Labelled Proof Nets for the Syntax and Semantics of Natural Languages

GUY PERRIER, *LORIA, Université Nancy 2, Campus Scientifique, B.P. 239, 54506 Vandœuvre-les-Nancy Cedex, France.*
E-mail: perrier@loria.fr

Abstract

We propose to represent the syntax and semantics of natural languages with labelled proof nets in the implicative fragment of intuitionistic linear logic. Resource-sensitivity of linear logic is used to express all dependencies between the syntactic constituents of a sentence in the form of a proof net. Phonological and semantic labelling of the proof net from its inputs to the unique output are used to produce the well-formed phonological form and the semantic representation of a sentence from entries of a lexicon. In this way we obtain a linguistic model of great flexibility because labelling is not completely determined logically: it is used to introduce linguistic constraints which go beyond the underlying logic.[1]

Keywords: linear logic, proof nets, Categorial Grammars

Introduction

Within the framework of Categorial Grammars, the Lambek Calculus has contributed to the unification of linguistic theory and the promotion of proof theory as a pertinent framework for analysing natural languages. Now, the non-commutativity of this logic strongly limits its expressivity: Pentus has proved that Lambek Grammars are equivalent to Context-Free Grammars [21], which are far from being sufficient to generate natural languages. Amongst all the dependencies between the syntactic components of a sentence, the only ones which are representable in the Lambek calculus are adjacent dependencies. In particular, the Lambek Calculus can represent only peripheral extraction from relative clauses and not medial extraction.

A way of going beyond these limitations is to extend the logic. The initial kernel of the Lambek Calculus is left unchanged but it is extended with new connectives which are used to express linguistic phenomena that defy the power of the pure Lambek Calculus. This approach has been fruitfully explored by Moortgat [13, 15], Morrill [16, 17], Barry, Hepple, Leslie and Morrill [2], Moortgat and Morrill[10], who have designed new connectives, such as unary modalities and binary operators for extraction and wrapping, and new logical systems such as multi-modal systems to extend the power of the Lambek Calculus.

Oehrle [20] proposes an alternative approach which consists in relaxing the logical

[1] Full version of a contributed paper presented at the *4th Workshop on Logic, Language, Information and Computation* (*WoLLIC'97*), http://www.di.ufpe.br/~wollic/wollic97, held in Fortaleza (Ceará), Brazil, August 19–22 1997, with scientific sponsorship from IGPL, FoLLI, and ASL, and organised by Univ. Federal do Ceará (UFC) and Univ. Federal de Pernambuco (UFPE).

framework drastically: instead of the Lambek Calculus, he chooses its commutative counterpart LP, that is the multiplicative fragment of Intuitionistic Linear Logic (IMLL). In doing this, he aims to represent all adjacent and long distance dependencies between syntactic constituents of a sentence in a logically uniform way and he defers the other syntactic phenomena such as word order to a non-logical level using labels. Each syntactic type is labelled with a phonological term which encodes a particular expression of the language belonging to this type. All phonological terms form a higher-order typed lambda-calculus on a free non-commutative monoid. Then, each rule of the IMLL deductive system is enriched to define how to compose the terms in parallel with the formulas they label. In this way, there is a division of the grammatical labor - as Oehrle says - between the logic system and the term system which is flexible enough to go beyond the limitations of the Lambek Calculus.

Morrill [19] arrived at a similar system but starting from another motivation: his goal was to find a common framework for implementing variants and extensions of the Lambek Calculus.

Oehrle presents his labelled deductive system in the framework of the sequent calculus but, as Girard has shown in his original paper on linear logic [4], there is a more natural framework for proofs in linear logic, that of *proof nets*. Oerhle was aware of this since, in [20], he presented a translation of his system in the form of *atomic polar decomposition* which explicitly refers to proof nets. We propose to continue in this direction by developing the approach of labelled deductive systems for grammars within the framework of proof nets. We restrict the logical system to the implicative fragment of intuitionistic Linear Logic (IILL), that is the fragment with linear implication $\multimap$ as the only connective, which is sufficient for the linguistic applications which we envisage here.

Roughly speaking, a cut free IILL proof net is a set of syntactic trees of formulas connected together at the level of their leaves by means of *axiom links*. Linguistically, the syntactic trees correspond to the grammatical components of a sentence and the axiom links express the dependencies between these components, so that the whole proof net expresses the syntactic structure of the corresponding sentence. An intuitionistic proof net is oriented from the inputs to the unique output. The inputs represent the lexical entries used to build the sentence and the output represents the sentence as a whole.

A proof net can be viewed as a medium for the circulation of information from the inputs to the output. The travelling information can be materialised by means of elements of a *free commutative monoid* labelling formula occurrences in the proof net. Labelling is initialised at the inputs and then it is propagated step by step according to a law which is similar to the law of nodes for electric currents and which preserves all of information that goes through the net. The proof net is correct iff all the information entering the inputs ends up at the output. Such labelling was introduced by Roorda [22] but he doubted it to be sufficient for ensuring the correctness of a proof net; de Groote has proved that it is sufficient [3]. We call such labelling *minimal labelling* because it guarantees the correctness of a proof net by propagation of minimal information through the proof net.

¿From a linguistic point of view, labels represent multi-sets of words which constitute the resources that are consumed in building the syntactic constituents of a sentence. Such labelling is not sufficient to guide the construction of the correct utterance of a sentence. In particular, it does not give any information about word order. In the approach that keeps the Lambek Calculus as the logical kernel, Moortgat [14], Morrill [18], Merenciano and Morrill [11] have shown that it is possible to use Lambek proof nets to represent the syntax of sentences but with the limitations of the Lambek Calculus in their expressive power. As we want to remain in a commutative logical framework, we solve the problem using the same principle as Oehrle: the logic only expresses the fact that the grammatical constituents fulfill their valencies to build a sentence and all other grammatical aspects are taken into account by labels. For this, we constrain minimal labelling by adding non-commutativity to transform it into *phonological labelling*. Then, labels represent the phonological form of grammatical constituents and they are the vehicle for expressing additional linguistic constraints, which go beyond the power of the Lambek Calculus. In particular, medial extraction from a relative clause is representable in this framework.

According to the view of Categorial Grammars, all syntactic information is extracted from a *lexicon* and well-formed sentences are built from entries of the lexicon using only deduction rules of the logic. In our approach, the lexicon comprises syntactic trees of IILL formulas. For each tree, the leaves represent the atomic categories that make up the corresponding IILL formula and they are polarised: some are inputs and the others are outputs. Each one is provided with a *label pattern* which plays an essential role as to the expressive power of the model. The patterns of the input leaves are used to filter the input labels and the patterns of the output leaves are used to determine their labels from the labels of the inputs.

Parsing a sentence, then, consists in selecting syntactic trees from the lexicon that correspond to the words of the sentence. We obtain a proof frame which we label from the inputs to the outputs. By adding axiom links, we allow labels to be propagated from the output of one tree to the input of another. The process succeeds when all labelling streams have merged on the unique output of the proof frame which is confirmed as a proof net at this moment. The efficiency of the algorithm can be improved by transforming the problem into a problem of building and solving a specific system of equations in a monoid. Morrill [18], Merenciano and Morrill [11] use a similar method for parsing and generating natural sentences with Lambek proof nets but in the opposite direction, from the output to the inputs.

Following the program of Montague, we use the IILL proof net that represents the syntactic dependencies in a sentence to compose its meaning from the meaning of the words, given by the lexicon. For this, like Oehrle, we use *semantic labelling*, which is performed in parallel with phonological labelling during the parsing of a sentence. Roorda [22] introduced this form of labelling to establish a specific criterion for ILL and Lambek proof nets but he did not see that such a labelling is redundant under this consideration with respect to minimal labelling. A semantic label is a linear lambda-term which keeps a trace of the history of the structure explored up to the occurrence of the formula where the label is attached. Semantic labelling begins by putting distinct variables on the inputs of the proof net and ends by labelling its output. The label of the output codes a function that computes the meaning of the sentence from the meaning of its words. This meaning is expressed with typed lambda-terms of higher order logic.

Finally, the laws of both phonological and semantic labelling are not sufficient for building well-formed phonological forms and semantic representations of sentences.

Additional information is necessary to control the building of the syntax and semantics of sentences. A flexible way of doing this consists in using *feature matrices* for labelling proof nets. By considering phonological and semantic labels as particular features of these structures, we obtain a unique system of labelling for proof nets. The mechanism of parsing, which is based on connecting internal inputs and outputs with axiom links, is enriched in such a way that it integrates *unification of feature matrices*. In this way, the model takes into account some characteristics of Unification Grammars [1].
Section 1 is devoted to a purely proof-theoretic study of labelled proof nets while Section 2 shows how this theoretical tool can apply to the analysis of the syntax and semantics of natural languages.

1 Labelled proof nets

In this section, we study labelled proof nets from a purely proof-theoretic viewpoint independently of their linguistic applications. As Lamarche has shown [6, 8], the correctness of IILL proof nets can be characterised in a way that is specific to the intuitionistic framework. De Groote has reformulated this criterion in terms of proof net minimal labelling [3]. Minimal labelling leads naturally to an algorithm for proof net construction which can be improved by transforming the problem into a problem of building and solving a system of equations in a monoid.

1.1 The syntax of proof nets

We start by defining the syntax of $IILL$ formulas.

Definition 1.1 *Let $\mathcal{P}$ be a set of propositions. The set of $IILL$ formulas built from $\mathcal{P}$ is defined by the grammar $\mathcal{F} ::= \mathcal{P} \mid \mathcal{F} \multimap \mathcal{F}$ with the convention that linear implication $\multimap$ is right associative.*

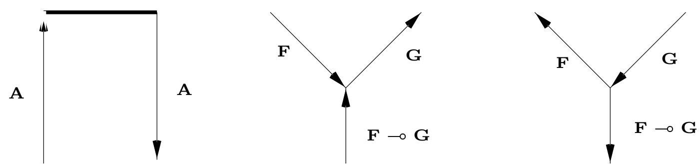
Since we need to consider proof nets in construction, we must first define the syntax of uncompleted proof nets or proof frames, extending the terminology Roorda has established for Lambek proof nets [22].

Definition 1.2 *An $IILL$ proof frame is an oriented graph such that:*

- *Each edge, called a formula occurrence, is labelled with an $IILL$ formula and a polarity + or -.* [2]
- *Each vertex, called a link, has one of the following types which are characterised by their outgoing and incoming formula occurrences:* [3]

[2] Graphically, the polarity of a formula occurrence is not represented by a sign but by a vertical orientation: negative occurrences are represented by up arrows and positive occurrences by down arrows.

[3] The capital A represents an atomic formula whereas F and G represent any formulas and, in our representation, an axiom link is a vertex and not an edge.



axiom link ⊸-input link ⊸-output link

- *A negative formula occurrence without target link is called an internal output whereas a positive formula occurrence without source link is called an internal input.*
  A negative formula occurrence without source link is called an external input whereas a positive formula occurrence without target link is called an external output. A proof frame has exactly one external output.

A proof frame represents the syntax of a proof net in construction. During the construction, internal outputs will be connected with internal inputs by axiom links. At the end of the construction, a proof frame is internally closed and becomes a *proof structure*. Its interface with the outside is constituted by the external inputs and the unique external output.

Definition 1.3 *An $IILL$ proof structure is a proof frame without internal inputs or outputs.*

Our representation of $IILL$ proof nets corresponds to the two-sided sequent calculus: in a proof structure, the inputs correspond to the formulas on the left hand side of the associated sequent and the unique output corresponds to the unique formula on the right hand side of the sequent. There is another representation of $IILL$ using the one-sided sequent calculus with polarities [7]: in this representation, the $\multimap$-input and $\multimap$-output links are respectively replaced with $\otimes$-links and $\parr$-links, up and down formulas become input and output formulas but there is no essential difference between the two representations.

1.2 Minimal labelling and correctness of proof nets

Of course, all proof structures are not proof nets. Among the variety of correctness criteria for proof nets, we have chosen that of de Groote [3] because it uses the specific properties of intuitionistic proof nets and it is well suited to linguistic applications.

1.2.1 Notion of minimal labelling

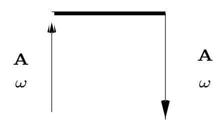
The criterion of de Groote uses minimal labelling which requires the assumption of a free commutative monoid $\mathcal{M}$ with a countably infinite number of primes. They are denoted α, β, $\gamma\ldots$, possibly with subscripts, and the commutative operation of the monoid is denoted $+$.

The step by step labelling of a proof net with elements of $\mathcal{M}$ can be viewed as a trip of information through the proof net. At each link, a principle of conservation of information is applied which is similar to the law of nodes for electric currents in a network. The structure is correct as a net iff the quantity of information that has

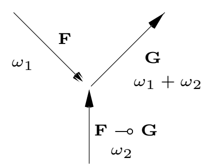
entered the external inputs goes out unchanged from the external output. At first, the information input stream is scattered; subsequently it is merged inside the proof net in such a way that it is transformed into a unique output stream. We can consider a dual trip of information in a proof net from the output to the inputs [6]. On this view, the unique output information stream is split into several input streams. This second view is more difficult to implement by means of labels. In particular, splitting the output stream is non-deterministic unlike the converse process.
To define this criterion precisely, we need to extend the notion of input for a proof structure. The external inputs of a proof structure correspond to hypotheses but we must also take into account *virtual inputs.* In a proof frame, virtual inputs are the up premises of $\multimap$-output links and they correspond to discharged hypotheses in natural deduction proofs, as we will see later.
Given all inputs, minimal labelling is defined step by step according to the following algorithm.

Definition 1.4 *All actual and virtual inputs of the proof structure are labelled with distinct primes of $\mathcal{M}$. Then, for each link that has its incoming edges labelled and one outgoing edge not labelled, this edge is labelled according to one of the following rules:*

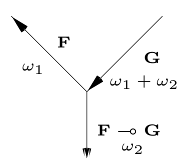
rule 1: *if the link is an axiom link, the label of the incoming edge is transmitted to the outgoing edge;*



rule 2: *if the link is an $\multimap$-input link, the negative premise is labelled with the sum of the labels of the positive premise and the conclusion;*

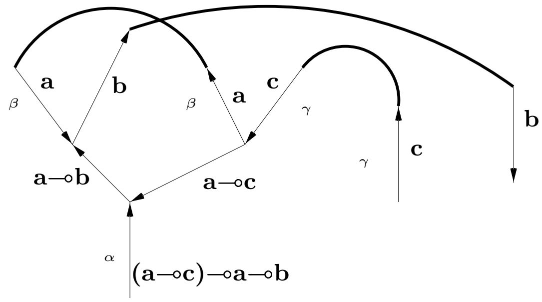


rule 3: *if the link is an $\multimap$-output link, the conclusion is labelled with the difference between the labels of the premises. If the label of the negative premise does not appear in the label of the positive premise, the difference is not defined and labelling the conclusion fails.*

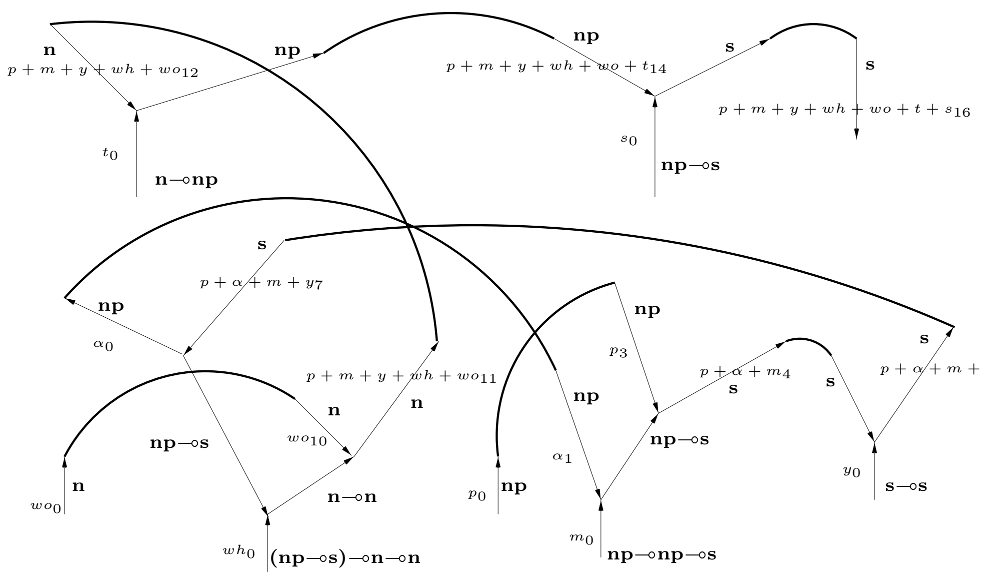


Of course, such an algorithm terminates and the values of the labels that are put on the inputs do not matter provided that they are distinct; they are only used to identify the inputs of the proof structure. From this, we can identify all minimal labellings of a given proof structure: there is only one (up to alternative choices of primes).

Example 1.5 *The proof structure below corresponds to the non-theorem* $(a \multimap c) \multimap (a \multimap b),\ c\ \vdash\ b$. *Minimal labelling is not total because the label* γ *coming into the* $\multimap$*-output link cannot be propagated, which expresses that this link connects two independent parts of the proof structure and entails that this is not a proof net.*



Example 1.6 *The syntactic structure of the sentence* the woman whom Peter met yesterday smokes *is represented by the labelled proof structure below. The monoid* $\mathcal{M}$ *used to build labels is composed of bags of words taken among the words of the sentence*[4] *completed with a label* α *for the unique virtual input of the proof structure. Then we apply the labelling algorithm step by step. To simplify the representation, the words of the sentence are replaced with their initials and some labels are omitted. Moreover, the rank of each step in the process of labelling is added as a subscript of the corresponding new label.*



If we interpret the labelled proof structure from a linguistic point of view, we observe that it only expresses the management of resources in the composition of the syntax

[4] If the same word occurs twice in the sentence, each of its occurrences is represented by a distinct prime.

of the sentence. Other linguistic phenomena such as word order are not captured by this labelling, which exactly corresponds to the expressive power of commutative linear logic. We will see later how to enrich the structure with linguistic information.

1.2.2 Correctness criterion

In the process of minimal labelling, we recover natural deduction in a certain sense. As IILL only contains one connective, linear implication, its natural deduction system is very simple; it is constituted of two rules: the *abstraction rule* which introduces linear implication and the *application rule* which eliminates linear implication. In a proof structure, the steps of labelling that lead to the label of a particular formula occurrence F constitute a tree which can be viewed as the proof of F in a natural deduction setting:

- crossing an $\multimap$-input link represents an application which is represented with an addition at the level of labels;
- crossing an $\multimap$-output link represents an abstraction, for which hypothesis cancellation is performed by a subtraction of labels; [5]
- crossing an axiom link represents nothing because there is no distinction between up and down formulas in natural deduction.

This tree will be called the *natural deduction tree of* F and denoted D_F. It exactly corresponds to the notion of *empire* defined by Lamarche [8]. The leaves of D_F represent all hypotheses that have been used to prove F but only those that have not been cancelled in this proof appear in the label of F.
Overall, the process of minimal labelling can be viewed as a process of natural deduction constrained by a proof structure, which consists in building proof trees in parallel as far as possible.
The process of minimal labelling succeeds if it converges on the external output of the proof structure. That means that all proof trees have been merged in a unique proof tree, the conclusion of which is the output of the proof structure. The correctness criterion of de Groote for intuitionistic proof nets follows naturally from these considerations.

Definition 1.7 *A proof structure is a proof net iff the minimal labelling such that the external inputs are labelled with $\alpha_1, \ldots, \alpha_n$ has the output labelled with $\alpha_1 + \cdots + \alpha_n$.*

According to this definition, the proof structure of Example 1.5 is not a proof net whereas that of Example 1.6 is.
The following theorem gives an equivalent definition of the correctness criterion, which highlights the necessity for the labelling function to be total.

Theorem 1.8 *A proof structure is a proof net iff its minimal labelling is total.*

PROOF. Let Π be a proof net. We want to prove that the minimal labelling of Π is a total function. Let F be any formula occurrence of Π. Let n be the depth at which F is situated in Π, this being the number of links that are under F. We propose to

[5] The co-indexation of an abstraction and the corresponding cancelled hypothesis is explicitely realized in a proof net by means of an $\multimap$-output link.

show by induction on n that F belongs to the natural deduction tree of a formula attached to a down edge.
For $n = 0$, according to Definition 1.7, the external output of Π has a natural deduction tree and the external inputs of Π are leaves of this tree.
For the induction step, we know by induction hypothesis that the conclusion G of the link which has F as a premise belongs to the natural deduction tree of a down formula. Consequently, F belongs to the same tree. As a consequence, all formula occurrences of Π belong to natural deduction trees and thus all are labelled.
Conversely, let Π be a proof structure for which the minimal labelling is total. We want to prove that Π is a proof net. According to the definition of a proof net, we have to prove that the output F of Π is labelled with $\alpha_1 + \cdots + \alpha_n$, $\alpha_1, \ldots, \alpha_n$ being the labels of the external inputs of Π.
Let G be any input of Π. Since Π is totally labelled, the longest labelling path from G in D_F terminates on F. If G is a virtual input, the path goes through the $\multimap$-output link that has G as premise. In this case, the label of G is not present in the label of F. If G is an external input, its label is present in that of F. As a result, F is labelled with $\alpha_1 + \cdots + \alpha_n$, $\alpha_1, \ldots, \alpha_n$ being the labels of the external inputs of Π. ∎

The natural deduction tree of the external output of a proof net can be viewed as a proof in the natural deduction formalism that is extracted from a proof net by minimal labelling. This proof is normal in the sense that it contains no redex: an abstraction is never immediately followed by a corresponding application. The converse process, which consists in building a proof net from a normal natural deduction proof, works also but, for this, we slightly relax the syntax of proof nets by allowing axiom links between complex formulas. This modification is standard and it leaves the set of theorems unchanged. The only reason for making it is to simplify the transformation of a natural deduction proof into a proof net.

Theorem 1.9 *A formula G is provable from the hypotheses $F_1, \ldots, F_n$ in the IILL natural deduction system iff there exists a proof net with $F_1, \ldots, F_n$ as external inputs and G as the external output.*

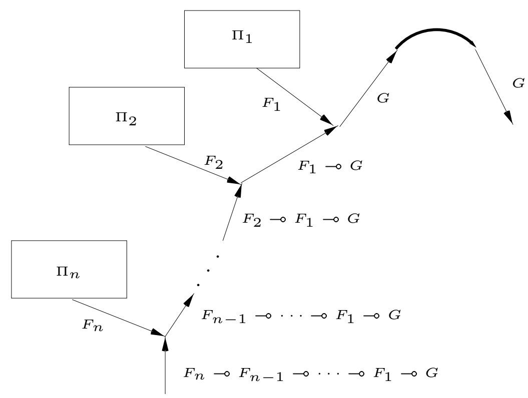
Proof. To extract a natural deduction proof from a proof net, it is sufficient to consider the natural deduction tree of the output of the proof net.
Conversely, we assume a natural deduction proof D in normal form and we have to prove that there exists a corresponding proof net Π. We proceed by induction on the structure of D.
If D is reduced to a unique formula G, there exists a proof net that is reduced to an axiom link between two formulas G. If D includes at least one inference, we distinguish two cases according to the type of the last inference I of D.

1. *I is an abstraction.*
   Let G and $F \multimap G$ be respectively the premise and the conclusion of I. By induction hypothesis, there exists a proof net with G as the output and with the hypotheses of D plus F as the external inputs. By adding an $\multimap$-output link, we obtain a proof structure which verifies the criterion given in Definition 1.7 and this is the expected proof net.
2. *I is an application.*
   Since D is in normal form, it has the following form:

$$\dfrac{\begin{array}{c}\vdots\\ F_n \quad F_n \multimap \cdots \multimap F_1 \multimap G\end{array}}{F_{n-1} \multimap \cdots \multimap F_1 \multimap G}\multimap_E$$

$$\dfrac{\begin{array}{c}\vdots\\ F_1\end{array} \qquad \dfrac{\begin{array}{c}\vdots\\ F_2\end{array} \qquad \begin{array}{c}\vdots\\ F_2 \multimap F_1 \multimap G\end{array}}{F_1 \multimap G}\multimap_E}{G}\,\mathrm{I}$$

By induction hypothesis, there exist n proof nets $\Pi_1, \ldots, \Pi_n$ that result from the transformation of the natural deduction proofs of $F_1, \ldots, F_n$. So, we can build the following proof structure that corresponds to the proof D.



The correctness criterion is easily verified and consequently we obtain the expected proof net.

∎

If we define the IILL sequent calculus by restricting the inference rules of ILL to the left and the right introduction rules of the linear implication, the equivalence of provability with proof nets and provability in the IILL sequent calculus is a corollary of Theorem 1.9. De Groote gives a direct proof of this corollary in [3].

1.3 From minimal labelling to complete labelling

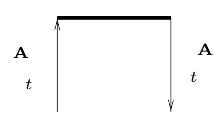
Minimal labelling checks the minimum of information that has to be propagated through a proof net in order to verify its correctness. By contrast, *complete labelling* is used to propagate the information that is necessary to rebuild the whole of the proof net when this information arrives at the output. Complete labelling resorts to λ-terms which are well suited to represent the structure of intuitionistic proofs like IILL proof nets.

To define complete labelling, we assume a set $\mathcal{V}$ of variables. ¿From $\mathcal{V}$ we build a set $\mathcal{T}$ of λ-terms by means of the operations of application $(\cdot\cdot)$, for which a convention of

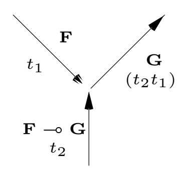
left associativity is assumed, and abstraction $\lambda x.\cdot$. The variables of $\mathcal{V}$ will be denoted x, y, z, possibly with subscripts, and the terms of $\mathcal{T}$ will be denoted t, r, s, possibly with subscripts.
The labelling algorithm is the same as that for minimal labelling, only the nature of labels changes.

Definition 1.10 *All actual and virtual inputs of the proof structure are labelled with distinct variables of* $\mathcal{V}$*. Then, for each link that has its incoming edges labelled and one outgoing edge not labelled, this edge is labelled according to one of the following rules:*

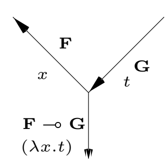
rule 1: *if the link is an axiom link, the label of the incoming edge is transmitted to the outgoing edge;*



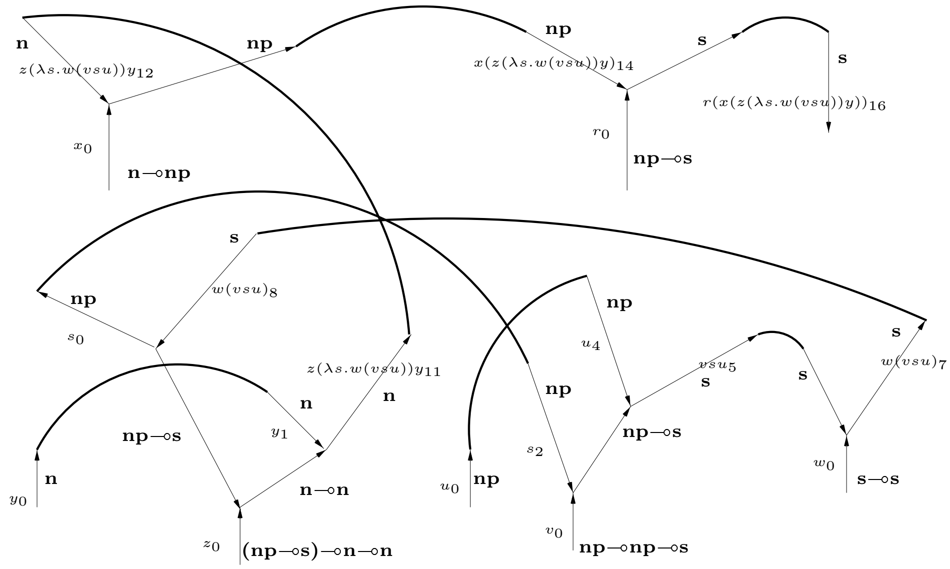
rule 2: *if the link is an* $\multimap$*-input link, the up premise is labelled with the application of the label of the conclusion to the label of the down premise;*



rule 3: *if the link is an* $\multimap$*-output link, the conclusion edge is labelled with the abstraction of the label of the down premise with respect to the variable labelling the up premise.*



Example 1.11 *Let us take again the proof structure representing the syntactic structure of the sentence* the woman whom Peter met yesterday smokes*. Step by step, we label the structure with* λ*-terms. As with the example of minimal labelling, the rank of each step in the process of labelling is indicated as a subscript of the corresponding new label.*



With the term $r(x(z(\lambda s.w(vsu))y))$ which labels the output of the proof net, we are able to reconstruct all of the proof net as indicated by Theorem 1.9.

The λ-terms that label a proof structure are linear λ-terms, that is, terms where free variables occur exactly once and where each abstraction contains a unique occurrence of its bound variable in its body.
The complete propagation rules show that complete labelling propagates more information than minimal labelling:

- in rule 2, t_1 and t_2 are not permutable because they occur in an application where t_2 is a function and t_1 its argument whereas, in minimal labelling, the resulting label keeps no trace of the asymmetry between ω_1 and ω_2;
- in rule 3, the resulting abstraction keeps trace of the bound variable whereas, in minimal labelling, ω_1 totally disappears in the resulting label.

Now, we can extract a minimal labelling from the complete one by considering the set of free variables that are present in the complete labels.
The following theorem highlights the fact that complete labelling encodes all the information that is necessary to recover the structure of the proof net.

Theorem 1.12 *For any linear λ-term t typed with an IILL formula F, there is a unique proof net Π and a complete labelling of Π such that the label of the output F of Π is t.*

Proof. By induction on the structure of t. ∎

Even if proof nets and lambda-terms are two equivalent forms of the same object, each one is well suited to a particular use: whereas lambda-terms are used in the framework of functional programming, proof nets are more appropriate in the framework of proof search, as is the case here.

1.4 Labelled proof net construction

NP-completeness of IILL follows directly from the proof of NP-completeness for the multiplicative fragment of propositional linear logic [5]. Thus, we cannot hope to obtain algorithms for proof net construction in IILL which are tractable and general at the same time.

1.4.1 A first algorithm

If we consider minimal labelling, the associated correctness criterion gives us a general algorithm for building proof nets. Let $F_1, \ldots, F_n \vdash G$ be an IILL sequent to be validated. The decision algorithm to do so can be described as follows:

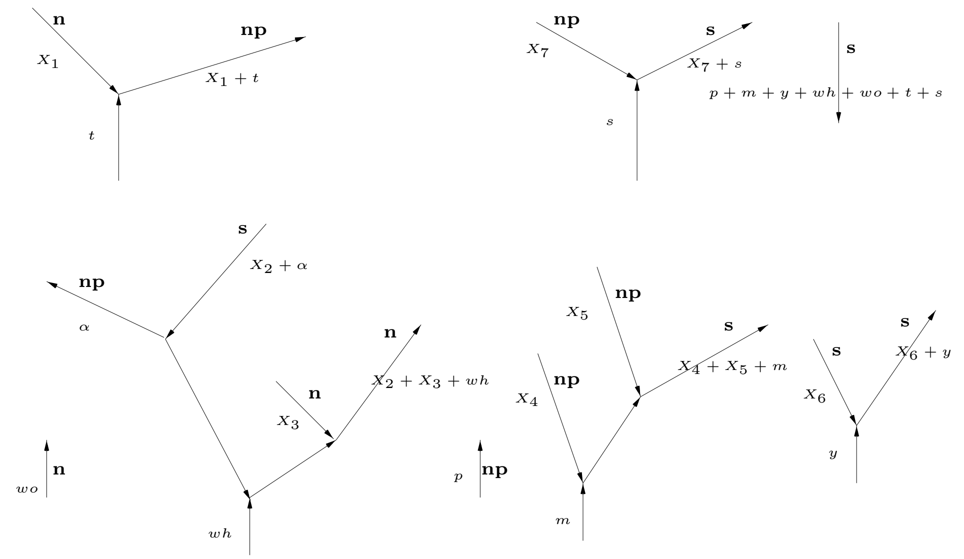
1. We initialise the proof frame Π with n inputs $F_1, \ldots, F_n$ and one output G.
2. From the initial edges, we expand the syntactic trees corresponding to the formulas $F_1, \ldots, F_n, G$ up to their atomic components.
3. We label all actual and virtual inputs of Π with distinct primes of $\mathcal{M}$.
4. We initialise the set *Out* of the internal outputs that are candidate for axiom links to the ones that are labelled.
5. While there is an element of *Out* for which a new connection by means of an axiom link is possible,
   (a) we choose such an internal output x;
   (b) while x has not been connected by an axiom link and there is a new possibility of connection,
      i. we connect x to a new internal input x' by means of an axiom link;
      ii. we propagate the label of x as far as possible;
      iii. if the propagation does not succeed because rule 3 of minimal propagation fails, we remove the axiom link between x and x'.
   (c) If the connection succeeds, then we update *Out* else we backtrack to the point that precedes the last addition of an axiom link.

At the end, if Π is internally closed, according to Theorems 1.8 and 1.9, the sequent $F_1, \ldots, F_n \vdash G$ is valid; otherwise, the sequent is not valid.

1.4.2 Reduction of the problem to building and solving a system of equations

We can improve the previous algorithm to anticipate and consequently to avoid the failures caused by rule 3 of minimal labelling. In the same way we have extended the notion of input to virtual input, we extend the notion of output to *virtual output*. In a proof frame, a virtual output is the positive premise of an $\multimap$-input link. Thus, we consider a set $\mathcal{V}_{\mathcal{M}}$ of variables on $\mathcal{M}$ which are denoted X, Y, Z possibly with subscripts. We label all outputs of the initial frame with distinct elements of $\mathcal{V}_{\mathcal{M}}$. Then we propagate labels to the leaves of each syntactic tree that constitutes the initial frame. By doing this, we find again the labelling which was introduced by Roorda [22] but which was not proved to be sufficient for the correctness of a proof net. Let us take our previous example again to illustrate this.

Example 1.13
Proving that the sentence the woman whom Peter met yesterday smokes *is grammatically correct leads to proving the sequent:* $n \multimap np,\ n,\ (np \multimap s) \multimap n \multimap n,\ np,\ np \multimap np \multimap s,\ s \multimap s,\ np \multimap s\ \vdash\ s$*. The formulas that constitute the antecedent of the sequent represent the grammatical categories of the words of the sentence. Using variables, we initialise the proof frame as follows.*



In this way, the problem of proof net construction in IILL is reduced to a problem of building and solving equations from two given sets of terms in a monoid. If the sequent $F_1, \ldots, F_n \vdash G$ is valid, the corresponding initial proof frame contains as many internal outputs as internal inputs. Therefore assume this to have been confirmed. Each internal output is the end of a path that begins at an actual or virtual input and thus corresponds to an element α of $\mathcal{M}$ that has been chosen to label the input. Thus, its label has the form $X_1 + \cdots + X_k + \alpha$. Dually, each internal input is the beginning of a path that ends with an actual or virtual output. One corresponds to the unique actual output and its label has the form $\alpha_1 + \cdots + \alpha_l$. Each of the others corresponds to a virtual output labelled with a variable X of $\mathcal{V}_{\mathcal{M}}$. Thus, the label of each internal input has the form $X + \alpha_1 + \cdots + \alpha_h$.
Consequently, the problem of adding axiom links to the proof frame so that we obtain a proof net can be expressed as follows. We have two lists of terms: *output terms* which are the labels of internal outputs and *input terms* which are the labels of internal inputs.

output terms
$X_{k_1} + \cdots + X_{k_{m_1}} + \alpha_1$
$\vdots$
$X_{k_{p-1}} + \cdots + X_{k_{m_{p-1}}} + \alpha_{p-1}$
$X_{k_p} + \cdots + X_{k_{m_p}} + \alpha_p$

input terms
$\alpha_{h_1} + \cdots + \alpha_{h_{n_1}} + X_1$
$\vdots$
$\alpha_{h_{p-1}} + \cdots + \alpha_{h_{n_{p-1}}} + X_{p-1}$
$\alpha_{h_p} + \cdots + \alpha_{h_{n_p}}$

The last entry in the list of input terms labels the output of the future proof net and it is determined by the correctness criterion given in Definition 1.7: if the external inputs are labelled with $\alpha_{h_p}, \ldots, \alpha_{h_{n_p}}$, the output is labelled with $\alpha_{h_p} + \cdots + \alpha_{h_{n_p}}$. Then we have to find a bijective mapping from the first list to the second list such that the resulting system of p equations is solvable.

Example 1.14 *If we continue with our example, we obtain the following lists:*

output terms	
terms	*types*
$X_1 + t$	np
wo	n
α	np
$X_2 + X_3 + wh$	n
p	np
$X_4 + X_5 + m$	s
$X_6 + y$	s
$X_7 + s$	s

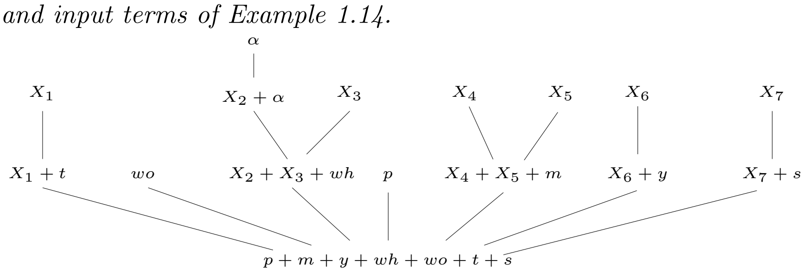
input terms	
terms	*types*
X_1	n
$X_2 + \alpha$	s
X_3	n
X_4	np
X_5	np
X_6	s
X_7	np
$p + m + y + wh + wo + t + s$	s

Now, we have an algebraic representation of our problem: the two lists of terms represent the initial proof frame, the p equations that we have to find represent the p axiom links that are added to produce a proof structure. The correctness criterion is expressed by the solvability of the system of p equations.
Now, we can ask ourselves if every problem that is formulated in this setting can be viewed as a problem of proof net construction. The answer is no because to represent a proof frame the two lists of terms must verify a condition which is expressed by means of a *dependence relation* between terms. This relation, which is well known in the area of game semantics [9], was promoted by Lamarche as a tool for abstracting the minimum information needed to represent a proof frame.

Definition 1.15 *An output term depends on an input term iff the variable corresponding to the second is present in the first and an input term depends on an output term iff the constant corresponding to the second is present in the first.*

With this definition, the two lists of terms represent a problem of proof net construction if the graph of the associated dependence relation is a tree.

Example 1.16 *Here is the graph that represents the relation of dependence for the output and input terms of Example 1.14.*



This graph is a tree because the lists correspond to a proof frame, the proof frame of Example 1.13.

1.4.3 The improved algorithm in an algebraic setting

The algebraic setting is used to improve the initial algorithm in such a way that failures caused by rule 3 of minimal labelling are avoided.

1. We initialise the list l_o of output terms and the list l_i of input terms.
2. While one among the two following alternatives is possible, we choose it:
   - If there exists a *constant term* t_o of l_o of type F and a term t_i of l_i of type F for which the equation $t_o = t_i$ is solvable and has not been yet tested, we select them.
     We respectively delete t_o and t_i from l_o and l_i.
     In l_o, we instantiate the variable associated with t_i with the solution of the equation.
   - If no such equation can be found, we backtrack to the point that immediately precedes the last equation that has been built.

At the end, if the lists l_o and l_i are empty, the sequent is valid, otherwise, it is not.

Example 1.17 *By initialising l_o and l_i with the lists of Example 1.14, the algorithm generates two proof nets for the sequent $n \multimap np,\ n,\ (np \multimap s) \multimap n \multimap n,\ np,\ np \multimap np \multimap s,\ s \multimap s,\ np \multimap s\ \vdash\ s$. The first corresponds to the syntax of the sentence* the woman whom Peter met yesterday smokes*. The second corresponds to an ungrammatical sentence where the relative pronoun is the subject of the verb* met*. We will prevent this by adding case information to the noun phrases that are present in the sentence.*[6]

We can imagine a dual algorithm that works in the converse direction: the equations are built from constant input terms which means that the proof net is built from the output to the inputs. The main difference is that equations can have many solutions so that a part of non-determinism is transferred from the choice of the equations to the choice of their solutions. Such an algorithm is used by Morrill [18], Merenciano and Morrill [11] in a non-commutative framework.

2 Application to the syntax and the semantics of natural languages

We use IILL proof nets to represent all dependencies between the syntactic constituents of a sentence. With minimal labelling, the labels that are propagated through the proof net can be interpreted as phonological components of the sentence but we must add order to the monoid to take into account word order. With this addition, minimal labelling takes the form of *phonological labelling*.
Then, in line with the Montagovian tradition [12], we want to keep syntax and semantics of natural languages very close. This takes the form, in the first place, of an intimate connection between syntactic categories and semantic types, a principle of Categorial Grammars embodied in the Montagovian type map. Then complete labelling appears as an encoding of the syntax that is used to produce the meaning of a sentence from the meaning of its components. That is why this second form of

[6] Such a solution, as a referee remarked it, is not completely satisfactory because this leads to rejecting grammatical phrases like *the woman whom Peter believes smokes*.

labelling is called *semantic labelling*.
Finally, neither kind of labelling is sufficient to produce the phonological form and the meaning of the sentence from its elementary components. Additional syntactic and semantic information is necessary and, for this, we use *feature matrices*. By integrating the phonological and semantic labels as particular features of these matrices, we keep only one form of labelling. In this way, the mechanism of parsing which was initially based on the identification of internal outputs with internal inputs by means of axiom links is enriched with unification of feature structures in a way similar to Unification Grammars [1].

2.1 Minimal labelling plus order as phonological labelling

2.1.1 Linear logic formulas as syntactic categories

We assume a finite set of IILL predicates. Each predicate with its arguments represents a grammatical category which is enriched with features. Each argument position corresponds to a feature and it is filled by a term that represents a value of this feature. For instance, *np(nom)* represents a noun phrase in the subject position: the feature *case* has the value *nominative*. Variables are used for expressing unspecified values of some features. They are denoted by capital letters. For instance, in the noun phrase *Peter*, the case is not constrained, which can be expressed by the predication *np(C)* where C is a variable.
All predicates with their arguments constitute the set Cat_0 of basic grammatical categories. The set Cat of all grammatical categories is built from Cat_0 in the logical framework of IILL extended with universal quantification. As in Prolog, all formulas are closed and universal quantification is implicit.

Example 2.1 *The appositive, and restrictive functions of a relative clause correspond to the respective categories of the relative pronoun whom* $(np(acc) \multimap s) \multimap np(C) \multimap np(C)$ *and* $(np(acc) \multimap s) \multimap n \multimap n$. *The relative pronoun whom expects a sentence where the object of a verb is missing and which has the category* $np(acc) \multimap s$ *to build a relative clause. This clause acts as noun phrase modifier of the category* $np(C) \multimap np(C)$ *when it is appositive or a common noun modifier of the category* $n \multimap n$ *when it is restrictive. In the first case,* $np(C) \multimap np(C)$ *means* $\forall C\ (np(C) \multimap np(C))$.

By extending the propositional framework with universal quantification, we enrich the basic operation of the parsing process: putting an axiom link, which before amounted to matching two types, now becomes unification of two basic categories.
Unlike Lambek Calculus, IILL expresses only the valences of grammatical categories with logical formulas. Because of commutativity, it cannot express the order in which syntactic constituents are aggregated to build bigger ones. This will be represented at a non-logical level with phonological terms.

2.1.2 Phonological terms

Phonological information is represented by minimal labelling enriched with order. For this, instead of working in a commutative monoid, we work in a non-commutative monoid the terms of which are called *phonological terms* and defined as follows. Let $\mathcal{W}$ be the set of words in a particular natural language. The phonological terms built

from $\mathcal{W}$ are sequences of words. In a compositional view, they represent phrases which are composed together to form more complex phrases. Sometimes, a phrase includes a gap which corresponds to a missing syntactic constituent. This gap will be filled further in the interaction of the phrase with another. For representing gaps inside phonological terms, we assume a countable set $\mathcal{T}$ of *traces*. Traces are denoted α, β, γ, ... For example, *Peter.met.*α*.yesterday* is a phonological term which contains a trace α and this trace plays the role of the object for the verb *met*.
Formally, the set *Phon* of phonological terms is the set of finite concatenations of elements of $\mathcal{W} \cup \mathcal{T}$.

2.1.3 A lexicon of labelled syntactic trees

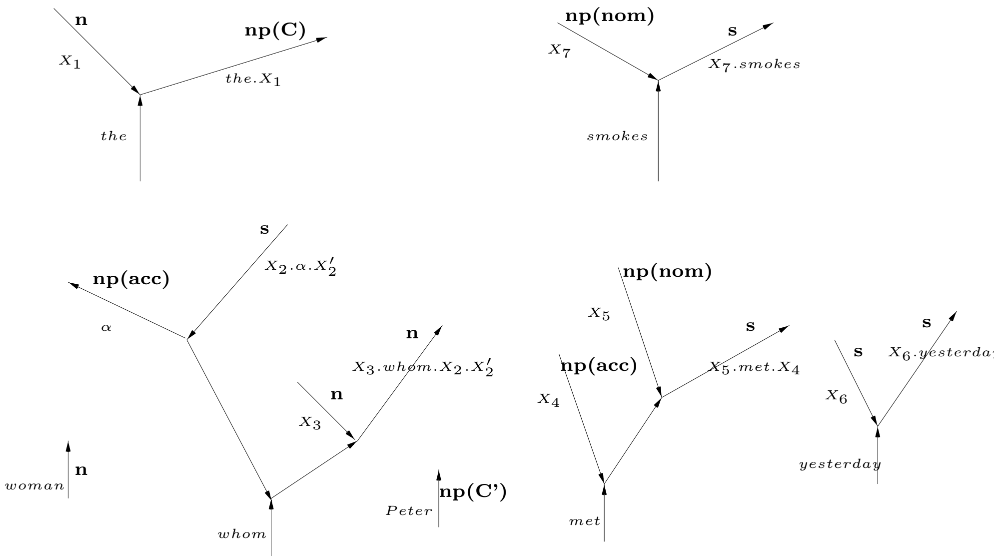
In accordance with lexicalism, all the information necessary to the analysis of a sentence is given by entries in a lexicon corresponding to the words comprising the sentence. On the basis of this information and the general rules of proof net construction, we build the labelled proof net representing the syntactic structure of the sentence.
Each lexical entry is structured as a syntactic tree which is the expanded form of its grammatical category. Each internal input of the tree is labelled with a phonological term where variables can be present. Such a term acts as a filter for the information that arrives at the tree by means of axiom links. In the pure algorithm for proof net construction given in Subsection 1.4, the labels of the internal inputs, which have the form $X+\alpha_1+\alpha_2+\cdots+\alpha_h$, already acted as filters, but now, non-commutativity adds new constraints: the labels of internal inputs become $X_0.\alpha_1.X_1.\alpha_2.X_2\cdots\alpha_h.X_h$ where non-commutativity of the operation prevents collapsing the variables $X_0,\ldots,X_h$. As regards the labels of internal outputs, again we find the variables of the input labels and they are used to compute the output labels as functions of the input ones.
Before defining the syntax of lexical entries formally, let us study an example.

Example 2.2 *The entries of the lexicon that are used to parse the sentence* the woman whom Peter met yesterday smokes *may take the form given by the diagram below.*[7]
For instance, the relative pronoun whom *has an entry that acts in the process of parsing as follows. It produces a trace α which combines with two phonological terms X_2 and X_2' to constitute a sentence $X_2.\alpha.X_2'$ where it plays the role of a noun phrase in the accusative.*[8] *The sentence then is transformed into the relative clause whom.$X_2.X_2'$. This clause expects a common noun X_3 to modify it and to transform it into the common noun X_3.whom.$X_2.X_2'$.*
In the parsing process, if X_2 or X_2' are empty, we recover the peripheral extraction from a relative clause which is expressible in the Lambek Calculus and if both X_2 and X_2' are not empty, we obtain medial extraction which is not expressible in the Lambek Calculus. This is an illustration of the flexibility of the model.
Of course, labelling of a syntactic tree must be consistent with the rules of minimal labelling. For example, in the syntactic tree associated with whom*, if we forget order on the labels and if we propagate the labels from the inputs to the output on the left,*

[7] The variables and constants that are present in the labels of the lexical entry for each occurrence of a word in a string to be analysed must be fresh.

[8] In more complex relative clauses like *the woman whom Peter believes smokes*, the relative pronoun *whom* can bind the subject of a verb as we have mentionned it previously.

we find $X_3 + whom + X_1 + X_2$ *again.*



The previous example will help us understand the formal definition given of the syntax of entries in the lexicon. For this, we consider a countably infinite set $\mathcal{V}$ of variables. The elements of $\mathcal{V}$ are denoted X, Y, Z ..., possibly with subscripts. Then, from $\mathcal{V}$ and *Phon*, we build the set *Phon*$[\mathcal{V}]$ of phonological elements which are concatenations of elements of $\mathcal{V} \cup \mathcal{W} \cup \mathcal{T}$. On this basis, we can define the syntax of lexical entries.

Definition 2.3 *Each entry of the lexicon is the syntactic tree of a category taken from Cat the leaves of which are labelled with phonological elements of* Phon$[\mathcal{V}]$ *according to the following rules:*

- *two variables cannot be consecutive in the label of an input;*
- *labelling must be coherent with the rules of minimal propagation (assuming that the operation becomes commutative, the rules of minimal labelling hold).*

The first condition in the definition above is technical: since the labels of inputs are used to screen the incoming information, two consecutive variables are useless and can be replaced with a single one.
The second condition aims to preserve the linearity of syntactic dependences, which is necessary if we want to remain in the framework of multiplicative linear logic. At the same time, this is a limitation because some non-linear linguistic phenomena are not expressible in this framework.

2.1.4 Parsing a sentence as building a proof net

Parsing a sentence amounts to building a proof net from the syntactic trees that are extracted from the lexicon and correspond to the words of the sentence. For this, we can use the general algorithm described in Subsection 1.4 with one refinement: phonological labelling must not only respect the general rules of minimal labelling but also the constraints on word order given by the lexicon: now, equations are

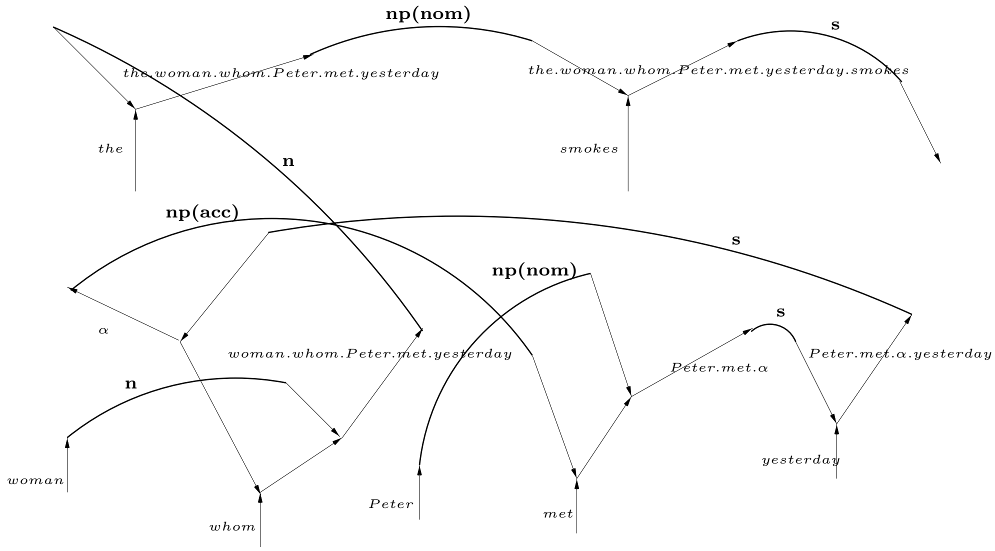


FIG. 1: Proof net representing the syntax of the sentence *the woman whom Peter met yesterday smokes*

situated in a non-commutative monoid and they express additional constraints under the form of pattern matching.

Example 2.4 *For parsing of the sentence* the woman whom Peter met yesterday smokes*, we assume that the lexical entries described in Example 2.2 constitute the initial proof frame. From this, the lists l_o and l_i of output and input terms that characterise the proof frame are defined as follows:*

output terms	
terms	*types*
the	$np(C)$
$woman$	n
α	$np(acc)$
$X_3.whom.X_2.X'_2$	n
$Peter$	$np(C')$
$X_5.met.X_4$	s
$X_6.yesterday$	s
$X_7.smokes$	s

input terms	
terms	*types*
X_1	n
$X_2.\alpha.X'_2$	s
X_3	n
X_4	$np(acc)$
X_5	$np(nom)$
X_6	s
X_7	$np(nom)$
$the.woman.whom.Peter.met.yesterday.smokes$	s

Parsing succeeds with a unique solution which is the proof net with its phonological labelling given by Figure 1. For clarity, not all labels are marked.

As we mentioned at the beginning of Subsection 1.4, general algorithms for proof net construction in a commutative framework are not efficient because the number of axiom links we have to test is exponential with respect to the number of atomic formulas of the proof frame. When we apply them to parsing sentences as we propose here, there is an important difference: the operation on phonological terms is non-commutative and the phonological terms attached to the inputs of syntactic trees play the role of filters. In this way, the search space for input labels that match a given output label is restricted in such way that the disadvantage of using a commutative framework is much reduced.

2.2 Complete labelling as semantic labelling

2.2.1 Semantic types

The close connection between syntax and semantics is expressed in particular by the fact that each syntactic category corresponds to a unique semantic type.
The set Typ of semantic types is defined by the grammar:

$$Typ \ ::= B \mid I \mid Typ \rightarrow Typ$$

The constructor $\rightarrow$ is assumed to be right associative. The basic types B and I represent boolean values and individuals respectively.
Now, we assume a type assignment function $Type$ from Cat_0 to Typ. For instance, $Type(s) = B$, $Type(n) = I \rightarrow B$. Then, we extend the function $Type$ to Cat by the property:

$$Type(A \multimap B) = Type(A) \rightarrow Type(B)$$

Example 2.5 *In Example 2.2, the lexicon gives the syntactic category* $(np(acc) \multimap s) \multimap n \multimap n$ *to the relative pronoun* whom*. From a semantic point of view, this relative pronoun takes a property as argument which is represented by the relative clause and modifies another property which is represented by the common noun that precedes it. Consequently, its semantic type, which is automatically deduced from the syntactic category, is:* $(I \rightarrow B) \rightarrow (I \rightarrow B) \rightarrow I \rightarrow B$.

2.2.2 Semantic terms

The meaning of correct phrases will be given by semantic terms. A semantic term is a term of higher-order logic which is represented in the framework of the simply typed λ-calculus. The types of higher-order logic are the elements of Typ which are constructed from the basic types I and B by means of the arrow operator. For each type t, we assume a countably infinite set $\mathcal{V}_t$ of variables and a finite set $\mathcal{C}_t$ of constants. Higher-order logic requires that some constants be logical. For example, the conjunction $\wedge$ has the type $B \rightarrow B \rightarrow B$ and the universal quantifier $\forall_t$ related to elements of type t has the type $(t \rightarrow B) \rightarrow B$. Then, semantic terms are recursively built from these sets of variables and constants by abstraction and application.

Example 2.6 $\lambda PQx.(Px) \wedge (Qx)$ *is the semantic term that gives the meaning of the relative pronoun* whom*. This is a function which takes two properties* P *and* Q *and returns their intersection* $\lambda x.(Px) \wedge (Qx)$*. This conforms to its semantic type:* $(I \rightarrow B) \rightarrow (I \rightarrow B) \rightarrow I \rightarrow B$*. The properties* P *and* Q *represent the meanings of, respectively, the relative clause body and the common noun that is modified by the relative clause.*

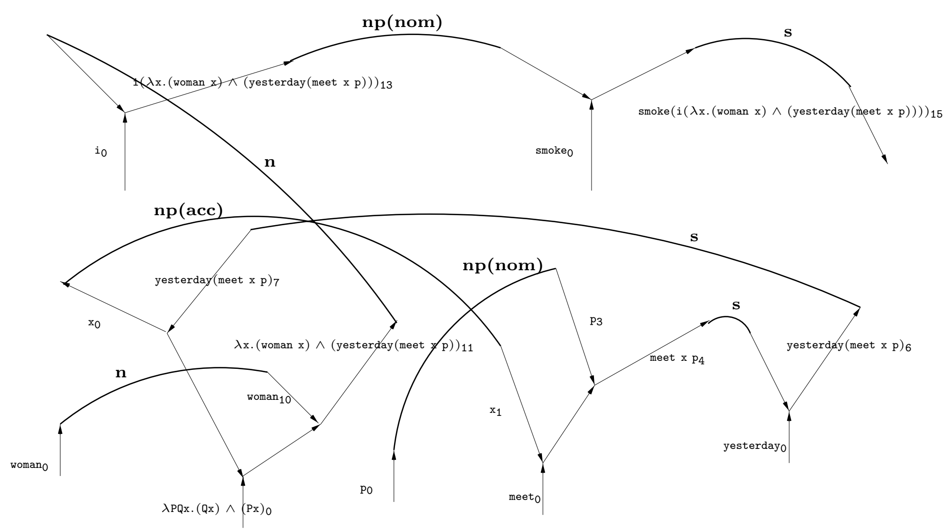
2.2.3 Semantic labelling for building the meaning of a sentence

In the same way that we use minimal labelling of proof nets to build the phonological form of a sentence, we use complete labelling to build its meaning. Both forms of labelling are processed in parallel during the process of parsing. Initially, the actual inputs of the proof frame are labelled with semantic terms given by the lexicon and the virtual inputs are labelled with distinct variables. Then at the same time that a new phonological label is added, a semantic label is attached to the same edge by application of a rule of complete propagation. At the end of the process, the output of the proof net is labelled with a term that represents the meaning of the whole sentence.

Example 2.7 *Let us examine how the meaning of the sentence* the woman whom Peter met yesterday smokes *can be built. The lexicon gives a meaning of each word making up the sentence in the form of a semantic term. Its semantic type is consistent with the syntactic category of the word.*

word	*semantic term*	*semantic type*	*syntactic category*
the	i	$(I \rightarrow B) \rightarrow I$	$n \multimap np(C)$
woman	$woman$	$I \rightarrow B$	n
whom	$\lambda PQx.(Px) \wedge (Qx)$	$(I \rightarrow B) \rightarrow (I \rightarrow B) \rightarrow I \rightarrow B$	$(np(acc) \multimap s) \multimap n \multimap n$
Peter	p	I	$np(C')$
met	$meet$	$I \rightarrow I \rightarrow B$	$np(acc) \multimap np(nom) \multimap s$
yesterday	$yesterday$	$B \rightarrow B$	$s \multimap s$
smokes	$smoke$	$I \rightarrow B$	$np(nom) \multimap s$

From these lexical entries, semantic labelling gives the following result (the rank of each step in the process of labelling is added as a subscript of the corresponding new label):



The meaning of the sentence the woman whom Peter met yesterday smokes *is provided by the output of the proof net in the form of the proposition:*
$smoke(i(\lambda x.(woman\ x) \wedge (yesterday(meet\ x\ j))))$.

2.3 Feature structures as an integrated form of labelling

As we have shown in Subsection 2.1, building of well-formed sentences is controlled by phonological labelling which guarantees good management of resources and word order at the same time. Nevertheless it is not sufficient to take all subtleties of natural languages into account. For instance, relative clauses are barriers to extraction but, as we have already mentioned, the mechanism of phonological labelling is unable to detect the ungrammaticality of the following sentence:

$*$ *the woman whom Peter who met yesterday lives in London smokes*

Additional syntactic information must be taken into account in the process of parsing. One way of doing this is to use features structures in the manner of Unification Grammars [1]. Up until now, we have used three kinds of labels for the proof nets that represent the syntactic structure of sentences: syntactic categories, phonological terms and semantic terms. We propose to integrate them in a unique structure in the form of a *feature matrix*. Thus, the three kinds of labels become three features of this matrix, respectively *Cat*, *Phon* and *Sem*. Other features are added to express additional information concerned with agreement, extraction, etc.
Since from now on we have only one kind of label, parallel labelling processes are integrated in a unique process. The key to this process consists in the pairwise identification of internal outputs with internal inputs by means of axiom links. Now, this mechanism takes the form of *matrix unification* in the same sense as in Unification Grammars [1].
The parsing algorithm can be summarised as follows:

- Initially, the lexicon provides two lists of feature matrices that correspond to the internal outputs and the internal inputs of the initial proof frame. The presence of variables in feature values is essential: they realize linking between output and input matrices in order to constitute the syntactic trees that form the initial proof frame.
- A parsing step consists in selecting a matrix of the output list that has its *Phon* feature completely determined and in finding a matrix from the input list which is unifiable with the first matrix. In the process of unification, some variables are instantiated in the input matrix and in this way, they realize the transmission of some features from an input leaf of a syntactic tree to output leaves of the same tree. Both matrices which have been unified are deleted from the lists.
- Parsing succeeds if, finally, it remains only one output matrix which corresponds to the complete sentence.

Example 2.8 *In the new setting where labels are feature matrices, the lexicon provides the two lists of matrices given by Table 1 for parsing the sentence* the woman whom Peter met yesterday smokes*. The feature* Move *which is present in some matrices indicates the set of traces that are included in the corresponding syntactic constituent and represents gaps that must by filled. It is used to restrict the movements of constituents that are extracted. When a constituent such as a relative clause represents a barrier to extraction, its feature* Move *is empty.*
After seven steps of unification between the output and input matrices, the process of parsing succeeds with the following matrix as result:

$$\begin{bmatrix} Cat & = & s \\ Phon & = & the.woman.whom.Peter.met.yesterday.smokes \\ Sem & = & smoke(i(\lambda x.(woman\ x) \wedge (yesterday(meet\ x\ j)))) \\ Move & = & \{\ \} \end{bmatrix}$$

words	*output matrices*	*input matrices*
the	$\begin{bmatrix} Cat & = & np \\ Phon & = & the.X_1 \\ Sem & = & i\ P \\ Pers & = & 3 \\ Move & = & \{\ \} \end{bmatrix}$	$\begin{bmatrix} Cat & = & n \\ Phon & = & X_1 \\ Sem & = & P \end{bmatrix}$
woman	$\begin{bmatrix} Cat & = & n \\ Phon & = & woman \\ Sem & = & woman \\ Pers & = & 3 \\ Num & = & sg \\ Gen & = & f \end{bmatrix}$	
whom	$\begin{bmatrix} Cat & = & np \\ Phon & = & \alpha \\ Sem & = & x \\ Case & = & acc \\ Move & = & \{\alpha\} \end{bmatrix}$ $\begin{bmatrix} Cat & = & n \\ Phon & = & X_3.whom.X_2.X_2' \\ Sem & = & \lambda x.(Q\ x) \wedge B_1 \end{bmatrix}$	$\begin{bmatrix} Cat & = & s \\ Phon & = & X_2.\alpha.X_2' \\ Sem & = & B_1 \\ Move & = & \{\alpha\} \end{bmatrix}$ $\begin{bmatrix} Cat & = & n \\ Phon & = & X_3 \\ Sem & = & Q \end{bmatrix}$
Peter	$\begin{bmatrix} Cat & = & np \\ Phon & = & Peter \\ Sem & = & p \\ Num & = & sg \\ Pers & = & 3 \\ Move & = & \{\ \} \end{bmatrix}$	
met	$\begin{bmatrix} Cat & = & s \\ Phon & = & X_5.met.X_4 \\ Sem & = & meet\ I_1\ I_2 \\ Move & = & E_1 \end{bmatrix}$	$\begin{bmatrix} Cat & = & np \\ Phon & = & X_4 \\ Sem & = & I_1 \\ Case & = & acc \\ Move & = & E_1 \end{bmatrix}$ $\begin{bmatrix} Cat & = & np \\ Phon & = & X_5 \\ Sem & = & I_2 \\ Case & = & nom \end{bmatrix}$
yesterday	$\begin{bmatrix} Cat & = & s \\ Phon & = & X_6.yesterday \\ Sem & = & yesterday\ B_2 \\ Move & = & E_2 \end{bmatrix}$	$\begin{bmatrix} Cat & = & s \\ Phon & = & X_6 \\ Sem & = & B_2 \\ Move & = & E_2 \end{bmatrix}$
smokes	$\begin{bmatrix} Cat & = & s \\ Phon & = & X_7.smokes \\ Sem & = & smoke\ I_3 \\ Move & = & \{\ \} \end{bmatrix}$	$\begin{bmatrix} Cat & = & np \\ Phon & = & X_7 \\ Sem & = & I_3 \\ Case & = & nom \\ Num & = & sg \\ Pers & = & 3 \end{bmatrix}$

TABLE 1: Lexical entries for the sentence *the woman whom Peter met yesterday smokes*

Now, the following sentences are correctly parsed:

$*$ the woman whom Peter who met yesterday lives in London smokes
the woman whom John believes Peter met yesterday smokes

Conclusion

We have defined a formal framework for the analysis of the syntax and the semantics of natural languages the main characteristic of which is to ground the syntactic structure of a sentence in the notion of proof net labelled with feature matrices and which takes advantage of both Categorial Grammars and Unification Grammars.
With respect to Categorial Grammars, our formalism keeps the logical management of the syntactic resources. By using the flexibility of labelling to take other grammatical properties such as word order into account, it goes beyond the limits of the Lambek Calculus in its expressive power.
With respect to Unification Grammars, our formalism keeps the mechanism of unification between feature structures but it differs in the resource management: in Unification Grammars, this is realised inside the feature structures by means of principles of well-formedness of these structures. In our formalism, this is realised outside the feature structures by means of a proof net which expresses their mutual interactions.
The parsing mechanism that results from this formalism has the advantage of being very simple.

Acknowledgements

Thanks to François Lamarche, Marc Dymetman and the referees for their helpful comments.

References

[1] A. Abeillé. *Les nouvelles syntaxes*. Linguistique. Armand Colin, 1993.

[2] G. Barry, M. Hepple, N. Leslie, and G. Morrill. Structural operations and structural modalities. In *EACL, Berlin*, 1991.

[3] P. de Groote. A correctness criterion for intuitionistic proof nets. In *Fourth Symposium on Logical Foundations of Computer Science, Yaroslav, Russia, July 1997*, 1997.

[4] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[5] M.I. Kanovich. The multiplicative fragment of linear logic is NP-complete. ITLI Prepublication Series X-91-13, University of Amsterdam, 1991.

[6] F. Lamarche. Proof Nets for Intuitionistic Linear Logic I: Essential Nets. Preliminary report, Imperial College, April 1994.

[7] F. Lamarche. Games semantics for full propositional linear logic. In D. Kozen, editor, *Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 464–473, San Diego, California, June 1995.

[8] F. Lamarche. From proof nets to games. *Electronic Notes in Theoretical Computer Science*, 3, 1996. Special Issue of Linear Logic'96, Tokyo Meeting, March 1996.

[9] P. Lorenzen. Ein dialogisches konstruktivitaetskriterium. In *[CA] Infinitistic Methods, Proc. Symp. Foundations Math.,Warsaw 1959*, pages 193–200, 1961.

[10] M. Moortgat and G. Morrill. Heads and phrases: Type calculus for dependency and constituent structure. Manuscript, 1991.

[11] J. Merenciano and G. Morrill. Generation as deduction. In C. Retoré, editor, *Logical Aspects of Computational Linguistics, LACL'96, Nancy, September 1996*, volume 1328 of *LNCS*, pages 310–328. Springer Verlag, 1997.

[12] R. Montague. Universal grammar. *Theoria*, 36:373–398, 1970. Reprinted in R. Thomason, editor, *Formal Philosophy*, 188-221, New Haven: Yale University Press, 1974.

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

[14] M Moortgat. Labelled deductive systems for categorial theorem proving. In *Eighth Amsterdam Colloquium, ILLC, Amsterdam*, 1992.

[15] M. Moortgat. Categorial Type Logics. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 2. Elsevier, 1996.

[16] G. Morrill. Grammar and logical types. In *7th Amsterdam Colloquium*, 1990. Published as *Grammar and Logic* in Theoria, Volume LXII, Part 3, 260–293.

[17] G. Morrill. *Type Logical Grammar.* Kluwer Academic Publishers, Dordrecht, 1994.

[18] G. Morrill. Clausal proofs and discontinuity. *Journal of the IGPL*, 3(2,3):403–427, 1995.

[19] G. Morrill. Higher order linear logic programming of categorial deduction,. In *EACL, Dublin*, 1995.

[20] R. Oehrle. Term-labeled categorial type systems. *Linguistics and Philosophy*, 17:633–678, 1994.

[21] M. Pentus. Lambek grammars are context free. In *Eighth Annual IEEE Symposium on Logic in Computer Science, Montreal, Canada, June 1993*, pages 429–433. IEEE Computer Society Press, 1993.

[22] D. Roorda. *Resource Logics. Proof-Theoretical Investigations.* PhD thesis, Universiteit van Amsterdam, 1991.

Received 7 November 1997. Revised 2 June 1999