are disjoint subterms of X. In this case we can reverse the order in which we apply these conversions.

2. One of the redexes is a subterm of the other, where the subterm is a subterm of a metavariable in the conversion. In these cases we can also permute the two conversions.

3. The two redexes can overlap. Meaning that one is a subterm of the other, but not in the position of a metavariable of the first conversion. Only in this case can application of one conversion remove the second redex from the label. The pair of terms X rewrites to when such an overlap occurs is called a *critical pair* in the literature on term rewrite systems.

This gives us a limit on the number of divergences we need to consider to prove we have a most general division free form for a division of a certain mode. We can just enumerate all labels for which such an overlap exists, compute the full search space for these labels when they occur as a direct subterm of a redex for a division and test if the first division free label we will find using breadth first search is equivalent to or more general than all later division free forms in the search space. If this is not the case we don't apply eager reduction to redexes of that type.

Example 5.5 *The search space generated for example 5.1 on page 53 shows that the label* $\mathbf{x} \circ_a \mathbf{y} \circ_a \mathbf{z}$*, when it occurs as a direct subterm of the constructor* $./_n$*. has a most general division free label (of the three initial branches only one has any division free form at all).*

Showing this fragment allows eager evaluation for all modes and divisions requires generating similar search spaces for all other diverging labels, which is possible but very tedious.

Sets of structural postulates where all postulates are equivalences (applicable both as $X \to Y$ and as $Y \to X$) trivially satisfy this criterion. For grammar fragments containing only these rules we can always apply eager reduction for all modes as we know all division free forms for all labels will be equivalent.

Though computing which divisions allow eager reduction is very expensive (we will have to do a lot of $O(n!)$ computations), we only have to do this computation once for each set of structural postulates and then eager evaluation of the labels will be 'free', i.e. do only computations which a lazy evaluation strategy would have to do anyway. This will allow us to fail early on structural grounds without having to pay a heavy computational price.

5.4 Discussion

In this chapter a number of problems have been left open for further research:

In general, we may want to look at restrictions to the structural postulates to get a set of postulates with better properties like strong normalisation and confluence, possibly by using some completion strategy. This would remove the nondeterminism from the reductions, and would allow the correct normal form to be found much more efficiently.

Another line to follow would be to investigate what kind of computational effect the different packages of structural rules proposed in the literature have

when they are added to a grammar fragment. Some label algebras may have a worst case complexity much better than the $O(n!)$ guideline we used throughout this chapter, and that would make eager label reduction for these algebras more feasible.

For the goal driven reductions, when we want to apply them eagerly we need a way to automatically transform a set of structural postulates (possibly with restrictions on the set of conversions) into a set of postulates which allows eager evaluation. It is also extremely desirable, given the computational complexity, to perform the goal driven reductions incrementally, and with the current eager strategies this is impossible. For now, we have the choice of either rewriting the set of conversions 'by hand' or using the lazy reduction strategy, neither of which is particularly attractive.

For the data driven reductions, a way of transforming grammar fragments which don't allow eager reductions into fragments that do would be valuable. Currently, we have to resort to lazy reduction for non-convergent modes but at least we have a method for automatically determining for which modes this is necessary. This allows us to incrementally reduce the label we are constructing for large classes of postulates, and makes this strategy for the moment superior to the alternatives.

Chapter 6

Conclusion

Our initial goals were twofold.

First, we wanted to develop a uniform proof theory which offered an integrated account of syntax and semantics. Second, we wanted to use this proof theory as a feasible algorithm.

Of the formalisms we presented, the labeled proof net calculus fitted the multiple demands we made of a proof theory best. We showed it was sound and complete with respect to the labeled sequent calculus. Both the rules of the calculus and the treatment of syntax and semantics for the proof net calculus were uniform, while the sequent calculus was only uniform from the first point of view, and natural deduction only from the second. Finally, a simple decision procedure exists.

Though the second demand is impossible to meet in general, several practical grammar fragments have been implemented in the algorithm presented in this paper and its performance on these fragments is quite good. The combination of early failure on logical and structural grounds has as a result that we only rarely parse strings in the time indicated by the worst-case complexity. The performance of the algorithm is similar to or better than that of any other published categorial parser.

We can conclude that a combination of proof nets and labeling gives us a clean, uniform algorithm for categorial deduction which is also reasonably efficient.

Bibliography

[Abramsky 93] Abramsky, S. *Computational Interpretations of Linear Logic* Theoretical Computer Science 111, pp. 3-57, 1993.

[Abrusci 96] Abrusci, M.V. *Semantics of Proofs for Noncommutative Linear Logic* Preprint CILA, University of Bari, 1996.

[Andreoli 92] Andreoli, J.M. *Logic Programming With Focussing Proofs in Linear Logic* Journal of Logic and Computation 2(3), 1992.

[Benthem 91] Benthem, J. van *Language in Action: Categories, Lambdas and Dynamic Logic*, Studies in Logic and the Foundations of Mathematics Volume 130, North-Holland, Amsterdam, 1991.

[Benthem 96] Benthem, J. van, and A. ter Meulen (eds.) *Handbook of Logic and Language* Elsevier, to appear.

[Cervesato e.a. 96] Cervesato, I., J.S. Hodas and F. Pfenning *Efficient Resource Management for Linear Logic Proof Search* Proceedings 1996 International Workshop on Extensions of Logic Programming, Leipzig Germany, 1996

[Danos 90] Danos, V. *La Logique Linéaire Appliquée à l'Étude de Divers Processus de Normalisation at Principalement du Lambda-Calcul* Thèse de Doctorat, Université de Paris VII, 1990.

[Gabbay 94] Gabbay, D. *Labeled Deductive Systems* Report MPI-I-94-223, Max-Planck-Institut für Informatik, Saarbrücken. (To appear with Oxford University Press), 1994.

[Girard 87] Girard, J.Y. *Linear Logic* Theoretical Computer Science 50, 1987, pp. 1-102.

[Girard e.a. 89] Girard, J.Y., Y. Lafont and P. Taylor *Proof and Types* Cambridge University Press, 1989.

[Girard e.a. 95] Girard, J.Y., Y. Lafont and L. Regnier (ed.) *Advances in Linear Logic*, London Mathematical Society Lecture Notes, Cambridge University Press, 1995

[Hendriks 93] Hendriks, H. *Studied Flexibility: Categories and Types in Syntax and Semantics* Ph.D. Dissertation, University of Amsterdam, 1993.

[Hendriks & Roorda 91] Hendriks, H. and D. Roorda *Spurious Ambiguity in Categorial Grammar* BRA 3175 DYANA, 1991.

[Hodas & Miller 94] Hodas, J.S. and D. Miller *Logic Programming in a Fragment of Linear Logic* Journal of Information and Computation, 110(2), pp. 327-365, 1994.

[Janssen 95] Janssen, G.H.J. *Products and Slashes. Parsing Grammar Logics with Labeled Deductive Systems* MA Thesis, Utrecht University, 1995.

[Kaplan 88] Kaplan, S. *Rewriting With a Nondeterministic Choice Operator* Theoretical Computer Science 56, pp. 37-57, 1988.

[Klop 92] Klop J.W., *Term rewriting systems* Chapter 1 of: Handbook of Logic in Computer Science, Volume 2, S. Abramsky, D. Gabbay, T. Maibaum (Eds), Oxford University Press, pp. 1-116, 1992.

[Kurtonina 95] Kurtonina, N. *Frames and Labels. A Modal Analysis of Categorial Inference* Ph.D. Dissertation, OTS Utrecht, ILLC Amsterdam, 1995.

[Kurt. & Moortg. 95] Kurtonina, N. and M. Moortgat *Structural Control* DYANA deliverable R1.1.C, BRA 6852, Utrecht, 1995.

[Lambek 58] Lambek, J. *The Mathematics of Sentence Structure* American Mathematical Monthly 65, 1958, pp. 154-170.

[Moortgat 88] Moortgat, M. *Categorial Investigations. Logical and Linguistic Aspects of the Lambek Calculus* Foris, Dordrecht, 1988.

[Moortgat 90a] Moortgat, M. *Unambiguous proof representations for the Lambek calculus* Proceedings 7th Amsterdam Colloquium, 1990.

[Moortgat 90b] Moortgat, M. *Categorial logics: a computational perspective*, in van de Goor (ed.) Proceedings Computing Science in the Netherlands 1990. Mathematisch Centrum, Amsterdam, pp. 329-347.

[Moortgat 96] Moortgat, M. *Categorial Type Logics*, chapter 2 of [Benthem 96].

[Moortgat 96a] Moortgat, M. *Labeled Deduction in the Composition of Form and Meaning*, in [Ohlbach & Reyle 96].

[Moortgat & Oehrle 93] Moortgat, M. and R. Oehrle *Logical Parameters and Linguistic Variation. Lecture Notes on Categorial Grammar* Fifth European Summer School in Logic, Language and Information, Lisbon, 1993.

[Moortgat & Oehrle 94] Moortgat, M. and R. Oehrle *Adjacency, Dependency and Order*. Proceedings Amsterdam 9th Colloquium, 1994, pp. 447-466.

[Morrill 94] Morrill, G. *Type Logical Grammar. Categorial Logic of Signs* Kluwer, Dordrecht, 1994.

[Morrill 94a] Morrill, G. *Higher-Order Linear Logic Programming of Categorial Deduction* Report de Recerca LSI-94-42-R, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 1994.

[Morrill 95] Morrill, G. *Clausal Proofs and Discontinuity* Bulletin of the IGPL 3(2,3). Special Issue on Deduction and Language (ed. R. Kempson), 1995, pp. 403-417.

[Muskens 95] Muskens, R. *Categorial Grammar and Discourse Representation Theory*

[Ohlbach & Reyle 96] Ohlbach, H.J. and U. Reyle (eds.) *Logic, Language and Reasoning. Essays in Honor of Dov Gabbay, Part I* Kluwer, Dordrecht, to appear.

[O'Keefe 90] O'Keefe, R. A. *The Craft of Prolog* MIT Press, Cambridge, Massachusetts, 1990.

[Pentus 93] Pentus, M. *Lambek Grammars Are Context Free* Proceedings Eighth Annual IEEE Symposium on Logic in Computer Science, Montreal, 1993.

[Prawitz 71] Prawitz, D. *Ideas and Results in Proof-theory* in: Proceedings of the second Scandinavian Logic Symposium, North-Holland 1971.

[Roorda 89] Roorda, D. *Proof Nets for Lambek Calculus* Ms. ITLI Amsterdam. Unpublished PhD thesis, 1989.

[Shieber 85] Shieber, S.M. *Evidence Against the Context-Freeness of Natural Language* Linguistics and Philosophy 8, 1985, pp. 333-343.

Appendix A

Prolog Source

A.1 Representation

We represent the formulas as follows. Each case of the inductive definition has its own predicate symbol. Atomic formulas are not of the form `np` but `lit(np)`. As this is often cumbersome a macro to rewrites the former into the latter

Formulas

at	lit(at)
$\Diamond_i A$	dia(i,A)
$\Box_i^{\downarrow} A$	box(i,A)
$A \bullet_i B$	p(i,A,B)
$A /_i B$	dr(i,A,B)
$A \backslash_i B$	dl(i,A,B)

Representation of the labels is as follows, again each case of the inductive definition has its own predicate symbol. Label variables $\mathbf{x}$ are a *pair* N-X, where N is a place code and X a lexically determined variable.

Labels

x	N-X
$\langle X \rangle^i$	zip(i,X)
$\lfloor X \rfloor^i$	unzip(i,X)
$\lceil X \rceil^i$	unpack(i,X)
$X \circ_i Y$	p(i,X,Y)
$X^{\triangleleft_i}$	l(i,X)
$X^{\triangleright_i}$	r(i,X)
$X /_i Y$	dr(i,X,Y)
$X \backslash_i Y$	dl(i,X,Y)

Representation of the graph is as a list of vertices, where each vertex has a number to distinguish it from all others, a list of *pairs* of edges (the par links), and a list of atomic formulas.

A.2 Top Level

The top call to our algorithm is basically a failure driven loop, which prints all parses for a given sentence to the screen.

There are two main predicates: `test/2` expects a list of words as its first argument and looks up the formulas in the lexicon. `t/2` expects a list of formulas as its first argument and bypasses the lexical lookup. In both cases the second argument is a single succedent formula.

```
% ============================================================
% Top level
% ============================================================

test(X,Goal) :-
     test(X,Goal,Meaning,S0,S),
       write('==='),nl,
       print(X),write(' => '),print(Goal),nl,
       write('==='),nl,nl,
       write(Meaning),nl,
       nl,
       write(S0),nl,
       write(S),nl,
       fail.

test(_,_).

% = test(+ListOfWords,+GoalType,-ReducedSem,-Label,-NormalLabel)

test(X,Goal,Meaning,S0,S) :-
     init_db(X,Goal,Meaning,S0,[E|Es],[]),
     prove(Es,E),
     normalize(S0,S).

% = t(+ListOfSyn,+GoalType)

t(X,Goal) :-
    init_syn(X,Goal,Meaning,S0,[E|Es],[]),
    prove(Es,E),
    normalize(S0,S),
       write('==='),nl,
       print(X),write(' => '),print(Goal),nl,
       write('==='),nl,nl,
       write(Meaning),nl,
       nl,
       write(S0),nl,
       write(S),nl,
       fail.

t(_,_).
```

```
% = init_db(+ListOfWords,+GoalType,-GoalSem,-GoalLabel,-DataBase)
% compute initial database for given sentence and goaltype


init_db(X,G0,Meaning0,S0,[vertex(0,Bs,Ps)|As0],As) :-
     lookup(X,0,M,1,N,As0,As1),
     macro_expand(G0,G),
     link0(G,Meaning0,S0,0,M,N,_,Bs,[],Ps,[],As1,As).

init_syn(X,G0,Meaning0,S0,[vertex(0,Bs,Ps)|As0],As) :-
     link_list(X,0,M,1,N,As0,As1),
     macro_expand(G0,G),
     link0(G,Meaning0,S0,0,M,N,_,Bs,[],Ps,[],As1,As).
```

A.3 Formula Decomposition

The formula decomposition code does two things; it decomposes the formulas like described in section 3.2.2 and it constructs a graph where the only links are par links and the vertices all have a set (represented as a list) of atomic formulas assigned to them. The predicates `link0/13` (for succedent formulas) and `link1/13` (for antecedent formulas) look a bit monstrous because of the many different kinds of information involved. The first three arguments are formula, semantics, and structure label, decomposed according to the definitions on page 30 and on page 31. The next two arguments are the start and end string positions from the decomposition in section 4.2.1 on page 47. The following two are an accumulator pair, the first of which is a fresh constant represented by a number which is increased after it is used. The final six are three difference list pairs, one for the number of atomic types at the current vertex, one for the par links at the current vertex and the final one for the entire graph, represented as a list of vertices.

```
% ============================================================
% Formula decomposition
% ============================================================

link_list([],M,M,N,N,Es,Es).
link_list([X|Xs],M0,M,N0,N,[vertex(N0,As,Ps)|Es0],Es) :-
     macro_expand(X,Y),
     M1 is M0+1,
     N1 is N0+1,
     link1(Y,_,M0-'$VAR'(M0),M0,M1,N1,N2,As,[],Ps,[],Es0,Es1),
     link_list(Xs,M1,M,N2,N,Es1,Es).


% = antecedent formulas

link1(lit(A),V,S,Pos0,Pos1,N,N,
      [neg(A,V,S,Pos0,Pos1)|As],As,Ps,Ps,Es,Es).
```

```
link1(dia(J,X),V,R,Pos0,Pos1,N0,N,
      As0,As,Ps0,Ps,Es0,Es) :-
     (continuous_dia(J) ->
      Pos2=Pos0,Pos3=Pos1;true),
     link1(X,dedia(V),unzip(J,R),Pos2,Pos3,N0,N,
           As0,As,Ps0,Ps,Es0,Es).

link1(box(J,X),V,R,Pos0,Pos1,N0,N,
      As0,As,Ps0,Ps,Es0,Es) :-
    (continuous_dia(J) ->
     Pos2=Pos0,Pos3=Pos1;true),
     link1(X,debox(V),zip(J,R),Pos2,Pos3,N0,N,
           As0,As,Ps0,Ps,Es0,Es).

link1(dr(J,X,Y),U,R,Pos0,Pos1,N0,N,
      As0,As,Ps0,Ps,Es0,Es) :-
    (continuous(J) ->
     Pos2=Pos0,Pos3=Pos1,Pos4=Pos5;true),
     link0(Y,V,S,Pos3,Pos4,N0,N1,
           As0,As1,Ps0,Ps1,Es0,Es1),
     link1(X,appl(U,V),p(J,R,S),Pos2,Pos5,N1,N,
           As1,As,Ps1,Ps,Es1,Es).

link1(dl(J,Y,X),U,R,Pos0,Pos1,N0,N,
      As0,As,Ps0,Ps,Es0,Es) :-
    (continuous(J) ->
     Pos2=Pos0,Pos3=Pos1,Pos4=Pos5;true),
     link0(Y,V,S,Pos5,Pos2,N0,N1,
           As0,As1,Ps0,Ps1,Es0,Es1),
     link1(X,appl(U,V),p(J,S,R),Pos4,Pos3,N1,N,
           As1,As,Ps1,Ps,Es1,Es).

link1(p(J,X,Y),V,R,Pos0,Pos1,N0,N,
      As,As,[N0-N2|Ps],Ps,
       [vertex(N0,Bs,Qs),vertex(N2,Cs,Rs)|Es0],Es) :-
    (continuous(J) ->
     Pos2=Pos0,Pos3=Pos1,Pos4=c(N0),Pos5=c(N0);true),
     N1 is N0+1,
     link1(X,fst(V),l(J,R),Pos2,Pos4,N1,N2,
           Bs,[],Qs,[],Es0,Es1),
     N3 is N2+1,
     link1(Y,snd(V),r(J,R),Pos5,Pos3,N3,N,
           Cs,[],Rs,[],Es1,Es).

% = succedent formulas

link0(lit(A),V,S,Pos0,Pos1,N,N,
      [pos(A,V,S,Pos0,Pos1)|As],As,Ps,Ps,Es,Es).
```

```
link0(dia(J,X),condia(V),zip(J,R),Pos0,Pos1,N0,N,
      As0,As,Ps0,Ps,Es0,Es) :-
    (continuous_dia(J) ->
     Pos2=Pos0,Pos3=Pos1;true),
     link0(X,V,R,Pos2,Pos3,N0,N,
           As0,As,Ps0,Ps,Es0,Es).


link0(box(J,X),conbox(V),unpack(J,R),Pos0,Pos1,
      N0,N,As0,As,Ps0,Ps,Es0,Es) :-
    (continuous_dia(J) ->
     Pos2=Pos0,Pos3=Pos1;true),
     link0(X,V,R,Pos2,Pos3,N0,N,
           As0,As,Ps0,Ps,Es0,Es).


link0(dr(J,X,Y),lambda(U,V),dr(J,R,x-'$VAR'(N0)),Pos0,Pos1,N0,N,
      As,As,[N0-N2|Ps],Ps,
      [vertex(N0,Bs,Qs),vertex(N2,Cs,Rs)|Es0],Es) :-
    (continuous(J) ->
     Pos2=Pos0,Pos3=Pos1,Pos4=c(N0),Pos5=c(N0);true),
     N1 is N0+1,
     link1(Y,U,x-'$VAR'(N0),Pos3,Pos4,N1,N2,
           Bs,[],Qs,[],Es0,Es1),
     N3 is N2+1,
     link0(X,V,R,Pos2,Pos5,N3,N,
           Cs,[],Rs,[],Es1,Es).


link0(dl(J,Y,X),lambda(U,V),dl(J,x-'$VAR'(N0),R),Pos0,Pos1,N0,N,
      As,As,[N0-N2|Ps],Ps,
      [vertex(N0,Bs,Qs),vertex(N2,Cs,Rs)|Es0],Es) :-
    (continuous(J) ->
     Pos2=Pos0,Pos3=Pos1,Pos4=c(N0),Pos5=c(N0);true),
     N1 is N0+1,
     link0(X,V,R,Pos5,Pos3,N1,N2,
           Bs,[],Qs,[],Es0,Es1),
     N3 is N2+1,
     link1(Y,U,x-'$VAR'(N0),Pos4,Pos2,N3,N,
           Cs,[],Rs,[],Es1,Es).


link0(p(J,X,Y),pair(U,V),p(J,R,S),Pos0,Pos1,N0,N,
      As0,As,Ps0,Ps,Es0,Es) :-
    (continuous(J) ->
     Pos2=Pos0,Pos3=Pos1,Pos4=Pos5;true),
     link0(Y,V,S,Pos2,Pos4,N0,N1,
           As0,As1,Ps0,Ps1,Es0,Es1),
     link0(X,U,R,Pos5,Pos3,N1,N,
           As1,As,Ps1,Ps,Es1,Es).
```

A.4 Graph Reductions

The predicate `prove/2` is the prolog source for the algorithm on page 42 in chapter 4.

On the graph we get after the formula decomposition the first two steps of the algorithm are already performed. The predicates `select_atom/4` and `select_conj/4` select a pair of atoms, as in step 3 of the algorithm, and the `merge_vertices/4` predicate reduces the axiom link, provided the call to `cyclic/4` fails.

The predicate `reduce_graph/2` performs step 4 of the algorithm. It is a repeat loop which keeps on reducing the graph until no 2 redexes remain, and then tests for connectedness.

We end with a recursive call to `prove/2` with the new graph.

```
% ============================================================
% Graph Reductions
% ============================================================

prove([],vertex(_,[],[])).

prove([G0|Gs0],G) :-
       select_atom(neg(A,V,S0,P0,P1),vertex(N,As,Ps),
                   [G,G0|Gs0],Gs1),
       select_conj(neg(A,V,S,P0,P1),vertex(M,Bs,Qs),
                   Gs1,Gs2),
       \+ cyclic(vertex(N,As,Ps),vertex(M,Bs,Qs),Gs2,_),
       merge_vertices(vertex(N,As,Ps),vertex(M,Bs,Qs),Gs2,Gs3),
       reduce_graph(Gs3,[G4|Gs4]),
       remove_divisions(S0,S),
       prove(Gs4,G4).


select_atom(neg(B,V,S,P0,P1),vertex(N,As,Ps),G0,G) :-
       select(vertex(N,[A|As0],Ps),G0,G),
       select(neg(B,V,S,P0,P1),[A|As0],As),
       ground(S),
       !.


select_conj(neg(B,V,S,P0,P1),vertex(N,As,Ps),G0,G) :-
       select(vertex(N,[A|As0],Ps),G0,G),
       select(pos(B,V,S,P0,P1),[A|As0],As).


reduce_graph(G0,G) :-
        select(vertex(N,As,Ps0),G0,G1),
        select(M-M,Ps0,Ps),               % 2-redex found
        !,
        select(vertex(M,Bs,Qs),G1,G2),    % reduce it
        merge_vertices(vertex(N,As,Ps),vertex(M,Bs,Qs),G2,G3),
        reduce_graph(G3,G).
```

```
reduce_graph(G,G) :-
        /* no 2-redexes in graph */
        connected(G).


replace_edges([],_,_,[]).
replace_edges([vertex(N,As,Ps)|Ls],X,Y,[vertex(N,As,Qs)|Ms]) :-
       replace_edges1(Ps,X,Y,Qs),
       replace_edges(Ls,X,Y,Ms).


replace_edges1([],_,_,[]).
replace_edges1([P1-P2|Ps],X,Y,[Q1-Q2|Qs]) :-
        replace_vertex(P1,X,Y,Q1),
        replace_vertex(P2,X,Y,Q2),
        replace_edges1(Ps,X,Y,Qs).


replace_vertex(V,X,Y,W) :-
     ( V=X ->
       W=Y
     ;
       W=V).


connected([_]) :- !.


connected(L) :-
       connected1(L).


connected1([vertex(_,As,Ps)|R]) :-
      (As = [] ->
       Ps = [_|_],connected1(R)
      ;
       connected1(R)
      ).
connected1([]).


cyclic(E1,E2,G0,G) :-
     (
      ancestor(E1,E2,G0,G)
     ;
      ancestor(E2,E1,G0,G)
     ).
```

```
ancestor(vertex(N,_,[P1-P2|Ps]),vertex(M,_,Rs),G0,G) :-
      (
        P1=M
      ;
        P2=M
      ;
        select(vertex(P1,_,Qs),G0,G1),
        ancestor(vertex(P1,_,Qs),vertex(M,_,Rs),G1,G)
      ;
        select(vertex(P2,_,Qs),G0,G1),
        ancestor(vertex(P2,_,Qs),vertex(M,_,Rs),G1,G)
      ;
        ancestor(vertex(N,_,Ps),vertex(M,_,Rs),G0,G)
       ).


merge_vertices(vertex(N,As,Ps),vertex(M,Bs,Qs),
               G0,[vertex(N,Cs,Rs)|G]) :-
      append(As,Bs,Cs),
      append(Ps,Qs,Rs),
      replace_edges(G0,M,N,G).
```

A.5 Label Reductions

This code belongs to chapter 5. The divisions are removed after each axiom link, while reduction of the label to a normal form is only performed after the proof structure is complete.

The call to `select_divisions/4` removes all divisions from the label at once. The divisions are then removed with an innermost reduction strategy (meaning we will have no sublabels which are themselves divisions).

The breadth first search used is after [O'Keefe 90, pp.54-56]. I modified it to use a balanced binary tree (instead of ordered set) representation for the closed set, as the speed of this predicate is crucial.

The predicate `postulate(Label1,Label2,Name)` is defined by the user.

```
% ============================================================
% Label reductions
% ============================================================

% remove_divisions(+Label,-DivFreeLabel)
% remove all dr, dl occurences from label

% first select_divisions/4 is called, which returns a (possibly
% non-ground) DivFreeLabel and a difference-list containing
% Division-Hole pairs

remove_divisions(S0,S) :-
     select_divisions(S0,S1,L,[]),
     remove_divisions1(L,S1,S).
```

```
% remove_divisions(+ListOfDHPairs,?LabelWithHole,-GroundLabel)
% successively remove the divisions from the list of pairs and put
% them back in the corresponding holes

remove_divisions1([],S0,S) :-
     breadth_star([],1,[S0|B],B,
                  node(S0,0,empty,empty),S). % check lp even if there
                                             % are no divisions

remove_divisions1([D-H|Rest],S0,S) :-
     breadth_star([],1,[D|B],B,node(D,0,empty,empty),H),
     remove_divisions1(Rest,S0,S).

% search is breadth first with closed set, and succeeds only once
%
% solution(Label) = check_lp(Label)
% child(Parent,Child) = rewrite(Parent,Child)

% = breadth_star(+NewChildren,+QLength,+QFront,+QBack,+Closed,?Answer)

breadth_star([],N0,[Node|F],B,Closed,Answer) :-
     N0 > 0,
     N is N0-1,
     (
       check_lp(Node)
     ->
       Answer = Node
     ;
       children(Node,Children),
       union(Children,Closed,Closed1,Children1,[]),
       breadth_star(Children1,N,F,B,Closed1,Answer)
     ).

breadth_star([X|Xs],N0,F,[X|B],Closed,Answer) :-
     N is N0+1,
     breadth_star(Xs,N,F,B,Closed,Answer).

% = children/2 will generate a set of children from a given parent.

children(ParentNode,ChildrenNodes) :-
     findall(ChildNode,rewrite(ParentNode,ChildNode),ChildrenNodes).

% = normalize(+Label,-NormalLabel)
% perform label reductions and rewrite according to
% structural postulates until a normal label results.

% search is breadth first with closed set, and succeeds only once.
% solution(Label) = normal(Label)
% child(Parent,Child) = rewrite(Parent,Child)

normalize(Start0,Answer) :-
     remove_divisions(Start0,Start),
     breadth_star1([],1,[Start|B],B,node(Start,0,empty,empty),Answer).
```

```
breadth_star1([],N0,[Node|F],B,Closed,Answer) :-
     N0 > 0,
     N is N0-1,
     ( normal(Node)
     ->
       Answer = Node
     ;
       children(Node,Children),
       union(Children,Closed,Closed1,Children1,[]),
       breadth_star1(Children1,N,F,B,Closed1,Answer)
     ).

breadth_star1([X|Xs],N0,F,[X|B],Closed,Answer) :-
     N is N0+1,
     breadth_star1(Xs,N,F,B,Closed,Answer).

% rewrite(+Label0,?Label)
% true if Label0 is obtainable from Label in a single residuation
% reduction or postulate rewrite

rewrite(D,D1) :-
     reduce(D,D1).
rewrite(D,D1) :-
     postulate(D,D1,_Name).
rewrite(unpack(J,D),unpack(J,D1)) :-
     rewrite(D,D1).
rewrite(unzip(J,D),unzip(J,D1)) :-
     rewrite(D,D1).
rewrite(zip(J,D),zip(J,D1)) :-
     rewrite(D,D1).
rewrite(l(J,D),l(J,D1)) :-
     rewrite(D,D1).
rewrite(r(J,D),r(J,D1)) :-
     rewrite(D,D1).
rewrite(p(J,D,G),p(J,D1,G)) :-
     rewrite(D,D1).
rewrite(p(J,G,D),p(J,G,D1)) :-
     rewrite(D,D1).
rewrite(dr(J,D,G),dr(J,D1,G)) :-
     rewrite(D,D1).
rewrite(dl(J,G,D),dl(J,G,D1)) :-
     rewrite(D,D1).

% = Residuation reductions

reduce(p(J,l(J,D),r(J,D)),D). % product
reduce(dr(J,p(J,G,D),D),G).   % division right
reduce(dl(J,D,p(J,D,G)),G).   % division left
reduce(zip(J,unzip(J,D)),D).  % diamond
reduce(unpack(J,zip(J,D)),D). % box
```

```
% = Linear precedence
% we evaluate the word order of the label eagerly for modes
% which are declared transparent.

check_lp(Node) :-
     check_lp(Node,x,_).

% check_lp(+Node,+GreaterThan,-Max)

check_lp(N-_,X,Y) :-
      N = x
    ->
      X = Y
    ;
      precedes(X,N),
      N = Y.

check_lp(p(I,A,B),N0,N) :-
      transparent(I)   -> check_lp(A,N0,N1), check_lp(B,N1,N)
    ; check_lp(A,x,_),check_lp(B,x,_).

check_lp(zip(I,A),N0,N) :-
      transparent_dia(I) -> check_lp(A,N0,N)
    ; check_lp(A,x,_).

check_lp(unzip(_,A),N0,N) :-
     check_lp(A,N0,N).

check_lp(unpack(I,A),N0,N) :-
     lazy_unpack(I),
     check_lp(A,N0,N).

check_lp(dl(I,_,A),N0,N) :-
     lazy_dl(I),
     check_lp(A,N0,N).

check_lp(dr(I,A,_),N0,N) :-
     lazy_dr(I),
     check_lp(A,N0,N).

check_lp(l(_,A),N0,N) :-
     check_lp(A,N0,N).

check_lp(r(_,A),N0,N) :-
     check_lp(A,N0,N).
```

A.6 Auxiliaries

Some auxiliary and library predicates

```
% ============================================================
% Auxiliaries
% ============================================================

% = normal(+Label)
% a label is normal if all occurences of unzip, unpack,
% dl, dr, l and r have been reduced, the words are in the
% right order, and it contains no grammar-internal modes.

normal(Label) :-
     normal(Label,-1,_).

normal(N-_,N0,N) :-
     N is N0+1.

normal(p(I,A,B),N0,N) :-
     external(I),
     normal(A,N0,N1),
     normal(B,N1,N).

normal(zip(I,A),N0,N) :-
     external_dia(I),
     normal(A,N0,N).

% = select_divisions(+Label,-LabelWithHole,-DList)
% when called with a Label select divisions will return a copy of that
% label with a hole on the place of the divisions in it, and a
% difference list containing Division-Hole pairs. The pairs are
% ordered in such a way that a Division is ground when all Holes
% before it one the list are filled


select_divisions(N-W,N-W) -->
     [].

select_divisions(zip(I,A0),zip(I,A)) -->
     select_divisions(A0,A).

select_divisions(unzip(I,A0),unzip(I,A)) -->
     select_divisions(A0,A).

select_divisions(unpack(I,A0),H) -->
     select_divisions(A0,A),
     [unpack(I,A)-H].

select_divisions(dr(I,A0,V),H) -->
     select_divisions(A0,A),
     [dr(I,A,V)-H].

select_divisions(dl(I,V,A0),H) -->
     select_divisions(A0,A),
     [dl(I,V,A)-H].
```

```
select_divisions(p(I,A0,B0),p(I,A,B)) -->
     select_divisions(A0,A),
     select_divisions(B0,B).

select_divisions(l(I,A0),l(I,A)) -->
     select_divisions(A0,A).

select_divisions(r(I,A0),r(I,A)) -->
     select_divisions(A0,A).

% = literal(+Syn)
% true if Syn abbreviates lit(Syn), i.e. is a basic
% syntactic category

literal(X) :-
     atom(X).

% = time(+Call)
% print time used to find the first solution (if any) to Call.
% The statistics/2 predicate is Quintus-specific, and should
% be replaced by an appropriate other predicate for other
% Prologs (there is no real standard predicate for this).

time(Call) :-
     statistics(runtime,[T0|_]),
     call1(Call),
     statistics(runtime,[T|_]),
     Time is (T-T0)*0.001,
     write('CPU Time used: '),write(Time),nl.

% = call1(Goal)
% call goal once, succeed always

call1(Call) :- call(Call),!.
call1(_).

catch_fail(Goal,Message,Indicator) :-
     \+ Goal ->
     write(Message),write(Indicator),nl,fail
    ;
     Goal.


% precedes(+LPNumber0,+LPNumber)
% true if LPNumber0 precedes LPNumber. LPNumbers can be either a
% number or the constant 'x' which stands for an unknown position

precedes(x,_) :- !.
precedes(X,Y) :- X @=< Y. % also includes precedes(Num,x)
```

```
select(X,[X|Xs],Xs).
select(X,[Y|Ys],[Y|Zs]) :-
       select(X,Ys,Zs).

% = union(+List,+BalancedBinaryTree,-NewTree,-NewElementsDl)
% this predicate adds the elements of List to the
% BalancedBinaryTree and returns the new elements in a
% difference list

union([],AVL,AVL,RN,RN).

union([X|Xs],AVL0,AVL,RN0,RN) :-
       insert(AVL0,X,AVL1,RN0,RN1,_),
       union(Xs,AVL1,AVL,RN1,RN).

insert(empty,         Key, node(Key,0,empty,empty),
       [Key|RN], RN, 1).

insert(node(K,B,L,R), Key, Result, RN0, RN, Delta) :-
      compare(O, Key, K),
      insert(O, Key, Result, Delta, K, B, L, R, RN0, RN).

insert(=, Key, node(Key,B,L,R), 0, _, B, L, R, RN, RN).

insert(<, Key, Result,          Delta, K, B, L, R, RN0, RN) :-
      insert(L, Key, Lavl, RN0, RN, Ldel),
      Delta is \(B) /\ Ldel,      % this grew iff left grew
                                  % and was balanced
      B1 is B-Ldel,
      (   B1 =:= -2 ->      % rotation needed
          Lavl = node(Y,OY,A,CD),
          (   OY =< 0 ->
            NY is OY+1, NK is -NY,
            Result = node(Y,NY,A,node(K,NK,CD,R))
          ;/* OY = 1, double rotation needed */
            CD = node(X,OX,C,D),
            NY is 0-((1+OX) >> 1),
            NK is (1-OX) >> 1,
            Result = node(X,0,node(Y,NY,A,C),node(K,NK,D,R))
          )
      ;   Result = node(K,B1,Lavl,R)
      ).

insert(>, Key, Result,          Delta, K, B, L, R, RN0, RN) :-
      insert(R, Key, Ravl, RN0, RN, Rdel),
      Delta is \(B) /\ Rdel,      % this grew iff right grew
                                  % and was balanced
      B1 is B+Rdel,
      (   B1 =:= 2 ->            % rotation needed
          Ravl = node(Y,OY,AC,D),
          (   OY >= 0 ->
```

```
            NY is OY-1, NK is -NY,
            Result = node(Y,NY,node(K,NK,L,AC),D)
          ;/* OY = -1, double rotation needed */
            AC = node(X,OX,A,C),
            NY is (1-OX) >> 1,
            NK is 0-((1+OX) >> 1),
            Result = node(X,0,node(K,NK,L,A),node(Y,NY,C,D))
          )
      ;   Result = node(K,B1,L,Ravl)
      ).
```

A.7 Lexicon

This part of the code looks up the definition of a word in the lexicon. A simple macro facility is also provide by means of `macro/2` declarations.

```
% ============================================================
% Lexicon
% ============================================================

% expand macro definitions

macro_expand(S0,S) :-
     apply_macro(S0,S1),
     !,
     macro_expand(S1,S).

macro_expand(S,S).

% reduce macro definitions
macro_reduce(S0,S) :-
     apply_macro(S1,S0),
     !,
     macro_reduce(S1,S).

macro_reduce(S,S).

apply_macro(dia(I,S0),dia(I,S)) :-
     apply_macro(S0,S).
apply_macro(box(I,S0),box(I,S)) :-
     apply_macro(S0,S).
apply_macro(p(I,R0,S),p(I,R,S)) :-
     apply_macro(R0,R).
apply_macro(p(I,R,S0),p(I,R,S)) :-
     apply_macro(S0,S).
apply_macro(dl(I,R0,S),dl(I,R,S)) :-
     apply_macro(R0,R).
apply_macro(dl(I,R,S0),dl(I,R,S)) :-
     apply_macro(S0,S).
apply_macro(dr(I,R0,S),dr(I,R,S)) :-
```

```
     apply_macro(R0,R).
apply_macro(dr(I,R,S0),dr(I,R,S)) :-
     apply_macro(S0,S).
apply_macro(S0,S) :-
     macro(S0,S).

% = lexical lookup

lookup([],M,M,N,N,L,L).

lookup([W|Ws],M0,M,N0,N,[vertex(N0,As,Ps)|L0],L) :-
     catch_fail(lex(W,Syn0,Sem),'Lexical lookup failed: ',W),
     M1 is M0+1,
     N1 is N0+1,
     macro_expand(Syn0,Syn),
     link1(Syn,Sem,M0-W,M0,M1,N1,N2,As,[],Ps,[],L0,L1),
     lookup(Ws,M1,M,N2,N,L1,L).
```

Appendix B

Designing Grammar Fragments

This appendix gives an illustration on how to design grammar fragments for the program of the previous appendix.

A grammar fragment must at least contain the following:

Structural Postulates A set of `postulate/3` predicates. We will translate conversions of the form $X \rightarrow_{\text{Name}} Y$ into `postulate(X,Y,Name)` declarations, where `Name` is `print`able.

Lexicon A set of `lex/3` predicates. A lexical entry for a word has a syntactic formula and semantic term associated with it. This will give declarations of the form `lex(Word,Formula,Term)`.

Macros A set of `macro/2` predicates, for some handy abbreviations.

Please note that these predicates are *compiled* Prolog code, so to prevent exceptions when one of these components is empty a 'dummy' predicate should be used, for example

```
postulate('$LABEL1','$LABEL2','$NAME')
```

In addition, the following declarations are supported

Lazy Declarations Are necessary for modes which don't converge on division free forms (see section 5.3.1). If you are unsure about this property, declare lazy evaluation for all modes as follows

```
lazy_unpack(_).
lazy_dl(_).
lazy_dr(_).
```

Transparency Declarations Can be given for all modes which have intuitively the following property: when placed in a larger context, the word order possibilities of resources composed in this way do not increase. If you are unsure about this property, don't declare any modes as transparent.

Continuity Declarations Can be given for all modes which are continuous (see definition 4.1). If you are unsure about this property don't declare any modes as continuous.

External Declarations Are useful when you want to prevent certain modes from occurring in the output. Modes not declared as external are for grammar internal or auxiliary purposes only. By default all modes should be external.

```
external(_).
external_dia(_).
```

A small package `analysis.pl` is provided to assist the user with making these declarations.

B.1 Example: Dutch Verb Raising

In this section, I will present a small grammar fragment for verb raising in Dutch, which is essentially the same as the one used in [Moortgat 96a]. We have three binary modes; a for regular phrasal composition, h for the verb cluster and w for 'head-wrapping'. We also have the unary modes; lh marks a lexical head, h marks a phrasal head, and w marks a head-wrapping element.

The fragment has six structural postulates. The first is an inclusion postulate stating that we can use a lexical head lh as a phrasal head h. The two distribution principles 'project' a phrasal head upward through the regular phrasal composition a. The fourth postulate is the crux of the fragment. When a wrapper (an auxiliary verb) and a lexical head (a main verb, which has not yet found any of its arguments) are composed with the binary head-wrapping mode, we can project the lexical head up and replace the head-wrapping mode w by the verb cluster mode h. Finally the mixed associativity and commutativity postulates make sure the auxiliary verbs can 'move' to the right position.

This gives us intuitively the following account of verb raising. The main verb is lexically marked as the head (by an assignment of e.g. $\Box^{\downarrow}_{lh}(np\backslash_a inf)$ for 'plagen'), and the auxiliary verbs as head-wrappers (by an assignment of e.g. $\Box^{\downarrow}_{w}(iv/_w inf)$ for 'vinden'). The mixed associativity and commutativity postulates move the auxiliary verbs to the right position, where the verb raising postulate can be applied. Only then do we apply the inclusion postulate to use the lexical head as a phrasal head, and distribute the h upward to derive the $\Box^{\downarrow}_{h}s$ goal formula.

```
% ============================================================
% Postulates
% ============================================================

postulate(zip(lh,A),zip(h,A),'LH').
postulate(p(a,zip(h,A),B),zip(h,p(a,A,B)),'K1').
postulate(p(a,A,zip(h,B)),zip(h,p(a,A,B)),'K2').
postulate(p(w,zip(w,A),zip(lh,B)),zip(lh,p(h,A,B)),'VR').
postulate(p(w,B,p(a,A,C)),p(a,A,p(w,B,C)),'MxCom').
postulate(p(w,A,p(a,B,C)),p(a,p(w,A,B),C),'MxAss').
```

```
% ============================================================
% Lexicon
% ============================================================

% = lex(Label,Formula,Term).

lex(fred,np,fred).
lex(mary,np,mary).
lex(gek,ap,gek).
lex(zal,box(w,dr(w,iv,inf)),zal).
lex(moeten,box(w,dr(w,inf,inf)),moeten).
lex(plagen,box(lh,dl(a,np,inf)),plagen).
lex(vinden,box(lh,dl(a,ap,dl(a,np,inf))),vinden).

% ============================================================
% Macros
% ============================================================

macro(iv,dl(a,np,s)).  % verb phrase
macro(X,lit(X)) :-
     literal(X).
```

Declarations To ensure the binary w mode *must* be removed by the verb raising postulate, only the a and h mode are declared as external. We will allow none of the unary modes in the output for similar reasons. This will give the following external declarations.

```
external(h).
external(a).
```

Only the binary h mode is trivially continuous, because no postulates are applicable to it. The mixed commutativity postulate makes both the a and w mode discontinuous. We give the following declaration

```
continuous(h).
```

Only the phrasal head mode h needs lazy evaluation, as it has both $[K1]$ and $[K2]$ postulates defined for it.

```
lazy_unpack(h).
```

Finally, we can declare all modes as transparent with the following declarations.

```
transparent(_).
transparent_dia(_).
```

Alternatively we leave these declarations to `analysis.pl`.

B.2 Session Transcript

```
| ?- [labels,analysis].
% compiling file /users.ruulot/moot/prolog/mm/labels.pl
%  compiling file /users.ruulot/moot/prolog/mm/reduce_sem.pl
%  reduce_sem.pl compiled in module user, 0.300 sec 3,028 bytes
%  compiling file /users.ruulot/moot/prolog/mm/pp.pl
%  pp.pl compiled in module user, 0.500 sec 6,972 bytes
% labels.pl compiled in module user, 2.516 sec 30,596 bytes
% compiling file /users.ruulot/moot/prolog/mm/analysis.pl
%  loading file /usr/local/quintus/generic/qplib3.2/library/unify.qof
%   loading file /usr/local/quintus/generic/qplib3.2/library/occurs.qof
%   occurs.qof loaded in module unify, 0.017 sec 3,024 bytes
%  unify.qof loaded, 0.067 sec 5,284 bytes
%  module unify imported into user
% analysis.pl compiled in module user, 1.000 sec 16,384 bytes

yes
| ?- load_fragment(vr).
0. === Compiling File
% compiling file /users.ruulot/moot/prolog/mm/vr.pl
% vr.pl compiled in module user, 0.200 sec 3,612 bytes
1. === Retracting Mode Declarations
2. === Modes Found:
     = unary:  [h,lh,w]
     = binary: [a,h,w]
3. === Testing Convergence:
     = unary:
        h:diverges:
          --(^w(A) *w (^lh(B) *a C)) ->LH
            (^w(A) *w (^h(B) *a C)) ->MxCom
            (^h(B) *a (^w(A) *w C)) ->K1
            ^h((B *a (^w(A) *w C)))
          --(^w(A) *w (^lh(B) *a C)) ->MxAss
            ((^w(A) *w ^lh(B)) *a C) ->VR
            (^lh((A *h B)) *a C) ->LH
            (^h((A *h B)) *a C) ->K1
            ^h(((A *h B) *a C))
        lh:converges
        w:converges
     = binary:
        a:converges
        h:converges
        w:converges
4. === Testing Transparency:
     = unary:
        h:transparent
        lh:transparent
        w:transparent
     = binary:
```

```
        a:transparent
        h:transparent
        w:transparent

yes
| ?- test([fred,mary,gek,zal,moeten,vinden],box(h,s)).
===
[fred,mary,gek,zal,moeten,vinden] => @h(s)
===

zal(fred,moeten(vinden(mary,gek)))

@h((fred *a (mary *a (gek *a (^w(zal) *w (^w(moeten) *w ^lh(vinden)))))))) -->
(fred *a (mary *a (gek *a (zal *h (moeten *h vinden)))))

1 solution found.

== statistics ==

CPU Time used : 0.083
Total links   : 16
ACC links     : 14 (2)
label links   : 13 (1)

yes
| ?- test([fred,mary,zal,gek,moeten,vinden],box(h,s)). % non-derivable

No solutions!

== statistics ==

CPU Time used : 0.067
Total links   : 16
ACC links     : 14 (2)
label links   : 13 (1)
```

B.3 Sequent and Natural Deduction Output

Packages `nd.pl` and `seq.pl` are provided to give more readable natural deduction and sequent output. LaTeX capacity and paper size make these packages only useful for shorter sentences.

$$\frac{vinden \vdash \Box_{lh}(ap\backslash_a(np\backslash_a inf))}{\langle vinden\rangle^{lh} \vdash ap\backslash_a(np\backslash_a inf)}\,[\Box E]$$

$$\frac{gek \vdash ap \qquad \langle vinden\rangle^{lh} \vdash ap\backslash_a(np\backslash_a inf)}{(gek, \langle vinden\rangle^{lh})^a \vdash np\backslash_a inf}\,[\backslash E]$$

$$\frac{mary \vdash np \qquad (gek, \langle vinden\rangle^{lh})^a \vdash np\backslash_a inf}{(mary, (gek, \langle vinden\rangle^{lh})^a)^a \vdash inf}\,[\backslash E]$$

$$\frac{zal \vdash \Box_w(iv/_w inf)}{\langle zal\rangle^w \vdash iv/_w inf}\,[\Box E]$$

$$\frac{\langle zal\rangle^w \vdash iv/_w inf \qquad (mary, (gek, \langle vinden\rangle^{lh})^a)^a \vdash inf}{(\langle zal\rangle^w, (mary, (gek, \langle vinden\rangle^{lh})^a)^a)^w \vdash iv}\,[/E]$$

$$\frac{fred \vdash np \qquad (\langle zal\rangle^w, (mary, (gek, \langle vinden\rangle^{lh})^a)^a)^w \vdash iv}{(fred, (\langle zal\rangle^w, (mary, (gek, \langle vinden\rangle^{lh})^a)^a)^w)^a \vdash s}\,[\backslash E]$$

$$\frac{(fred, (\langle zal\rangle^w, (mary, (gek, \langle vinden\rangle^{lh})^a)^a)^w)^a \vdash s}{(fred, (mary, (\langle zal\rangle^w, (gek, \langle vinden\rangle^{lh})^a)^w)^a)^a \vdash s}\,[MxCom]$$

$$\frac{(fred, (mary, (\langle zal\rangle^w, (gek, \langle vinden\rangle^{lh})^a)^w)^a)^a \vdash s}{(fred, (mary, (gek, (\langle zal\rangle^w, \langle vinden\rangle^{lh})^w)^a)^a)^a \vdash s}\,[MxCom]$$

$$\frac{(fred, (mary, (gek, (\langle zal\rangle^w, \langle vinden\rangle^{lh})^w)^a)^a)^a \vdash s}{(fred, (mary, (gek, \langle (zal, vinden)^h\rangle^{lh})^a)^a)^a \vdash s}\,[VR]$$

$$\frac{(fred, (mary, (gek, \langle (zal, vinden)^h\rangle^{lh})^a)^a)^a \vdash s}{(fred, (mary, (gek, \langle (zal, vinden)^h\rangle^{h})^a)^a)^a \vdash s}\,[LH]$$

$$\frac{(fred, (mary, (gek, \langle (zal, vinden)^h\rangle^{h})^a)^a)^a \vdash s}{(fred, (mary, \langle (gek, (zal, vinden)^h)^a\rangle^{h})^a)^a \vdash s}\,[K2]$$

$$\frac{(fred, (mary, \langle (gek, (zal, vinden)^h)^a\rangle^{h})^a)^a \vdash s}{(fred, \langle (mary, (gek, (zal, vinden)^h)^a)^a\rangle^{h})^a \vdash s}\,[K2]$$

$$\frac{(fred, \langle (mary, (gek, (zal, vinden)^h)^a)^a\rangle^{h})^a \vdash s}{\langle (fred, (mary, (gek, (zal, vinden)^h)^a)^a)^a\rangle^{h} \vdash s}\,[K2]$$

$$\frac{\langle (fred, (mary, (gek, (zal, vinden)^h)^a)^a)^a\rangle^{h} \vdash s}{(fred, (mary, (gek, (zal, vinden)^h)^a)^a)^a \vdash \Box_h s}\,[\Box I]$$

Figure B.1: Natural Deduction output for **zal**(**fred**, **vinden**(**mary**, **gek**))

$$\frac{}{np \Rightarrow np}\,[Ax] \qquad \frac{}{s \Rightarrow s}\,[Ax]$$

$$\frac{np \Rightarrow np \qquad s \Rightarrow s}{(np, iv)^a \Rightarrow s}\,[L\backslash_a] \qquad \frac{}{inf \Rightarrow inf}\,[Ax]$$

$$\frac{(np, iv)^a \Rightarrow s \qquad inf \Rightarrow inf}{(np, (iv/_w inf, inf)^w)^a \Rightarrow s}\,[L/_w]$$

$$\frac{(np, (iv/_w inf, inf)^w)^a \Rightarrow s}{(np, (\langle \Box_w(iv/_w inf)\rangle^w, inf)^w)^a \Rightarrow s}\,[L\Box_w]$$

$$\frac{\frac{}{np \Rightarrow np}\,[Ax] \qquad (np, (\langle \Box_w(iv/_w inf)\rangle^w, inf)^w)^a \Rightarrow s}{(np, (\langle \Box_w(iv/_w inf)\rangle^w, (np, np\backslash_a inf)^a)^w)^a \Rightarrow s}\,[L\backslash_a]$$

$$\frac{\frac{}{ap \Rightarrow ap}\,[Ax] \qquad (np, (\langle \Box_w(iv/_w inf)\rangle^w, (np, np\backslash_a inf)^a)^w)^a \Rightarrow s}{(np, (\langle \Box_w(iv/_w inf)\rangle^w, (np, (ap, ap\backslash_a(np\backslash_a inf))^a)^a)^w)^a \Rightarrow s}\,[L\backslash_a]$$

$$\frac{(np, (\langle \Box_w(iv/_w inf)\rangle^w, (np, (ap, ap\backslash_a(np\backslash_a inf))^a)^a)^w)^a \Rightarrow s}{(np, (\langle \Box_w(iv/_w inf)\rangle^w, (np, (ap, \langle \Box_{lh}(ap\backslash_a(np\backslash_a inf))\rangle^{lh})^a)^a)^w)^a \Rightarrow s}\,[L\Box_{lh}]$$

$$\frac{(np, (\langle \Box_w(iv/_w inf)\rangle^w, (np, (ap, \langle \Box_{lh}(ap\backslash_a(np\backslash_a inf))\rangle^{lh})^a)^a)^w)^a \Rightarrow s}{(np, (np, (\langle \Box_w(iv/_w inf)\rangle^w, (ap, \langle \Box_{lh}(ap\backslash_a(np\backslash_a inf))\rangle^{lh})^a)^w)^a)^a \Rightarrow s}\,[MxCom]$$

$$\frac{(np, (np, (\langle \Box_w(iv/_w inf)\rangle^w, (ap, \langle \Box_{lh}(ap\backslash_a(np\backslash_a inf))\rangle^{lh})^a)^w)^a)^a \Rightarrow s}{(np, (np, (ap, (\langle \Box_w(iv/_w inf)\rangle^w, \langle \Box_{lh}(ap\backslash_a(np\backslash_a inf))\rangle^{lh})^w)^a)^a)^a \Rightarrow s}\,[MxCom]$$

$$\frac{(np, (np, (ap, (\langle \Box_w(iv/_w inf)\rangle^w, \langle \Box_{lh}(ap\backslash_a(np\backslash_a inf))\rangle^{lh})^w)^a)^a)^a \Rightarrow s}{(np, (np, (ap, \langle (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h\rangle^{lh})^a)^a)^a \Rightarrow s}\,[VR]$$

$$\frac{(np, (np, (ap, \langle (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h\rangle^{lh})^a)^a)^a \Rightarrow s}{(np, (np, (ap, \langle (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h\rangle^{h})^a)^a)^a \Rightarrow s}\,[LH]$$

$$\frac{(np, (np, (ap, \langle (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h\rangle^{h})^a)^a)^a \Rightarrow s}{(np, (np, \langle (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a\rangle^{h})^a)^a \Rightarrow s}\,[K2]$$

$$\frac{(np, (np, \langle (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a\rangle^{h})^a)^a \Rightarrow s}{(np, \langle (np, (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a)^a\rangle^{h})^a \Rightarrow s}\,[K2]$$

$$\frac{(np, \langle (np, (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a)^a\rangle^{h})^a \Rightarrow s}{\langle (np, (np, (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a)^a)^a\rangle^{h} \Rightarrow s}\,[K2]$$

$$\frac{\langle (np, (np, (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a)^a)^a\rangle^{h} \Rightarrow s}{(np, (np, (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a)^a)^a \Rightarrow \Box_h s}\,[R\Box_h]$$

$$\frac{(np, (np, (ap, (\Box_w(iv/_w inf), \Box_{lh}(ap\backslash_a(np\backslash_a inf)))^h)^a)^a)^a \Rightarrow \Box_h s}{(fred, (mary, (gek, (zal, vinden)^h)^a)^a)^a \Rightarrow \Box_h s}\,[Lex]$$

Figure B.2: Sequent output for **zal**(**fred**, **vinden**(**mary**, **gek**))