Learning Lambek Grammars from Proof Frames

Roberto Bonato[1,⋆] and Christian Retoré[2,⋆⋆]

[1] Questel SAS, Sophia Antipolis, France
[2] IRIT, Toulouse, France & Univ. Bordeaux, France

Abstract. In addition to their limpid interface with semantics, categorial grammars enjoy another important property: learnability. This was first noticed by Buszkowski and Penn and further studied by Kanazawa, for Bar-Hillel categorial grammars.

What about Lambek categorial grammars? In a previous paper we showed that product free Lambek grammars are learnable from structured sentences, the structures being incomplete natural deductions. Although these grammars were shown to be unlearnable from strings by Foret ad Le Nir, in the present paper, we show that Lambek grammars, possibly with product, are learnable from proof frames i.e. incomplete proof nets.

After a short reminder on grammatical inference à la Gold, we provide an algorithm that learns Lambek grammars with product from proof frames and we prove its convergence. We do so for 1-valued "(also known as rigid) Lambek grammars with product, since standard techniques can extend our result to k-valued grammars. Because of the correspondence between cut-free proof nets and normal natural deductions, our initial result on product free Lambek grammars can be recovered.[1]

*We are glad to dedicate the present paper to
Jim Lambek for his 90th birthday: he is the living proof that research is
an eternal learning process.*

1 Presentation

Generative grammar exhibited two characteristic properties of the syntax of human languages that distinguish them from other formal languages:

1. Sentences should be easily parsed and generated, since we speak and understand each other in real time.
2. Any human language should be easily learnable, preferably from not so many positive examples, as first language acquisition shows.

⋆ I am deeply indebted to my co-author for having taken up again after so many years our early work on learnability for k-valued Lambek grammars, extended and coherently integrated it into the framework of learnability from proof frames.

⋆⋆ Thanks to IRIT-CNRS for hosting me during my sabbatical, to the Loci ANR project for its intellectual and financial support, to C. Casadio, M. Moortgat for their encouragement and to A. Foret for her helpful remarks.

[1] At the turn of the millenium, our initial work benefited from a number of valuable discussions with Philippe Darondeau. We are very sorry to learn of his premature passing. Adieu, Philippe.

C. Casadio et al. (Eds.): Lambek Festschrift, LNCS 8222, pp. 108–135, 2014.

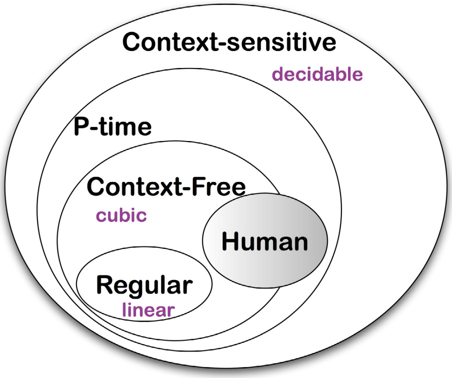

Fig. 1. Human languages and the classes of the Chomsky hierarchy (with parsing complexity)

Formally, the first point did receive a lot of attention, leading to the class of mildly context sensitive languages [20]: they enjoy polynomial parsing but are rich enough to describe natural language syntax. A formal account of learnability was harder to find. Furthermore, as soon as a notion of formal learnability was proposed, the first results seemed so negative that the learnability criterion was left out of the design of syntactical formalisms. This negative result stated that whenever a class of languages contains all the regular languages it cannot be learnt.

By that time, languages were viewed through the Chomsky hierarchy (see figure 1) and given that regular languages are the simplest and that human languages were known to include non regular languages, an algorithm that learns the syntax of a human language from positive examples was considered as impossible. This pessimistic viewpoint was erroneous for at least two reasons:

- The class of human languages does not include all regular languages and it is likely that it does to even include a single regular language, see on figure 1 the present hypothesis on human languages.
- The positive examples were thought to be sequences of words, while it has been shown long ago that grammatical rules operate on *structured* sentences and phrases (that are rather trees or graphs), see e.g. [8] for a recent account.

Gold's notion of *learning a class of languages* generated by a class of grammars $\mathcal{G}$ — that we shall recall in the first section of the present paper — is that a learning function ϕ maps a sequence of sentences $e_1, \ldots, e_n$ to a grammar $G_n = \phi(e_1, \ldots, e_n)$ in the class in such a way that, when the examples enumerate a language $\mathcal{L}(G)$ in the class $\mathcal{G}$, there exists an integer N such that if $n > N$ the G_n are constantly equal to G_N generating the same language i.e. $\mathcal{L}(G_n) = \mathcal{L}(G)$. The fundamental point is that the function learns a *class* of languages: the algorithm eventually finds out that the enumerated language cannot be any other language in the class. Therefore the very same language can be *learnable* as a member of a learnable class of languages, and *unlearnable* as the member of another class of languages. Although surprising at first sight, this notion according to which one *learns in a predefined class of languages* is rather compatible with our knowledge of first language acquisition.

Overtaking the pessimistic view of Gold's theorem, Angluin established in the 80s that some large but transversal classes of languages were learnable in Gold's sense. [4] Regarding categorial grammars, Buszkowski and Penn defined in late 80s [12,11] an algorithm that learns basic categorical grammars from structured sentences, functor-argument structures, and Kanazawa proved in 1994 that their algorithm converges: it actually learns categorial grammar in Gold's sense. [22,21]

The result in the present paper is much in the same vein as Buszkowski, Penn and Kanazawa.

Section 2. We first recall the Gold learning paradigm, identification in the limit from positive examples.

Sections 3, 4. Next we briefly present Lambek categorial grammars, and define the parsing of Lambek categorial grammar with product as cut-free proof nets construction and introduce the *proof frames*, that will be the structures we shall learn from. Informally, proof frames are name free parse structures, just like functor argument structures that are commonly used for learning basic categorial grammars. Such grammars ought to be learnt from *structured sentences* since Foret and Le Nir established that they cannot be learnt from strings [14].

Sections 5,6,7. After a reminder on unification and categorial grammars, we present our algorithm that learns rigid Lambek categorial grammars with product from proof frames and perform it on sample data involving introduction rules that are not in basic categorial grammars and product rules that are not in Lambek grammars. We then prove the convergence of this algorithm.

Section 8. We show that the present result strictly encompasses our initial result [10] that learns rigid product-free Lambek grammars from name-free natural deductions. To do so, we give the bijective correspondence between *cut-free* proof nets for the product-free Lambek calculus and *normal* natural deduction that are commonly used as parse structures.

In the conclusion, we discuss the merits and limits of the present work. We briefly explain how it can generalise to k-valued Lambek grammars with product and suggest direction for obtaining corpora with proof frame annotations from dependency-annotated corpora.

2 Exact Learning à la Gold: A Brief Reminder

We shall just give a brief overview of the Gold learning model of [17], with some comments, and explain why his famous unlearnability theorem of [17] (theorem 1 below) is not as negative as it may seem — as [4] or the present article shows.

The principles of first language acquisition as advocated by Chomsky [31] and more recently by Pinker [32,33] can be very roughly summarised as follows:

1. One *learns from positive examples only*: an argument says that in certain civilisations children uttering ungrammatical sentences are never corrected although they learn the grammar just as fast as ours — this can be discussed, since the absence of reaction might be considered as negative evidence, as well as the absence of some sentences in the input.
2. The target language is reached by *specialisation* more precisely by restricting word order from languages with a freer word order: rare are the learning algorithms for natural language that proceed by specialisation although, when starting from semantics, there are some, like the work of Tellier [39]
3. *Root meaning* is known first, hence the argumental structure or valencies are correct before the grammar learning process starts. This implies that all needed words are there, possibly in a non correct order, hence enforcing the idea of learning by specialisation — the afore mentioned work by Tellier proceeds from argument structures [39]
4. The examples that the child is exposed to are not so many: this is known as the *Poverty Of Stimulus* argument. It has been widely discussed since 2000 in particular for supporting quantitative methods. [31,34,35,8]

In his seminal 1967 paper, Gold introduced a formalisation of the process of the acquisition of one's first language grammar, which follows the first principle stated above, which is the easiest to formalise: the formal question he addressed could be more generally stated as *grammatical inference from positive examples.* It also should be said that this notion of learning may be used for other purposes every time one wants to extract some regularity out of sequences observations other fields being genomics (what would be a grammar of strings issued from ADN sequences) and diagnosis (what are the regular behaviours of system, what would be a grammar generating the sequences of normal observations provided by captors for detecting abnormal behaviours).

We shall provide only a minimum of information on formal languages and grammars. Let us just say that a language is a subset of inductive class $\mathcal{U}$. Elements of $\mathcal{U}$ usually are finite sequences (a.k.a. strings) of words, trees whose leaves are labelled by words, or graphs whose vertices are words — we here say "words" because they are linguistic words, while other say "letters" or "terminals," and we say "sentences" for sequences of words where others say "words" for sequences of "letters" or "terminals". A grammar G is a process generating the objects of a language $\mathcal{L}(G) \subset \mathcal{U}$. The membership question is said to be decidable for a grammar G when the characteristic function of $L(G)$ in U is computable. The most standard example of U is Σ^* the set of finite sequences over some set of symbols (e.g. words) Σ. The phrase structure grammars of Chomsky-Schutzenberger are the most famous grammars producing languages that are parts of Σ^*. Lambek categorial grammars and basic categorial grammars are an alternative way to generate sentences as elements of Σ^*: they produce the same languages as context–free languages [7,30,26, chapters 2, 3]. Finite labeled trees also are a possible class of object. A regular tree grammar produces such a tree language, and the yields of the trees in $\mathcal{L}(G)$ define a context free string language. In the formal study of human languages, $\mathcal{U}$ usually consists in strings of words or in trees.

Definition 1 (Gold, 1967, [17]). *A* learning function *for a class of grammars $\mathcal{G}$ producing $\mathcal{U}$-objects ($\mathcal{L}(G) \subset \mathcal{U}$) is a partial function ϕ that maps any finite sequence of positive examples $e_1, e_2, \ldots, e_k$ with $e_i \in \mathcal{U}$ to a grammar in the class $\phi(e_1, e_2, \ldots, e_k) \in \mathcal{G}$ such that:*

if *$(e_i)_{i \in I}$ is any enumeration of a language $\mathcal{L}(G) \subset \mathcal{U}$ with $G \in \mathcal{G}$,*
then *there exists an integer N such that, calling $G_i = \phi(e_1, \ldots, e_i)$:*

- *$G_P = G_N$ for all $P \geq N$.*
- *$\mathcal{L}(G_N) = \mathcal{L}(G)$.*

Several interesting properties of learning functions have been considered:

Definition 2. *A learning function ϕ is said to be*

- effective *or computable when ϕ is recursive. In this case one often speaks of a* learning algorithm*. We shall only consider effective learning functions: this is consistent both with language being viewed as a computational process and with applications to computational linguistics. Observe that the learning function does not have to be a total recursive function: it may well be undefined for some sequences of sentences and still be a learning function.*
- conservative *if $\phi(e_1, \ldots, e_p, e_{p+1}) = \phi(e_1, \ldots, e_p)$ whenever $e_{p+1} \in \mathcal{L}(\phi(e_1, \ldots, e_p))$.*
- consistent *if $\{e_1, \ldots, e_p\} \subset \mathcal{L}(\phi(e_1, \ldots, e_p))$ whenever $\phi(e_1, \ldots, e_p)$ is defined.*
- set driven *if $\phi(e_1, \ldots, e_p) = \phi(e'_1, \ldots, e'_q)$ whenever $\{e_1, \ldots, e_p\} = \{e'_1, \ldots, e'_q\}$ — neither the order of the examples nor their repetitions matters.*
- incremental *if there exists a binary function Ψ such that $\phi(e_1, \ldots, e_p, e_{p+1}) = \Psi(\phi(e_1, \ldots, e_p), e_{p+1})$*
- responsive *if the image $\phi(e_1, \ldots, e_p)$ is defined whenever there exists L in the class with $\{e_1, \ldots, e_p\} \subset L$*
- monotone increasing *when $\phi(e_1, \ldots, e_p, e_{p+1}) \subset \phi(e_1, \ldots, e_p)$*

In this paper the algorithm for learning Lambek grammars enjoys all those properties. They all seem to be sensible with respect to first language acquisition but the last one: indeed, as said above, children rather learn by specialisation.

It should be observed that the learning algorithm applies to a *class of languages*. So it is fairly possible that a given language L which both belongs to the classes $\mathcal{G}_1$ and $\mathcal{G}_2$ can be identified as a member of $\mathcal{G}_1$ and not as a member of $\mathcal{G}_2$. Learning L in such a setting is nothing more than to be sure, given the examples seen so far, that the language is not any other language in the class.

The classical result from the same 1967 paper by Gold [17] that has be over interpreted see e.g. [5,19] can be stated as follows:

Theorem 1 (Gold, 1967, [17]). *If a class $\mathcal{G}_r$ of grammars generates*

- *languages $(L_i)_i \in \mathbb{N}$ with $L_i \in \mathbb{N}$ which are strictly embedded that is $L_i \subsetneq L_{i+1}$ for all $i \in \mathbb{N}$*
- *together with the union of all these languages $\cup_{i \in \mathbb{N}} L_i \in \mathcal{G}_r$*

then no function may learn $\mathcal{G}_r$.

Proof. From the definition, we see that a learning function should have guessed the grammar of a language $\mathcal{L}(G)$ with $G \in \mathcal{G}$ after a finite number of examples in the enumeration of $\mathcal{L}(G)$. Consequently, for any enumeration of any language in the class,

(1) the learning function may only change its mind finitely many times.

Assume that is a learning function ϕ for the class $\mathcal{G}_r$. Since the L_i are nested as stated, we can provide an enumeration of $L = \cup L_i$ according to which we firstly see examples $x_0^1, \cdots, x_0^{p_0}$ from L_0 until ϕ proposes G_0 with $\mathcal{L}G_0 = L_0$, then we see examples $x_1^1, \cdots, x_1^p$ in L_1 until ϕ proposes G_1 with $\mathcal{L}G_1 = L_1$, then we see examples $x_2^1, \cdots, x_2^p$ in L_2 until ϕ proposes G_2 with $\mathcal{L}G_2 = L_2$, etc. In such an enumeration of L the learning function changes its mind infinitely many times, conflicting with (1). Thus there cannot exists a learning function for the class $\mathcal{G}_r$.

Gold's theorem above has an easy consequence that was interpreted quite negatively:

Corollary 1. *No class containing the regular languages can be learnt.*

Indeed, by that time the Chomsky hierarchy was so present that no one thought that transverse classes could be of any interest let alone learnable. Nowadays, it is assumed that the syntax of human languages contains no regular languages and goes a bit beyond context free languages as can be seen in figure 1. It does not seem likely that human languages contain a series of strictly embedded languages as well as their unions. Hence Gold's theorem does not prevent large and interesting classes of languages from being learnt. For instance Angluin showed that pattern languages, a transversal class can be learnt by identification in the limit [4] and she also provided a criterion for learnability base on telltale sets:

Theorem 2 (Angluin, 1980, [5]). *An enumerable family of languages L_i with a decidable membership problem is effectively learnable whenever for each i there is a computable finite $T_i \subset_f L_i$ such that if $T_i \subset L_j$ then there exists $w \in (L_j \setminus L_i)$*

As a proof that some interesting classes are learnable, we shall define particular grammars, Lambek categorial grammars with product, and their associated structure languages, before proving that they can be learnt from these structures, named proof frames.

3 Categorial Grammars and the LCGp Class

Given a finite set of words Σ and an inductively defined set of categories $\mathcal{C}$ including a special category s and an inductively defined set of derivable sequents $\vdash\, \subset (\mathcal{C}^* \times \mathcal{C})$ (each of them being written $t_1, \ldots, t_n \vdash t$) a categorial grammar G is defined as a map lex_G from words to finite sets of categories. An important property, as far as learnability is concerned, is the maximal number of categories per word i.e. $\max_{w \in \Sigma} |\mathrm{lex}_G(w)|$. When it is less than k, the categorial grammar G is said to be *k-valued* and 1-valued categorial grammars are said to be *rigid*.

Some standard family of categorial grammars are:

1. *Basic categorial grammars BCG* also known as AB grammars have their categories in $\mathcal{C} ::= \mathsf{s} \mid B \mid \mathcal{C} \setminus \mathcal{C} \mid \mathcal{C} / \mathcal{C}$ and the derivable sequents are the ones that are derivable in the Lambek calculus with elimination rules only $\Delta \vdash A$ and $\Gamma \vdash B / A$ (respectively $\Gamma \vdash A \setminus B$) yields $\Gamma, \Delta \vdash B$ (respectively $\Delta, \Gamma \vdash B$) — in such a setting the empty sequence is naturally prohibited even without saying so. [6]
2. The original *Lambek grammars* [23] also have their categories in the same inductive set $\mathcal{C} ::= \mathsf{s} \mid B \mid \mathcal{C} \setminus \mathcal{C} \mid \mathcal{C} / \mathcal{C}$ and the derivable sequents are the ones that are derivable in the Lambek calculus without empty antecedent, i.e. with rules of figure 3 except $\otimes_i$ and $\otimes_h$ — a variant allows empty antecedents.
3. *Lambek grammars with product* (LCGp) have their categories in $\mathcal{C}_\otimes ::= \mathsf{s} \mid B \mid \mathcal{C}_\otimes \setminus \mathcal{C}_\otimes \mid \mathcal{C}_\otimes / \mathcal{C}_\otimes \mid \mathcal{C}_\otimes \otimes \mathcal{C}_\otimes$ and the derivable sequents are the ones that are derivable in the Lambek calculus with product without empty antecedent i.e. with all the rules of figure 3 — a variant allows empty antecedents.

A phrase, that is a sequence of words $w_1 \cdots w_n$, is said to be of category C according to G when, for every i between 1 and p there exists $t_i \in \mathrm{lex}_G(w_i)$ such that $t_1, \ldots, t_n \vdash C$ is a derivable sequent. When C is s the phrase is said to be a *sentence* according to G. The string language generated by a categorial grammar is the subset of Σ^* consisting in strings that are of category s i.e. sentences. Any language generated by a grammar in one of the aforementioned classes of categorial grammars is context free.

In this paper we focus on Lambek grammars with product (LCGp). The explicit use of product categories in Lambek grammars is not so common. Category like $(a \otimes b) \setminus c$ can be viewed as $b \setminus (a \setminus c)$ so they do not really involve a product. The comma in the left-hand side of the sequent, as well as the separation between words are implicit products, but grammar and parsing can be defined without explicitly using the product. Nevertheless, there are cases when the product is appreciated.

- For analysing the French Treebank, Moot in [25] assigns the category $((np \otimes pp) \setminus (np \otimes pp)) / (np \otimes pp)$ to "*et*" ("*and*") for sentences like:
  (2) Jean donne un livre à Marie et une fleur à Anne.
- According to Glyn Morrill [28,27] past participles like *raced* should be assigned the category $((CN \setminus CN)/(N \setminus (N \setminus \mathsf{s}-)) \otimes (N \setminus (N \setminus \mathsf{s}-))$ where $\mathsf{s}-$ is an untensed sentence in sentences like:
  (3) The horse raced past the barn fell.

The derivable sequents of the Lambek syntactic calculus with product are obtained form the axiom $C \vdash C$ for any category C and the rules are given below, where A, B are categories and Γ, Δ finite sequences of categories:

$$\frac{\Gamma, B, \Gamma' \vdash C \quad \Delta \vdash A}{\Gamma, \Delta, A \setminus B, \Gamma' \vdash C} \setminus_h \qquad \frac{A, \Gamma \vdash C}{\Gamma \vdash A \setminus C} \setminus_i \quad \Gamma \neq \varnothing$$

$$\frac{\Gamma, B, \Gamma' \vdash C \quad \Delta \vdash A}{\Gamma, B / A, \Delta, \Gamma' \vdash C} /_h \qquad \frac{\Gamma, A \vdash C}{\Gamma \vdash C / A} /_i \quad \Gamma \neq \varnothing$$

$$\frac{\Gamma, A, B, \Gamma' \vdash C}{\Gamma, A \otimes B, \Gamma' \vdash C} \otimes_h \qquad \frac{\Delta \vdash A \quad \Gamma \vdash B}{\Delta, \Gamma \vdash A \otimes B} \otimes_i$$

Fig. 2. Sequent calculus rule for the Lambek calculus

4 Categorial Grammars Generating Proof Frames

The classes of languages that we wish to learn include some proper context free languages [7], hence they might be difficult to learn. So we shall learn them from *structured sentences*, and this section is devoted to present the proof frames that we shall use as structured sentences.

A neat natural deduction system for Lambek calculus with product is rather intricate [3,1], mainly because the product elimination rules have to be carefully commuted for having a unique normal form. Cut-free sequent calculus proofs are also not so good structures because they are quite redundant and some of their rules can be swapped. As explained in [26, chapter 6] proof nets provide perfect parse structure for Lambek grammars even if they use the product. When the product is not used, cut-free proof nets and normal natural deduction are isomorphic, as we shall show in subsection 8.1. Consequently the structures that we used for learning will be proof frames that are proof nets with missing informations. Let us see how categorial grammars generate such structures, and first let us recall the correspondence between polarised formulae of linear logic and Lambek categories.

4.1 Polarised Linear Formulae and Lambek Categories

A Lambek grammar is better described with the usual Lambek categories, while proof nets are better described with linear logic formulae. Hence we need to recall the correspondence between these two languages as done in [26, chapter 6]. Lambek categories (with product) are $\mathcal{C}_\otimes$ defined in the previous section 3. Linear formula L are defined by:

$$\mathsf{L} \ ::= \ \mathsf{P} \ \big| \ \mathsf{P}^\perp \ \big| \ (\mathsf{L} \otimes \mathsf{L}) \ \big| \ (\mathsf{L} \wp \mathsf{L})$$

the negation of linear logic $\_)^\perp$ is only used on propositional variables from P as the De Morgan laws allow:

$$(A^\perp)^\perp \equiv A \qquad (A \wp B)^\perp \equiv (B^\perp \otimes A^\perp) \qquad (A \otimes B)^\perp \equiv (B^\perp \wp A^\perp)$$

To translate Lambek categories into linear logic formulae, one has to distinguish the polarised formulae, the output or positive ones L° and the input or negative ones from $\mathsf{L}^\bullet$ with $F \in \mathsf{L}^\circ \iff F^\perp \in \mathsf{L}^\bullet$ and $(\mathsf{L}^\circ \cup \mathsf{L}^\bullet) \subsetneq \mathsf{L}$:

$$\begin{cases} \mathsf{L}^\circ \ ::= \ \mathsf{P} \ \big| \ (\mathsf{L}^\circ \otimes \mathsf{L}^\circ) \ \big| \ (\mathsf{L}^\bullet \wp \mathsf{L}^\circ) \ \big| \ (\mathsf{L}^\circ \wp \mathsf{L}^\bullet) \\ \mathsf{L}^\bullet \ ::= \ \mathsf{P}^\perp \ \big| \ (\mathsf{L}^\bullet \wp \mathsf{L}^\bullet) \ \big| \ (\mathsf{L}^\circ \otimes \mathsf{L}^\bullet) \ \big| \ (\mathsf{L}^\bullet \otimes \mathsf{L}^\circ) \end{cases}$$

Any formula of the Lambek L calculus can be translated as an output formula $+L$ of multiplicative linear logic and its negation can be translated as an input linear logic formulae $-L$ as follows:

L	$\alpha \in \mathsf{P}$	$L = M \otimes N$	$L = M \setminus N$	$L = N / M$
$+L$	α	$+M \otimes +N$	$-M \wp +N$	$+N \wp -M$
$-L$	$\alpha^\perp$	$-N \wp -M$	$-N \otimes +M$	$+M \otimes -N$

Conversely any output formula of linear logic is the translation of a Lambek formula and any input formula of linear logic is the negation of the translation of a Lambek formula. Let $(\ldots)^\circ_{\mathsf{Lp}}$ denotes the inverse bijection of "+", from L° to Lp and $(\ldots)^\bullet_{\mathsf{Lp}}$ denotes the inverse bijection of "–" from $\mathsf{L}^\bullet$ to Lp. These two maps are inductively defined as follows:

$F \in \mathsf{L}^\circ$	$\alpha \in \mathsf{P}$	$(G \in \mathsf{L}^\circ) \otimes (H \in \mathsf{L}^\circ)$	$(G \in \mathsf{L}^\bullet) \wp (H \in \mathsf{L}^\circ)$	$(G \in \mathsf{L}^\circ) \wp (H \in \mathsf{L}^\bullet)$
F°_{Lp}	α	$G^\circ_{\mathsf{Lp}} \otimes H^\circ_{\mathsf{Lp}}$	$G^\bullet_{\mathsf{Lp}} \setminus H^\circ_{\mathsf{Lp}}$	$G^\circ_{\mathsf{Lp}} / H^\bullet_{\mathsf{Lp}}$
$F \in \mathsf{L}^\bullet$	$\alpha^\perp \in \mathsf{P}^\perp$	$(G \in \mathsf{L}^\bullet) \wp (H \in \mathsf{L}^\bullet)$	$(G \in \mathsf{L}^\circ) \otimes (H \in \mathsf{L}^\bullet)$	$(G \in \mathsf{L}^\bullet) \otimes (H \in \mathsf{L}^\circ)$
$F^\bullet_{\mathsf{Lp}}$	α	$H^\bullet_{\mathsf{Lp}} \otimes G^\bullet_{\mathsf{Lp}}$	$H^\bullet_{\mathsf{Lp}} / G^\circ_{\mathsf{Lp}}$	$H^\circ_{\mathsf{Lp}} \setminus G^\bullet_{\mathsf{Lp}}$

4.2 Proof Nets

A proof net is a graphical representation of a proof which identifies inessentially different proofs. A cut-free proof net has several conclusions, and it consists of

- the subformula trees of its conclusions, that possibly stops on a sub formula which is not necessarily a propositional variable (axioms involving complex formulae simplify the learning process).
- a cyclic order on these sub formula trees
- axioms that links two dual leaves F and $F^\perp$ of these formula subtrees.

Such a structure can be represented by a sequence of terms — admittedly easier to type than a graph — with indices for axioms. Each index appears exactly twice, once on a formula F (not necessarily a propositional variable) and one on $F^\perp$. Here are two proof nets with the same conclusions:

(4) $\mathsf{s}^{\perp 1} \otimes (\mathsf{s}^2 \wp np^{\perp 3}), np^3 \otimes (\mathsf{s}^{\perp} \otimes np)^7, (np^{\perp} \wp \mathsf{s})^7 \otimes \mathsf{s}^{\perp 2}, \mathsf{s}^1$

(5) $\mathsf{s}^{\perp 1} \otimes (\mathsf{s}^2 \wp np^{\perp 3}), np^3 \otimes (\mathsf{s}^{\perp 4} \otimes np^5), (np^{\perp 5} \wp \mathsf{s}^4) \otimes \mathsf{s}^{\perp 2}, \mathsf{s}^1$

The second one is obtained from the first one by expansing the complex axiom $(\mathsf{s}^{\perp} \otimes np)^7, (np^{\perp} \wp \mathsf{s})^7$ into two axioms: $(\mathsf{s}^{\perp 4} \otimes np^5), (np^{\perp 5} \wp \mathsf{s}^4)$. Complex axioms always can be expansed into atomic axioms — this is known as η-expansion. This is the reason why proof nets are often presented with atomic axioms. Nevertheless as we shall substitute propositional variables with complex formula during the learning process we need to consider complex axioms as well — see the processing of example (9) in section 6.

No any such structure does correspond to a proof:

Definition 3. *A proof structure with conclusions $C^1, I_1^1, \ldots, I_n^1$ is said to be a proof* net *of the Lambek calculus when it enjoys the correctness criterion defined by the following properties:*

1. Acyclic: *any cycle contains the two branches of a $\wp$ link*
2. Intuitionistic: *exactly one conclusion is an output formula of L°, all other conclusions are input formulae of $\mathsf{L}^\bullet$*
3. Non commutative: *no two axioms cross each other*
4. Without empty antecedent: *there is no sub proof net with a single conclusion*

The first point in this definition is not stated precisely but, given that we learn from correct structured sentences, we shall not need a precise definition. The reader interested in the details can read [26, chapter 6]. Some papers require a form of connectedness but it is not actually needed since this connectedness is a consequence of the first two points see [18] or [26, section 6.4.8 pages 225–227].

Definition 4. *Proof nets for the Lambek calculus can be defined inductively as follows (observe that they contain exactly one output conclusion):*

- *given an output formula F an axiom $F, F^{\perp}$ is a proof net with two conclusions F and $F^{\perp}$ — we do no require that F is a propositional variable.*
- *given a proof net π^1 with conclusions $O^1, I_1^1, \ldots, I_n^1$ and a proof net π^2 with conclusions $O^2, I_1^2, \ldots, I_p^2$ where O^1 and O^2 are the output conclusions, one can add a $\otimes$-link between a conclusion of one and a conclusion of the other, at least one of the two being an output conclusion. We thus can obtain a proof net whose conclusions are:*
  - *$O^1 \otimes I_k^2, I_{k+1}^2, \ldots, I_p^2, O^2, I_1^2, I_{k-1}^2, I_1^1, \ldots, I_n^1$ — O^2 being the output conclusion*
  - *$I_l^1 \otimes O^2, I_1^2, \ldots, I_p^2, I_{l+1}^1, \ldots, I_n^1, O^1, I_1^1, \ldots, I_{l-1}^1,$ — O^1 being the output conclusion*
  - *$O^1 \otimes O^2, I_1^2, \ldots, I_p^2, I_1^1, \ldots, I_n^1$ — $O^1 \otimes O^2$ being the output conclusion.*
- *given a proof net π^1 with conclusions $O^1, I_1^1, \ldots, I_n^1$ one can add a $\wp$ link between any two consecutive conclusions, thus obtaining a proof nets with conclusions:*

- $O^1, I_1^1, \ldots, I_i \wp I_{i+1}, \ldots, I_n^1$ — O^1 *being the output conclusion*
- $O^1 \wp I_1^1, I_2^1 \ldots, I_n^1$ — $O^1 \wp I_1^1$ *being the output conclusion*
- $I_n^1 \wp O^1, I_1^1 \ldots, I_{n-1}^1$ — $O^1 \wp I_1^1$ *being the output conclusion*

A key result is that:

Theorem 3. *The inductively defined proof nets of definition 4, i.e. proofs, exactly correspond to the proof nets defined as graphs enjoying the universal properties of the criterion 3*

A parse structure for a sentence $w^1, \ldots, w^p$ generated by a Lambek grammar G defined by a lexicon lex_G is a proof net with conclusions $(c^n)^-, \ldots, (c^1)^-, \mathsf{s}^+$ with $c^i \in \text{lex}(w^i)$. This replaces the definition of parse structure as normal natural deductions [40] which does not work well when the product is used [3,1].

4.3 Structured Sentences to Learn from: s Proof Frames

An s proof frame (sPF) is simple a parse structure of a Lambek grammar i.e. a proof net whose formula names have been erased, except the s on the output conclusion. Regarding axioms, their positive and negative tips are also kept. Such a structure is the analogous of a functor argument structure for AB grammars or of a name free normal natural deduction for Lambek grammars used in [12,11,10] and it can be defined inductively as we did in 4, or by the conditions in definition 3.

Definition 5 (Proof frames, sPF). *An* s *proof frame (sPF) is a normal proof net* π *such that:*

- *The output of* π *is labelled with the propositional constant* s — *which is necessarily the conclusion of an axiom, the input conclusion of this axiom being labelled* $\mathsf{s}^\perp$.
- *The output conclusion of any other axiom in* π *is* O *its input conclusion being* $O^\perp = I$.

Given an s *proof net* π *its associated* s *proof frame* π_f *is obtained by replacing in* π *the output of any axiom by* O *(and its dual by* $I = O^\perp$*) except the* s *that is the output of* π *itself which is left unchanged.*

A given Lambek grammar G is said to generate an sPF ρ *whenever there exists a proof net* π *generated by* G *such that* $\rho = \pi^{IO}$. *In such a case we write* $\rho \in sPF(G)$.

The sPF associated with the two proof nets 4 and 5 above are:

(6) $\mathsf{s}^{\perp 1} \otimes (O^2 \wp I^3), O^3 \otimes (I^4 \otimes O^5), (O^5 \wp O^4) \otimes I^2, \mathsf{s}$

(7) $\mathsf{s}^{\perp 1} \otimes (O^2 \wp I^3), O^3 \otimes I^7, O^7 \otimes I^2, \mathsf{s}$

5 Unification, Proof Frames and Categorial Grammars

Our learning algorithm makes a crucial use of category-unification, and this kind of technique is quite common in grammatical inference [29], so let us briefly define unification of categorial grammars.

As said in paragraph 3, a categorial grammar is defined from a lexicon that maps every word w to a finite set of categories $\mathrm{lex}_G(w)$. Categories are usually defined from a finite set B of base categories that includes a special base category s. Here we shall consider simultaneously many different categorial grammars and to do so we shall have an infinite set B whose members will be s and infinitely many category variables denoted by x, y, x_1, x_2, ..., y_1, y_2, ... In other words, $B = \{\mathsf{s}\} \cup V$, $\mathsf{s} \notin V$, V being an infinite set of category variables. The categories arising from B are defined as usual by $\mathcal{V} ::= \mathsf{s} \mid V \mid \mathcal{V} \setminus \mathcal{V} \mid \mathcal{V} / \mathcal{V} \mid \mathcal{V} \otimes \mathcal{V}$. This infinite set of base categories does not change much categorial grammars: since there are finitely many words each of them being associated with finitely many categories, the lexicon is *finite* and a given categorial grammar only makes use of finitely many base categories. Choosing an infinite language is rather important, as we shall substitute a category variable with a complex category using fresh variables, thus turning a categorial grammar into another one, and considering families of grammars over the same base categories.

A *substitution* σ is a function from categories $\mathcal{V}$ to categories $\mathcal{V}$ that is generated by a mapping σ_V of finitely many variables $x_{i_1}, \cdots, x_{i_p}$ in V to categories of $\mathcal{V}$:

$$\sigma(\mathsf{s}) = \mathsf{s}$$
$$\text{given } x \in V, \quad \sigma(x) = \begin{cases} \sigma_V(x) & \text{if } x = x_{i_k} \text{ for some } k \\ x & \text{otherwise} \end{cases}$$
$$\sigma(A \setminus B) = \sigma(A) \setminus \sigma(B)$$
$$\sigma(B / A) = \sigma(B) / \sigma(A)$$

The substitution σ is said to be a *renaming* when σ_V is a bijective mapping from V to V — otherwise stated σ_V is a permutation of the $x_{i_1}, \cdots, x_{i_p}$).

As usual, substitutions may be extended to sets of categories by stipulating $\sigma(A) = \{\sigma(a) | a \in A\}$. Observe that $\sigma(A)$ can be a singleton while A is not: $\{(a / (b \setminus c)), (a / u)\}[u \mapsto (b \setminus c] = \{a / (b \setminus c)\}$. A substitution can also be applied to a categorial grammar: $\sigma(G) = G'$ with $\mathrm{lex}_{G'}(w) = \sigma(\mathrm{lex}_G(w))$ for any word w, and observe that a substation turns a k-valued categorical grammar into a k'-valued categorial grammar with $k' \leq k$, and possibly into a rigid (or 1-valued) categorial grammar (cf. section 3).

A substitution σ on Lambek categories (defined by mapping finitely many category variables x_i to Lambek categories L_i, $x_i \mapsto L_i$) clearly defines a substitution on linear formulae σ^ℓ (by $x_i \mapsto L_i^+$), which preserves the polarities $\sigma^\ell(F)$ is positive(respectively negative) if and only if F is. Conversely, a substitution ρ on linear formulae defined by mapping variables to positive linear formulae ($x_i \mapsto F_i$) defines a substitution on Lambek categories ρ^L with the mapping $x_i \mapsto F_{\mathsf{Lp}}^\circ$. One has: $\sigma(L) = (\sigma^\ell(L+))_{\mathsf{Lp}}^\circ$ and $\rho(F) = (\rho^L(F_{\mathsf{Lp}}^\circ))+$ if $F \in \mathsf{L}^\circ$ and

$\rho(F) = (\rho^L(F^\bullet_{\mathsf{Lp}}))-$. Roughly speaking as far as we use only polarised linear formulae and substitution that preserve polarities, it does not make any difference to perform substitutions on linear formulae or on Lambek categories.

Substitution preserving polarities (or Lambek substitutions) can also be applied to proof nets: $\sigma(\pi)$ is obtained by applying the substitution to any formula in π, and they turn an s Lambek proof net into an s Lambek proof net – this is a good reason for considering axioms on complex formulae.

Proposition 1. *If σ is a substitution preserving polarities and π a proof net generated by a Lambek grammar G, then $\sigma(\pi)$ is generated by $\sigma(G)$ and $\sigma(\pi)$ have the same associated s proof frame: $\sigma(\pi)_f = \pi_f$*

Two grammars G_1 and G_2 with their categories in $\mathcal{V}$ are said to be *equal* whenever there is renaming ν such that $\nu(G_1) = G_2$.

A substitution σ is said to unify two categories A, B if one has $\sigma(A) = \sigma(B)$. A substitution is said to unify a set of categories T or to be a unifier for T if for all categories A, B in T one has $\sigma(A) = \sigma(B)$ — in other words, $\sigma(T)$ is a singleton.

A substitution σ is said to unify a categorial grammar G or to be a unifier of G whenever, for every word in the lexicon σ unifies $\mathrm{lex}_G(w)$, i.e. for any word w in the lexicon $\mathrm{lex}_{\sigma(G)}(w)$ has a unique category — in other words $\sigma(G)$ is rigid.

A unifier does not necessarily exists, but when it does, there exists a *most general unifier (mgu)* that is a unifier σ_u such for every unifier τ there exists a substitution σ_τ such that $\tau = \sigma_\tau \circ \sigma_u$. This most general unifier is unique up to renaming. This result also holds for unifiers that unify a set of categories and even for unifiers that unify a categorial grammar. [22]

Definition 6. *Let π be an s proof net whose associated sPF is π_f. If all the axioms in π but the $\mathsf{s}, \mathsf{s}^\perp$ whose s is π's main output are $\alpha_i, \alpha_i^\perp$ with $\alpha_i \neq \alpha_j$ when $i \neq j$, π is said to be a most general labelling of π_f. If π_f is the associated sPF of an s proof net π and π_v one of the most general labelling of π_f, then π_v is also said to be a most general labelling of π. The most general labelling of an s proof net is unique up to renaming.*

We have the following obvious but important property:

Proposition 2. *Let π_v is a most general labelling of an s proof net π, then there exists a substitution σ such that $\pi = \sigma(\pi_v)$.*

6 An RG-Like Algorithm for Learning Lambek Categorial Grammars from Proof Frames

Assume we are defining a *consistent* learning function from positive examples for a class of categorial grammar (see definition 2). Assume that we already mapped $e_1, \ldots, e_n$ to a grammar G_n with $e_1, \ldots, e_n \subset \mathcal{L}(G_n)$ and $e_{n+1} \notin \mathcal{L}(G_n)$. This means that for some word w in the sentence e_{n+1} no category of $\mathrm{lex}_{G_n}(w)$

The algorithm for unifying two categories C_1 and C_2 can be done by processing a finite multi-set E of potential equations on terms, until it fails or reaches a set of equations whose left hand side are variables, each of which appears in a unique such equation — a measure consisting in triple of integers ordered ensures that this algorithm always stops. This set of equations $x_i = t_i$ defines a substitution by setting $\nu(x_i) = t_i$. Initially $E = \{C_1 = C_2\}$. In the procedure below, upper case letters stand for categories, whatever they might be, x for a variable, $*$ and $\diamond$ stand for binary connectives among $\backslash, /, \otimes$. Equivalently, unifications could be performed on linear formulae, as said in the main text. The most general unifier of n categories can be performed by iterating binary unification, the resulting most general unifier does not depend on the way one proceeds.

$$
\begin{array}{rrcl}
 & E \cup \{C{=}C\} & \longrightarrow & E \\
 & E \cup \{A_1{*}B_1{=}A_2{*}B_2\} & \longrightarrow & E \cup \{A_1{=}A_2, B_1{=}B_2\} \\
 & E \cup \{C{=}x\} & \longrightarrow & E \cup \{x{=}C\} \\
\text{if } x \in Var(C) & E \cup \{x{=}C\} & \longrightarrow & \bot \\
\text{if } x \notin Var(C) \wedge x \in Var(E) & E \cup \{x{=}C\} & \longrightarrow & E[x := C] \cup \{x{=}C\} \\
\text{if } \diamond \neq * & E \cup \{A_1{*}B_1{=}A_2 \diamond B_2\} & \longrightarrow & \bot \\
 & E \cup \{\mathsf{s}{=}A_2{*}B_2\} & \longrightarrow & \bot \\
 & E \cup \{A_1{*}B_1{=}\mathsf{s}\} & \longrightarrow & \bot
\end{array}
$$

Fig. 3. The unification algorithm for unifying two categories

is able to account for the behaviour of w in e_{n+1}. A natural but misleading idea would be to say: if word w^k needs category c^k_{n+1} in example e_{n+1}, let us add c^k to $\mathrm{lex}_{G_n}(w^k)$ to define $\mathrm{lex}_{G_{n+1}}(w^k)$. Doing so for every occurrence of problematic words in e_{n+1} we will have $e_1, \ldots, e_n, e_{n+1} \subset \mathcal{L}(G_{n+1})$ and in the limit we obtained the smallest grammar G_∞ such that $\forall i\ e_1, \ldots, e_i \in \mathcal{L}G_\infty$ which should be reached at some point. Doing so, there is little hope to identify a language in the limit in Gold sense. Indeed, nothing guarantees that the process will stop, and a categorial grammar with infinitely many types for some word is not even a grammar, that is a finite description of a possibly infinite language. Thus, an important guideline for learning categorial grammars is to bound the number of categories per word. That is the reason why we introduced in section 3 the notion of *k-valued* categorial grammars, which endow every word with at most k categories, and we shall start by learning *rigid* (1-valued) categorial grammars as the k-valued case derives from the rigid case.

Our algorithm can be viewed as an extension to Lambek grammars with product of the RG algorithm (learning Rigid Grammars) introduced by Buszkowski and Penn in [11,12] initially designed for rigid AB grammars. A difference from their seminal work is that the data ones learns from were functor argument trees while here they are proof frames (or natural deduction frames when the product is not used [10], see section 8). Proof frames may seem less natural than natural deduction, but we have two good reasons for using them:

- The first one is that product is of interest for some grammatical constructions as examples 2 and 3 show while there is no fully satisfying natural deduction for Lambek calculus with product. [3,1]
- The second one is that they resemble dependency structures, since an axiom between the two conclusions corresponding to two words expresses a dependency between these two words.

To illustrate our learning algorithm we shall proceed with the three examples below, whose corresponding s proof frames are given in figure 4. As their sPF structures shows, the middle one (9) involves a positive product in the (the $I \wp I$ in the category of "*and*") and the last one (10) involves an introduction rule (the $O \wp I$ in the category of "*that*").

(8) Sophie gave a kiss to Christian

(9) Christian gave a book to Anne and a kiss to Sophie

(10) Sophie liked a book that Christian liked.

Unfortunately the use for proof nets is to use a reverse word order, for having conclusions only, and these conclusions are linear formulae, the dual of Lambek categories as explained in section 4 — in some papers by Glynn Morrill e.g. [28] the order is not reversed, but then the linear formulae and the proof net structure are less visible. One solution that will make the supporters of either notation happy is to write the sentences vertically as we do in figure 4.

Definition 7 (RG like algorithm for sPF). *Let* $D = (\pi_f^k)_{1\leq k\leq n}$ *be the* s *proof frames associated with the examples* $(e_f^k)1 \leq k \leq n$*, and let* (π^k) *be most general labelings of the* $(\pi_f^k)_{1\leq k\leq n}$*. We can assume that they have no common variables — this is possible because the set of variables is infinite and because most general labelings are defined up to renaming. If example* e^k *contains* n *words* $w_1^k, \ldots, w_n^k$ *then* π^k *has* n *conclusions* $(c_n^k)-, \ldots, (w_1^k)-, \mathsf{s}$*, where all the* c_i^k *are Lambek categories.*

Let $GF(D)$ *be the non necessarily rigid grammar defined by the assignments* $w_i^k : c_i^k$ *— observe that a for a given word* w *there may exist several* i *and* k *such that* $w = w_i^k$*.*

Let $RG(D)$ *be the rigid grammar defined as the most general unifier of the categories* $\mathrm{lex}(w)$ *for each word in the lexicon when such a most general unifier exists.*

Define $\phi(D)$ *as* $RG(D)$*. When unification fails, the grammar can be defined by* $\mathrm{lex}(w) = \varnothing$ *for those words whose categories do not unify.*

With the sPF of our examples in sPF yields the following type assignments where the variable x_n corresponds to the axiom number n in the examples, they are all different as expected — remember that s is not a category variable but a constant.

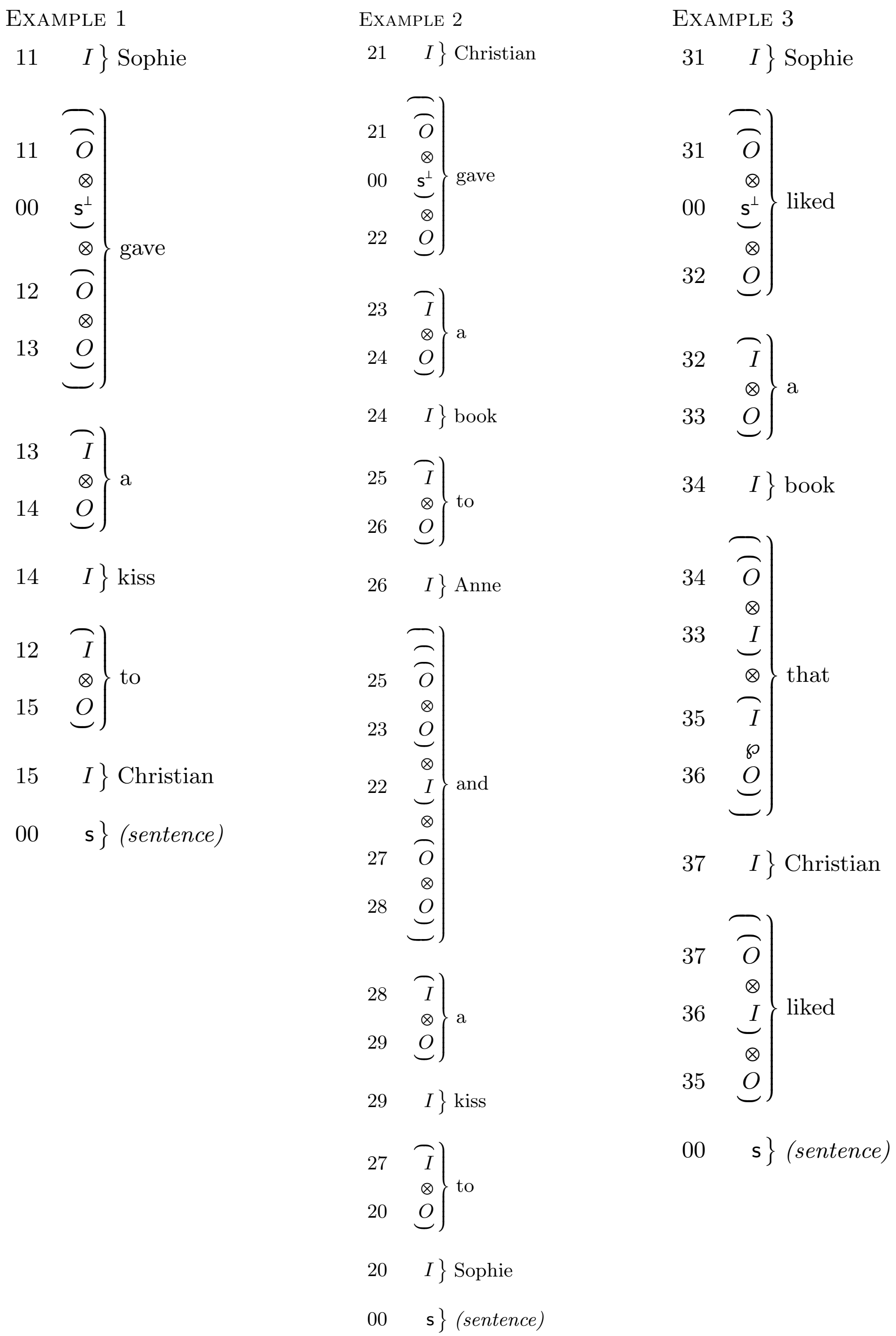

Fig. 4. Three S proof frames: three structured sentences for our learning algorithm

word	**category (Lambek)**	**category$^\perp$ (linear logic)**
and	$(((x_{23} \otimes x_{25}) \setminus x_{22})\ldots$ $\ldots / (x_{28} \otimes x_{27}))$	$((x_{28} \otimes x_{27}) \otimes \ldots$ $\ldots(x_{22} \otimes (x_{23} \otimes x_{25})))$
that	$((x_{34} \setminus x_{33}) / (x_{36} / x_{35}))$	$((x_{36} \wp x_{35}^\perp) \otimes (x_{33}^\perp \otimes x_{34}))$
liked	$(x_{31} \setminus \mathsf{s}) / x_{32}$	$x_{32} \otimes (\mathsf{s} \otimes x_{31})$
	$(x_{37} \setminus x_{36}) / x_{35}$	$x_{35} \otimes (x_{36} \otimes x_{37})$
gave	$((x_{11} \setminus \mathsf{s}) / (x_{13} \otimes x_{12}))$	$(x_{13} \otimes x_{12}) \otimes (\mathsf{s} \otimes x_{11})$
	$((x_{21} \setminus \mathsf{s}) / x_{22})$	$x_{22} \otimes (\mathsf{s} \otimes x_{21})$
to	x_{12} / x_{15}	$x_{15} \otimes x_{12}^\perp$
	x_{25} / x_{26}	$x_{26} \otimes x_{25}^\perp$
	x_{27} / x_{20}	$x_{20} \otimes x_{27}^\perp$
a	x_{13} / x_{14}	$x_{14} \otimes x_{13}^\perp$
	x_{23} / x_{24}	$x_{24} \otimes x_{23}^\perp$
	x_{28} / x_{29}	$x_{29} \otimes x_{28}^\perp$
	x_{32} / x_{33}	$x_{33} \otimes x_{32}^\perp$
Anne	x_{26}	$x_{26}^\perp$
Sophie	x_{11}	$x_{11}^\perp$
	x_{20}	$x_{20}^\perp$
	x_{31}	$x_{31}^\perp$
Christian	x_{15}	$x_{15}^\perp$
	x_{21}	$x_{21}^\perp$
	x_{37}	$x_{37}^\perp$
book	x_{24}	$x_{24}^\perp$
	x_{34}	$x_{34}^\perp$
kiss	x_{14}	$x_{14}^\perp$
	x_{29}	$x_{29}^\perp$

Unifications either performed on Lambek categories c_i^k or on the corresponding linear formulae (the $(c_i^k)-$ that appear in the second column) yield the following equations:

liked
$x_{31} = x_{37}$
$x_{36} = \mathsf{s}$
$x_{32} = x_{35}$

gave
$x_{11} = x_{21}$
$x_{22} = x_{13} \otimes x_{12}$

to
$x_{12} = x_{25} = x_{27}$
$x_{15} = x_{26} = x_{20}$

a
$x_{13} = x_{23} = x_{28} = x_{32}$
$x_{14} = x_{24} = x_{29} = x_{33}$

Sophie
$x_{11} = x_{20} = x_{31}$

Christian
$x_{15} = x_{21} = x_{37}$

kiss
$x_{14} = x_{29}$

book
$x_{24} = x_{34}$

These unification equations can be solved by setting:

$x_{36} = \mathsf{s}$
$x_{22} = x_{13} \otimes x_{12} = np \otimes pp$
$x_{12} = x_{25} = x_{27} = pp$ prepositional phrase introduced by "*to*"
$x_{13} = x_{23} = x_{28} = x_{32} = x_{35} = np$ noun phrase
$x_{14} = x_{24} = x_{29} = x_{33} = x_{34} = cn$ common noun
$x_{11} = x_{20} = x_{31} = x_{15} = x_{21} = x_{37} = x_{15} = x_{26} = pn$ proper name

The grammar can be unified into a rigid grammar G_r , namely:

word	**category (Lambek)**	**category$^\perp$ (linearlogic)**
and	$(((np \otimes pp) \setminus (np \otimes pp) ...$ $... / (np \otimes pp))$	$((np \otimes pp) \otimes ...$ $...((np \otimes pp)^\perp \otimes (np \otimes pp)))$
that	$((n \setminus n) / (\mathsf{s} / np))$	$((\mathsf{s} \wp np^\perp) \otimes (n^\perp \otimes n))$
liked	$(pp \setminus \mathsf{s}) / np$	$np \otimes (\mathsf{s} \otimes pn)$
gave	$(pp \setminus \mathsf{s}) / (pp \otimes np))$	$(np \otimes pp) \otimes (\mathsf{s} \otimes pn)$
to	np / pn	$pn \otimes np^\perp$
a	np / cn	$cn \otimes pp^\perp$
Anne	pn	$pn^\perp$
Sophie	pn	$pn^\perp$
Christian	pn	$pn^\perp$
book	cn	$cn^\perp$
kiss	cn	$cn^\perp$

As stated in proposition 1, one easily observes that the sPF are indeed produced by the rigid grammar G_r.

Earlier on, in the definition of sPF , we allowed non atomic axioms, and we can now precisely see why: the axiom 22 could be instantiated by the single variable x_{22} but, when performing unification, it got finally instantiated with $x_{13} \otimes x_{12}$. Thus, if we would have forced axioms to always be on propositional variables, the sPF of example 2 would not have been generated by the G_r: instead, G_r would not have generated exactly the example 2 but only the sPF with the axioms $x_{13}, x_{13}^\perp$ and $x_{12}^\perp, x_{12}$ linked by an $\otimes$ link $x_{13}^\perp \otimes x_{12}$ and by a $\wp$ link $x_{12}^\perp \wp x_{13}^\perp$.

7 Convergence of the Learning Algorithm

This algorithm converges in the sense defined by Gold [17], see definition 1. The first proof of convergence of a learning algorithm for categorial grammars is the proof by Kanazawa [21] of the algorithm of Buszkowki and Penn [12] for learning rigid AB grammars from functor argument structures (name free proofs os this calculus with elimination rules only). We shall do something similar, but we learn a different class of grammars from different structures, and the proof follows [9] that is a simplification of [22].

The proof of convergence makes use of the following notions and notations:

$G \subset G'$ This reflexive relation between G and G' holds whenever every lexical category assignment $a : T$ in G is in G' as well — in particular when G' is rigid, so is G, and both grammars are identical. Note that this is just the normal subset relation for each of the words in the lexicon G': $\text{lex}_G(a) \subset \text{lex}_{G'}(a)$ for every a in the lexicon of G', with $\text{lex}_G(a)$ non-empty. Keep in mind that in what follows we will also use the subset relation symbol to signify inclusion of the generated *languages*; the intended meaning should always be clear from the context.

size of a grammar The size of a grammar is simply the sum of the sizes of the occurrences of categories in the lexicon, where the size of a category is its number of occurrences of base categories (variables or s).

$G \sqsubset G'$ This reflexive relation between G and G' holds when there exists a substitution σ such that $\sigma(G) \subset G'$ which does not identify different categories of a given word, but this is always the case when the grammar is rigid.

$\mathsf{sPF}(G)$ As said earlier, $\mathsf{sPF}(G)$ is the the set of s proof structures generated by a Lambek categorial grammar G.

$GF(D)$ Given a set D of structured examples i.e. a set of s proof frames, the grammar $GF(D)$ is define as in the examples above: it is obtained by collecting the categories of each word in the various examples of D.

$RG(D)$ Given a set of sPF D, $RG(D)$ is, whenever it exists, the rigid grammar/lexicon obtained by applying the most general unifier to $GF(D)$.

Proposition 3. *Given a grammar G, the number of grammars H such that $H \sqsubset G$ is finite.*

Proof. There are only finitely many grammars which are included in G, since G is a finite set of assignments. Whenever $\sigma(H) = K$ for some substitution σ the size of H is smaller or equal to the size of K, and, up to renaming, there are only finitely many grammars smaller than a given grammar.

By definition, if $H \sqsubset G$ then there exist $K \subset G$ and a substitution σ such that $\sigma(H) = K$. Because there are only finitely many K such that $K \subset G$, and for every K there are only finitely many H for which there could exist a substitution σ with $\sigma(H) = K$ we conclude that there are only finitely many H such that $H \sqsubset G$. □

From the definition of $\sqsubset$ and from proposition 1 one immediately has:

Proposition 4. *If $G \sqsubset G'$ then $\mathsf{sPF}(G) \subset \mathsf{sPF}(G')$.*

Proposition 5. *If $GF(D) \sqsubset G$ then $D \subset \mathsf{sPF}(G)$.*

Proof. By construction of $GF(D)$, we have $D \subset \mathsf{sPF}(GF(D))$. In addition, because of proposition 4, we have $\mathsf{sPF}(GF(D)) \subset \mathsf{sPF}(G)$. □

Proposition 6. *If $RG(D)$ exists then $D \subset \mathsf{sPF}(RG(D))$.*

Proof. By definition $RG(D) = \sigma_u(GF(D))$ where σ_u is the most general unifier of all the categories of each word. So we have $GF(D) \sqsubset RG(D)$, and applying proposition 5 with $G = RG(D)$ we obtain $D \subset \mathsf{sPF}(RG(D))$. □

Proposition 7. *If $D \subset sPF(G)$ then $GF(D) \sqsubset G$.*

Proof. By construction of $GF(D)$, there is exactly one occurrence of a given category variable x in an sPF of D categorised as done in the example. Now, viewing the same sPF as an sPF of sPF(G) at the place corresponding to x there is a category label, say T. Doing so for every category variable, we can define a substitution by $\sigma(x) = T$ for all category variables x: indeed because x occurs once, such a substitution is well defined. When this substitution is applied to $GF(D)$ it yields a grammar which only contains assignments from G — by applying the substitution to the whole sPF , it remains a well-categorised sPF , and in particular the categories on the conclusions corresponding to the words must coincide — if it is the linear formula F then the corresponding Lambek category is $F^\bullet$, see subsection 4.1. □

Proposition 8. *When $D \subset sPF(G)$ with G a rigid grammar, the grammar $RG(D)$ exists and $RG(D) \sqsubset G$.*

Proof. By proposition 7 we have $GF(D) \sqsubset G$, so there exists a substitution σ such that $\sigma(GF(D)) \subset G$.

As G is rigid, σ unifies all the categories of each word. Hence there exists a unifier of all the categories of each word, and $RG(D)$ exists.

$RG(D)$ is defined as the application of most general unifier σ_u to $GF(D)$. By the definition of a most general unifier, which works as usual even though we unify sets of categories, there exists a substitution τ such that $\sigma = \tau \circ \sigma_u$.

Hence $\tau(RG(D)) = \tau(\sigma_u(GF(D))) = \sigma(GF(D)) \subset G$;
thus $\tau(RG(D)) \subset G$, hence $RG(D) \sqsubset G$. □

Proposition 9. *If $D \subset D' \subset sPF(G)$ with G a rigid grammar then $RG(D) \sqsubset RG(D') \sqsubset G$.*

Proof. Because of proposition 8 both $RG(D)$ and $RG(D')$ exist. We have $D \subset D'$ and $D' \subset$ sPF$(RG(D'))$, so $D \subset$ sPF$(RG(D'))$; hence, by proposition 8 applied to D and $G = RG(D')$ (a rigid grammar) we have $RG(D) \sqsubset RG(D')$. □

Theorem 4. *The algorithm RG for learning rigid Lambek grammars converges in the sense of Gold.*

Proof. Take $D_i, i \in \omega$ an increasing sequence of sets of examples in sPF(G) enumerating sPF(G), in other words $\cup_{i\in\omega} D_i =$ sPF(G):

$$D_1 \subset D_2 \subset \cdots D_i \subset D_{i+1} \cdots \subset \mathsf{sPF}(G)$$

Because of proposition 8 for every $i \in \omega$ $RG(D_i)$ exists and because of proposition 9 these grammars define an increasing sequence of grammars w.r.t. $\sqsubset$ which by proposition 8 is bounded by G:

$$RG(D_1) \sqsubset RG(D_2) \sqsubset \cdots RG(D_i) \sqsubset RG(D_{i+1}) \cdots \sqsubset G$$

As they are only finitely many grammars below G w.r.t. $\sqsubset$ (proposition 3) this sequence is stationary after a certain rank, say N, that is, for all $n \geq N$ $RG(D_n) = RG(D_N)$.

Let us show that the langue generated is the one to be learnt, let us prove that $\mathsf{sPF}(RG(D_N)) = \mathsf{sPF}(G)$ by proving the two inclusions:

1. Firstly, let us prove that $\mathsf{sPF}(RG(D_N)) \supset \mathsf{sPF}(G)$ Let π_f be an sPF in $\mathsf{sPF}(G)$. Since $\cup_{i\in\omega} D_i = \mathsf{sPF}(G)$ there exists a p such that $\pi_f \in \mathsf{sPF}(D_p)$.
   - If $p < N$, because $D_p \subset D_N$, $\pi_f \in D_N$, and by proposition 6 $\pi_f \in \mathsf{sPF}(RG(D_N))$.
   - If $p \geq N$, we have $RG(D_p) = RG(D_N)$ since the sequence of grammars is stationary after N. By proposition 6 we have $D_p \subset \mathsf{sPF}(RG(D_p))$ hence $\pi_f \in \mathsf{sPF}(RG(D_N)) = \mathsf{sPF}(RG(D_p))$.

   In all cases, $\pi_f \in \mathsf{sPF}(RG(D_N))$.
2. Let us finally prove that $\mathsf{sPF}(RG(D_N)) \subset \mathsf{sPF}(G)$: Since $RG(D_N) \sqsubset G$, by proposition 4 we have $\mathsf{sPF}(RG(D_N)) \subset \mathsf{sPF}(G)$ □

This exactly shows that the algorithm proposed in section 6 converges in the sense of Gold's definition (1).

8 Learning Product Free Lambek Grammars from Natural Deduction Frames

The reader may well find that the structure of the positive examples that we learn from, sorts of proofnets are rather sophisticated structures to learn from and he could think that our learning process is a drastic simplification w.r.t. standard work using functor argument structures.

Let us first see that normal natural deductions are quite a sensible structure to learn Lambek grammars from. Tiede [40] observed that natural deductions in the Lambek calculus (be they normal or not) are plain trees, defined by two unary operators (\ and / introduction rules) and two binary operators (\ and / elimination rules), from formulae as leaves (hypotheses, cancelled or free). As opposed to the intuitionistic case, there is no need to specify which hypothesis are cancelled by the introduction rules, as they may be inferred inductively: a \ (respectively /) introduction rule cancels the left-most (respectively right-most) free hypothesis. He also observed that *normal* natural deductions should be considered as the proper parse structures, since otherwise any possible syntactic structure (a binary tree) is possible. Therefore is is natural to learn Lambek grammars from normal natural deduction frames — natural deductions from which category names have been erased but the final s. Indeed, s natural deduction frames are to Lambek categorial grammars what the functor-argument (FA) structures are to AB categorial grammars — these FA structures are the standard structures used for learning AB grammars by Buszkowski, Penn and Kanazawa [12,22].

The purpose of this section is to exhibit a one to one correspondence between cut-free proof nets of the product free Lambek calculus and normal natural deductions, thus justifying the use of proof frames for learning Lambek grammars.

When there is no product, proof frames are the same as natural deduction frames that we initially used in [10]. They generalise the standard FA structures, and when the product is used, natural deduction become quite tricky [3,1] and there are the only structures one can think about.

The correspondence between on one hand natural deduction or the isomorphic λ-terms and on the other hand, proof nets, can be traced back to [36] (for second order lambda calculus) but the the closest result is the one for linear λ-calculus [13].

8.1 Proofnets and Natural Deduction: Climbing Principal Branches

As said in section 3, the formulae of product free Lambek calculus are defined by $\mathcal{C} ::= \mathsf{s} \mid B \mid \mathcal{C} \setminus \mathcal{C} \mid \mathcal{C} / \mathcal{C}$ hence their linear counterpart are a strict subset of the polarised linear formulae of subsection 4.1:

$$\begin{cases} \mathsf{L}_h^\circ & ::= & \mathsf{P} & \mid & (\mathsf{L}_h^\bullet \wp \mathsf{L}_h^\circ) & \mid & (\mathsf{L}_h^\circ \wp \mathsf{L}_h^\bullet) \\ \mathsf{L}_h^\bullet & ::= & \mathsf{P}^\perp & \mid & (\mathsf{L}_h^\circ \otimes \mathsf{L}_h^\bullet) & \mid & (\mathsf{L}_h^\bullet \otimes \mathsf{L}_h^\circ) \end{cases}$$

Let us call these formulae the *heterogeneous* polarised positive or negative formulae. In these heterogeneous formulae the connectives $\wp$ and $\otimes$ only apply to a pair of formulae with opposite polarity. The translation from Lambek categories to linear formulae and vice versa from subsection 4.1 are the same.

One may think that a proof net corresponds to a sequent calculus proof which itself corresponds to a natural deduction: as shown in our book [26], this is correct, as far as one does not care about *cuts* — which are problematic in non commutative calculi, see e.g.[24]. As it is well known in the case of intuitionnistic logic, cut-free and normal are different notions [41], and proof net are closer to sequent calculus in some respects. If one translate inductively, rule by rule, a natural deduction into a sequent calculus or into a proof net, the elimination rule from A and $A \setminus B$ yields a cut on the $A \setminus B$ formula, written $A^\perp \wp B$ in linear logic. We shall see how this can be avoided. .

From Normal Natural Deductions to Cut-Free Proof Nets. Let us briefly recall some basic facts on natural deduction for the product free Lambek calculus, from our book [26, section 2.6 pages 33-39]. In particular we shall need the following notation a formula C, and a sequence of length p of pairs consisting of a letter ε_i (where $\varepsilon_i \in \{l, r\}$) and a formula G_i we denote by

$$C[(\varepsilon_1, G_1), \ldots, (\varepsilon_p, G_p)]$$

the formula defined as follows:

if $p = 0$ $C[] = C$
if $\varepsilon_p = l$ $C[(\varepsilon_1, G_1), \ldots, (\varepsilon_{p-1}, G_{p-1}), (\varepsilon_p, G_p)] =$
$G_p \setminus C[(\varepsilon_1, G_1), \ldots, (\varepsilon_{p-1}, G_{p-1})]$
if $\varepsilon_p = r$ $C[(\varepsilon_1, G_1), \ldots, (\varepsilon_{p-1}, G_{p-1}), (\varepsilon_p, G_p)] =$
$C[(\varepsilon_1, G_1), \ldots, (\varepsilon_{p-1}, G_{p-1})] / G_p$

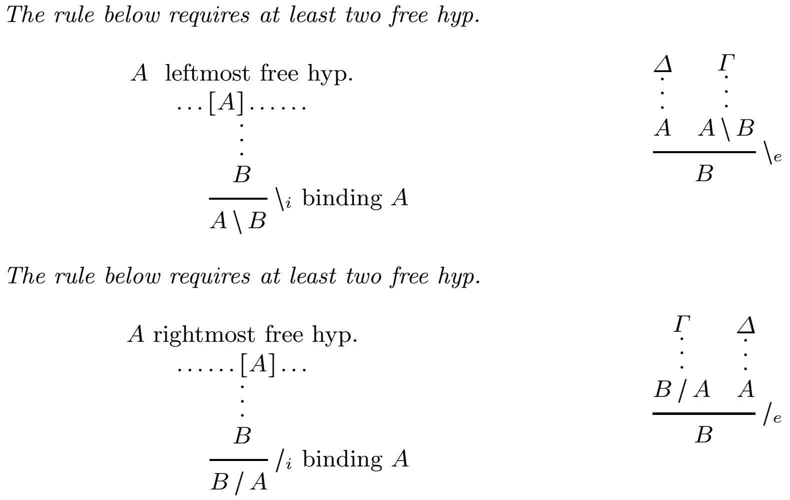

Fig. 5. Natural deduction rule for product free Lambek calculus

An important property of normal natural deductions is that whenever the last rule is an elimination rule, there is a principal branch leading from the conclusion to a free hypothesis [26, proposition 2.10 page 35] When a rule $\backslash_e$ (resp. $/_e$) is applied between a right premise $A \backslash X$ (resp. a left premise X / A) and a formula A as its left (resp. right) premise, the premise $A \backslash X$ (resp. a left premise X / A) is said to be the *principal* premise. In a proof ending with an elimination rule, a *principal branch* is a path from the root $C = X_0$ to a leaf $C[(\varepsilon_1, G_1), \ldots, (\varepsilon_p, G_p)] = X_p$ such that one has $X_i = C[(\varepsilon_1, G_1), \ldots, (\varepsilon_i, G_i)]$ and also $X_{i+1} = C[(\varepsilon_1, G_1), \ldots, (\varepsilon_{i+1}, G_{i+1})]$ and X_i is the conclusion of an elimination rule, $\backslash_e$ if $\varepsilon_{i+1} = l$ and $/_e$ if $\varepsilon_{i+1} = r$, with principal premise X_{i+1} and G_{i+1} as the other premise.

Let d be a normal natural deduction with conclusion C and hypotheses $H_1, \ldots, H_n$. It is inductively turned into a proof net with conclusions $H_n-, \ldots, H_1-, C+$ as follows (we only consider $\backslash$ because $/$ is symmetrical).

- If d is just an hypothesis A which is at the same time its conclusion the corresponding proof net is the axiom $A, A^\perp$.
- If d ends with a $\backslash$ intro, from $A, H_1, \ldots, H_n \vdash B$ to $H_1, \ldots, H_n \vdash A \backslash B$, by induction hypothesis we have a proof net with conclusions $(H_n)-, \ldots, (H_1)-, A-, B+$. The heterogeneous $\wp$ rule applies since $B+$ is heterogeneous positive and $A-$ heterogeneous negative. A $\wp$ rule yields a proof net with conclusions $(H_n)-, \ldots, (H_1)-, A - \wp B+$, and $A - \wp B+$ is precisely $(A \backslash B)+$
- The only interesting case is when d ends with an elimination rule, say $\backslash_e$. In this case there is a principal branch, say with hypothesis $C[(\varepsilon_1, G_1), \ldots, (\varepsilon_p, G_p)]$ which is applied to G_i's. Let us call $\Gamma_i = H_i^1, \ldots, H_i^{k_i}$ the hypotheses of G_i, and let d_i be the proof of G_i from Γ_i. By induction hypothesis we have a proof net π_i with conclusions $(\Gamma_i)-, (G_i)+$. Let us define the proof net π^k of conclusion $C^k- = C[(\varepsilon_1, G_1), \ldots, (\varepsilon_k, G_k)]-$, Γ_i for $i \leq k$ and $C+$ by:

- if $k = 0$ then it is an axiom $C^{\perp}, C$ (consistent with the translation of an axiom)
- otherwise π^{k+1} is obtained by a times rule between the conclusions C^k- of π^k and $G_{k+1}+$ of π_{k+1} When $\varepsilon_i = r$ then the conclusion chose the conclusion of this link to $G_{k+1} + \otimes C^k-$ that is $C^k - /G_{k+1}+ = C^{k+1}-$ and when $\varepsilon_i = l$ the conclusion is $C^k - \otimes G_{k+1}+$ that is $G_{k+1} + \backslash C^k - = C^{k+1}-$. hence, in any case the conclusions of π^{k+1} are $C^{k+1}+$ $C+$ and the Γ_i for $i \leq k+1$.

The translation of d is simply π^p, which has the proper conclusions.

As there is no cut in any of the translation steps, the result is a cut-free proof net.

From Cut-Free Proof Nets to Normal Natural Deductions. There is an algorithm that performs the reverse translation, presented for multiplicative linear logic and linear lambda terms in [13]. It strongly relies on the correctness criterion, which makes sure that everything happens as indicated during the algorithm and that it terminates. This algorithm always points at a sub formula of the proof net. Going *up* means going to an immediate sub formula, and going *down* means considering the immediate super formula.

1. Enter the proof net by its unique output conclusion.
2. Go up until you reach an axiom. Because of the polarities, during this upwards path, because of polarities, you only meet $\wp$-links, which correspond to \ and / introduction rules — λ_r and λ_l if one uses Lambek λ-terms. The hypotheses that are cancelled (the variables that are abstracted) are the ones on the non output premises — place a name on them.
3. Use the axiom link and go down with the input polarity. Hence you only meet $\otimes$ links (*) until you reach a conclusion or a $\wp$ link. In both cases, it is the head-variable of the λ-term. If it is the premise of a $\wp$-link, then it is necessarily a $\wp$ link on the path of step 2 (because of the correctness criterion) and the hypothesis of the principal branch is cancelled by \ and / introduction rules that we met during step 2 (the head variable bound by some of these λ_r or λ_l of the previous step). Otherwise it the hypothesis of the principal branch is free (the head variable is free).
4. The deductions (λ-terms) that are the arguments of the hypothesis of the principal branch (the head variable) are the ones on the output premises of the $\otimes$ links (*) that we met at step 3. They should be translated as we just did, going up from theses output formulae, starting again at step 2.

8.2 Learning Product Free Lambek Grammars from Natural Deduction

Because of the bijective correspondence between cut free product free proof nets and normal product free natural deduction we also have a correspondence between such structure without names but the S main conclusion. Hence if one wishes to it is possible to learn product free Lambek grammars from natural deduction without names but the final S, as we did in [10] Such structures are simply the generalisation to Lambek calculus of the FA structures that are commonly used for AB-grammars by [12,22].

9 Conclusion and Possible Extensions

A first criticism that can be addressed to our learning algorithm is that the rigid condition on Lambek grammars is too restrictive. One can say, as in [22] that k-valued grammars can be learned by doing all the possible unification that lead to less than k categories. Every successful unification of grammar with less than k categories should be kept, because it can thereafter work with other types, hence this approach is computationnally intractable. An alternative is to use a precise part of speech tagger and to consider word with different categories as distinct. This can be done and looks more sound and could be done partly with statistical techniques. [37,25]

The principal weakness of identification in the limit is that too much structure is required. Ideally, one would like to learn directly from strings, but in the case of Lambek grammars it has been shown to be impossible in [14]. One may think that it could be possible to try every possible structure on sentences as strings of words as done in [22] for basic categorial grammars. Unfortunately, in the case of Lambek grammars, with or without product, this cannot be done. Indeed, there can be infinitely many structures corresponding to a sentence, because a cancelled hypothesis does not have to be anchored in one the finitely many words of the sentence. Hence we ought to learn from structured sentences.

From the point of view of first language acquisition we know that some structure is available, but it is unlikely that the structured sentences are proof frames that are are partial categorial parse structure. The real input available to the learner is a mixture of prosodic and semantic information, and no one knows how to formalise these structures in order to simulate the natural data for language learning. From a computational linguistic perspective, our result is not as bad as one may think. Indeed, there exist tools that annotate corpora, and one may implement other tools that turn standard annotations into other more accurate annotations. These shallow processes may lead to structures from which one can infer the proper structure for algorithm like the one we presented in this paper. In the case of proof nets, as observed long ago, axioms express the consumption of valency. This the reason why, apart from the structure of the formulae, the structure of the proof frames is not so different from dependency annotations and can be used to infer categorial structures see e.g. [37,25]. However, the automatic acquisition of wide-coverage grammars for natural language processing applications, certainly requires a combination of machine learning techniques and of identification in the limit à la Gold, although up to now there are not so many such works.

Grammatical formalisms that can be represented in Lambek grammars can also be learnt like we did in this paper. For instance categorial version of Stabler's minimalist grammars [38] can be learnt that way as the attempts by Fulop or us show [15,10] This should be even better with the so-called Categorial Minimalist grammars of Lecomte, Amblard and us [1,2].

References

1. Amblard, M.: Calculs de représentations sémantiques et syntaxe générative: les grammaires minimalistes catégorielles. PhD thesis, Université Sciences et Technologies - Bordeaux I (September 2007)
2. Amblard, M., Lecomte, A., Retoré, C.: Categorial minimalist grammars: From generative grammar to logical form. Linguistic Analysis 36(1-4), 273–306 (2010)
3. Amblard, M., Retoré, C.: Natural deduction and normalisation for partially commutative linear logic and lambek calculus with product. In: Cooper, S.B., Kent, T.F., Löwe, B., Sorbi, A. (eds.) Computation and Logic in the Real World (Computing in Europe 2007). Quaderni del Dipartimento di Scienze Matematiche e Informatiche Roberto Magari, vol. ID487, pp. 28–35. Università degli Studi di Siena (September 2007)
4. Angluin, D.: Finding patterns common to a set of strings. Journal of Computer and Sytem Science 21(1), 46–62 (1980)
5. Angluin, D.: Inductive inference of formal languages from positive data. Information and Control 45, 117–135 (1980)
6. Bar-Hillel, Y.: A quasi arithmetical notation for syntactic description. Language 29, 47–58 (1953)
7. Bar-Hillel, Y., Gaifman, C., Shamir, E.: On categorial and phrase-structure grammars. Bulletin of the Research Council of Israel F(9), 1–16 (1963)
8. Berwick, R.C., Pietroski, P., Yankama, B., Chomsky, N.: Poverty of the stimulus revisited. Cognitive Science 35(5), 1207–1242 (2011)
9. Bonato, R.: Uno studio sull'apprendibilità delle grammatiche di Lambek rigide — A study on learnability for rigid Lambek grammars. Tesi di Laurea & Mémoire de D.E.A, Università di Verona & Université Rennes 1 (2000)
10. Bonato, R., Retoré, C.: Learning rigid Lambek grammars and minimalist grammars from structured sentences. In: Popelìnskỳ, L., Nepil, M. (eds.) Proceedings of the third workshop on Learning Language in Logic, LLL 2001. FI MU Report series, vol. FI-MU-RS-2001-08, pp. 23–34. Faculty of Informatics – Masaryk University, Strabourg (September 2001)
11. Buszkowski, W.: Discovery procedures for categorial grammars. In: van Benthem, J., Klein, E. (eds.) Categories, Polymorphism and Unification. Universiteit van Amsterdam (1987)
12. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica 49, 431–454 (1990)
13. de Groote, P., Retoré, C.: Semantic readings of proof nets. In: Kruijff, G.J., Morrill, G., Oehrle, D. (eds.) Formal Grammar, pp. 57–70. FoLLI, Prague (1996), `http://hal.archives-ouvertes.fr/hal-00823554`
14. Foret, A., Le Nir, Y.: Lambek rigid grammars are not learnable from strings. In: COLING 2002, 19th International Conference on Computational Linguistics, Taipei, Taiwan, vol. 1, pp. 274–279 (August 2002)
15. Fulop, S.: The Logic and Learning of Language. Trafford on Demand Pub. (2004)
16. Gleitman, L., Liberman, M. (eds.): An invitation to cognitive sciences, vol. 1. Language. MIT Press (1995)

17. Gold, E.M.: Language identification in the limit. Information and control 10, 447–474 (1967)
18. Guerrini, S.: A linear algorithm for mll proof net correctness and sequentialization. Theoretical Computer Science 412(20), 1958–1978 (2011)
19. Johnson, K.: Gold's theorem and cognitive science. Philosophy of Science 71, 571–592 (2004)
20. Joshi, A., Vijay-Shanker, K., Weir, D.: The convergence of mildly context-sensitive grammar formalisms. In: Sells, P., Schieber, S., Wasow, T. (eds.) Fundational Issues in Natural Language Processing. MIT Press (1991)
21. Kanazawa, M.: Learnable classes of categorial grammars. PhD thesis, Universiteit van Amsterdam (1994)
22. Kanazawa, M.: Learnable classes of categorial grammars. Studies in Logic, Language and Information. FoLLI & CSLI distributed by Cambridge University Press (1998)
23. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly, 154–170 (1958)
24. Melliès, P.A.: A topological correctness criterion for multiplicative non commutative logic. In: Ehrhard, T., Girard, J.Y., Ruet, P., Scott, P. (eds.) Linear Logic in Computer Science. London Mathematical Society Lecture Notes, vol. 316, pp. 283–321. Cambridge University press (2004)
25. Moot, R.: Semi-automated extraction of a wide-coverage type-logical grammar for French. In: Proceedings of Traitement Automatique des Langues Naturelles (TALN), Montreal (2010)
26. Moot, R., Retoré, C.: The logic of categorial grammars: A deductive account of natural language syntax and semantics. LNCS, vol. 6850. Springer, Heidelberg (2012)
27. Morrill, G.: Categorial Grammar: Logical Syntax, Semantics, and Processing. OUP, Oxford (2011)
28. Morrill, G.: Incremental processing and acceptability. Computational Linguistics 26(3), 319–338 (2000); preliminary version: UPC Report de Recerca LSI-98-46-R (1998)
29. Nicolas, J.: Grammatical inference as unification. Rapport de Recherche RR-3632. INRIA (1999)
30. Pentus, M.: Lambek grammars are context-free. In: Logic in Computer Science. IEEE Computer Society Press (1993)
31. Piattelli-Palmarini, M. (ed.): Théories du langage, théories de l'apprentissage — le débat Chomsky Piaget. Editions du Seuil. Number 138 in Points (1975)
32. Pinker, S.: Language acquisition. In: [16], ch. 6, pp. 135–182
33. Pinker, S.: Why the child holded the baby rabbits. In: [16], ch. 5, pp. 107–133
34. Pullum, G.K., Scholz, B.C.: Empirical assessment of stimulus poverty arguments. The Linguistic Review 19, 9–50 (2002)
35. Reali, F., Christiansen, M.H.: Uncovering the richness of the stimulus: Structure dependence and indirect statistical evidence. Cognitive Science 29(6), 1007–1028 (2005)
36. Retoré, C.: Le système F en logique linéaire. Mémoire de D.E.A. (dir.: J.-Y. Girard), Université Paris 7 (1987)
37. Sandillon-Rezer, N.-F., Moot, R.: Using tree transducers for grammatical inference. In: Pogodalla, S., Prost, J.-P. (eds.) LACL 2011. LNCS (LNAI), vol. 6736, pp. 235–250. Springer, Heidelberg (2011)
38. Stabler, E.: Derivational minimalism. In: Retoré, C. (ed.) LACL 1996. LNCS (LNAI), vol. 1328, pp. 68–95. Springer, Heidelberg (1997)

39. Tellier, I.: How to split recursive automata. In: Clark, A., Coste, F., Miclet, L. (eds.) ICGI 2008. LNCS (LNAI), vol. 5278, pp. 200–212. Springer, Heidelberg (2008)
40. Tiede, H.J.: Deductive Systems and Grammars: Proofs as Grammatical Structures. PhD thesis, Illinois Wesleyan University (1999), http://www.iwu.edu/htiede/
41. Zucker, J.: The correspondence between cut-elimination and normalisation i, ii. Annals of Mathematical Logic 7, 1–156 (1974)