A Study on Learnability for Rigid Lambek Grammars

Roberto Bonato

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Study on Learnability for Rigid Lambek Grammars

Roberto Bonato

N° ????

Juin 2006

Thème SYM

ISSN 0249-6399

A Study on Learnability for Rigid Lambek Grammars

Roberto Bonato

Thème SYM — Systèmes symboliques
Projet SIGNES

Rapport de recherche n° ???? — Juin 2006 — 83 pages

Abstract: We present basic notions of Gold's *learnability in the limit* paradigm, first presented in 1967, a formalization of the cognitive process by which a native speaker gets to grasp the underlying grammar of his/her own native language by being exposed to well formed sentences generated by that grammar. Then we present Lambek grammars, a formalism issued from categorial grammars which, although not as expressive as needed for a full formalization of natural languages, is particularly suited to easily implement a natural interface between syntax and semantics. In hte last part of this work, we present a learnability result for Rigid Lambek grammars from structured examples.

Key-words: Formal Learning Theory, machine learning, Lambek calculus, computational linguistics, formal grammars

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,

Une étude sur l'apprenabilité des Grammaires de Lambek Rigides

Résumé : On présente les notions basiques du paradigme d'*apprenabilité à la limite* pour une classe de grammaires formelles defini par Gold en 1967, comme possible formalisation du processus cognitif qui permet l'apprentissage d'une langue naturelle à partir d'exemples d'énoncés bien formés. Ensuite, nous presentons les grammaires de Lambek, un formalisme issu des grammaires catégorielles que, bien que encore insuffisant à rendre compte de nombre de phenomenes linguistiques, a des qualités intéréssantes par rapport à l'interface syntaxe-sémantique. Enfin, nous présentons un résultat d'apprenabilité pour les grammaires de Lambek Rigides dans le modèle d'apprentissage de Gold à partir d'exemples structurés.

Mots-clés : Théorie formelle de l'apprentissage, apprentissage automatique, calcul de Lambek, linguistique computationnelle, grammaires formelles

1 Introduction

> *How comes it that human beings, whose contacts with the world are brief and personal and limited, are nevertheless able to know as much as they do know?*
>
> Sir Bertrand Russell (citato da Noam Chomsky in [Cho75]).

Formal Learning Theory was first defined in an article by E. M. Gold in 1967 (see [Gol67]) as a first effort to provide a rigurous formalization of grammatical inference, that is the process by which a learner, presented with a certain given subset of well-formed sentences of a given language, gets to infer the grammar that generates it. The typical example of such a process is given by a child whi gets to master, in a completely spontaneous way and on the basis of the relatively small amount of information provided by sentences uttered in its cultural environment, the higly complex and subtle rules of her mother tongue, to the point that she can utter correct and *original* sentences before her third year of life. In [OWdJM97] such a formal framework is used in the broder context of the mathematical formalization of any kind of inductive reasoning. In this case the learner is "the scientist" who, on the basis of finite amount of empirical evidences provided by natural phenomena, formulates scientific hypotheses would could intensionally accunt for them.

After an initial skepticism about the grammars that could be actually learnt in Gold's paradigm (a skepticism shared and in a way enouraged by Gold himself, who proves the non-learnability in its model of the four classes of grammars of Chomsky's hierarchy), recently there has been a renewal of interest toward this computational model of learning. One of the most recent results is Shinohara's (see [Shi90]), who proves that as soon as we bound the number of rules in a context-sensitive grammar, it becomes learnable in Gold's paradigm.

Lambek Grammars have recently known a renewed interest as a mathematical tool for the description of certain linguistics phenomena, after having being long neglected after their first definition in [Lam58]. Van Benthem was among the first who stressed the singular correspondence between Montague Semantics (see [Mon97]) and the notion of structure associated to a sentence of a Lambek grammar. In particular, a recent work by Hans-Jorg Tiede (see [Tie99]) has made clearer the notion of structure of a sentence in a Lambek grammar, in contrast with a previsous definition given by Buszkowski (see [Bus86]). In doing so, he gets to prove a meaningful result about Lambek Grammars, that is that the class of tree languages generated by Lambek grammars strictly contains the class of tree languages generated by context-free grammars.

Section 2 introduces the basic notions of Learning Theory by Gold and provides a short review of most important known fact and results about it. Section 3 is a short introduction fo Lambek Grammars: we give their definition and we present the features which make them attractive from a computational linguistics point of view. Section 4 briefly presents the class of *rigid* Lambek Grammars, which is the object of our lerning algorithm, along with some basic properties and open questions. In Section 5 we present a learning algorithm for rigid

Lambek grammars from a *structured input*: the algorithm takes as its input a finite set of what has been defined in chapter 3 as *proof tree structures*. It is proved convergence for the algorithm and so the lernability for the class of rigid Lambek grammars.

2 Grammatical Inference

2.1 Child's First Language Acquisition

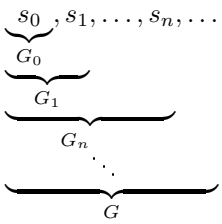
One of the most challenging goals for modern cognitive sciences is providing a sound theory accounting for the process by which any human being gets to master the highly complex and articulated grammatical structure of her mother tongue in a relatively small amount of time. Between the age of 3 and 5 we witness in children a *linguistic explosion*, at the end of which we can say that the child masters all the grammatical rules of her mother tongue, and subsequent learning is not but lexicon acquisition. Moreover, cognitive psychologists agree (see [OGL95]) in stating that the learning process is almost completely based on *positive* evidence provided by the cultural environment wherein the child is grown up: that is, correct statements belonging to her mother tongue. *Negative* evidence (any information or feedback given to the child to identify not-well-formed sentences), is almost completely absent and, in any case, doesn't seem to play any significant role in the process of learning (see [Pin94]). Simply stated, the child acquires a language due to the exposition to correct sentences coming from her linguistic environment and not to the negative feedback she gets when she utters a wrong sentence.

Providing a formal framework wherein to inscribe such an astounding ability to extract highly articulated knowledge (i.e. the grammar of a human language) from a relatively small amount of "raw" data (i.e. the statements of the language the child is exposed to during her early childhood) was one of the major forces that led to the the definition of a formal learning theory as the one we are going to describe in the following sections.

2.2 Gold's Model

The process of a child's first language acquisition can be seen as an instance of the more general problem of *grammatical inference*. In particular we will restrict our attention to the process of inference *from positive data only*. Simply stated, it's the process by which a learner can acquire the whole grammatical structure of a formal language on the basis of well-formed sentences belonging to the target language.

In 1967 Gold defined (see [Gol67]) the formal model for the process of grammatical inference from positive data that will be adopted in the present work. In Gold's model, grammatical inference is conceived as an *infinite* process during which a *learner* is presented with an infinite stream of sentences $s_0, s_1, \ldots, s_n \ldots$, belonging to language which has to be learnt, one sentence at a time. Each time the learner is presented with a new sentence s_i, she formulates a new hypothesis G_i on the nature of the underlying grammar that could generate the language the sentences she has seen so far belong to: since she is exposed to an infinite number of sentences, she will conjecture an infinite number of (not necessarily different) grammars $G_0, G_1, \ldots, G_n \ldots$.



Two basic assumptions are made about the stream of sentences she is presented with: (i) only grammatical sentences (i.e. belonging to the target language) appear in the stream, coherently with our commitment to the process of grammar induction *from positive data only*; (ii) every possible sentence of the language must appear in the stream (which must be therefore an *enumeration* of the elements of the language).

The learning process is considered *successful* when, from a given point onward, the grammar conjectured by the learner doesn't change anymore and it coincides with the grammar that actually generates the target language. It is important to stress the fact that one can never know at any finite stage whether the learning has been successful or not due to the infinite nature of the learning process itself: at each finite stage, no one can predict whether next sentence will change or not the current hypothesis. The goal of the theory lies in devising a successful *strategy* for making guesses, that is, one which can be proved to *converge* to the correct grammar after a finite (but unknown) amount of time (or positive evidence, which is the same in our model). Gold called this criterion of successful learning *identification in the limit*.

According to this criterion, a class of grammars is said to be *learnable* when, *for any language* generated by a grammar belonging to the class, and *for any enumeration* of its sentences, there is a learner that successfully identifies the correct grammar that generates the language. A good deal of current research on formal learning theory is devoted to identifying non-trivial classes of languages which are learnable in Gold's model or useful criterions to deduce (un)learnability for a class of languages on the basis of some structural property of the language.

As it will be made clear in the following sections, accepting this criterion for successful learning means that we are not interested in *when* the learning has taken place: in fact there's no effective way to decide if it has or not at any finite stage. Our aim is to devise effective procedures such that, *if applied to the infinite input stream of sentences*, are guaranteed to converge to the grammar we are looking for, if it exists.

3 Basic Notions

We present here a short review of (Formal) Learning Theory as described in [Kan98], whence we take the principal definitions and notation conventions.

3.1 Grammar Systems

The fist step in the formalization of the learning process is the formal definition of both the "cultural environment" wherein this process takes place and the "positive evidences" the learner is exposed to. To do this, we introduce the notion of *grammar system*.

Definition 3.1 (Grammar System) *A grammar system is a triple* $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$*, where*

- Ω *is a certain recursive set of finitary objects on which mechanical computations can be carried out;*
- $\mathcal{S}$ *is a certain recursive subset of* Σ^**, where* Σ *is a given finite alphabet;*
- L *is a function that maps elements of* Ω *to subsets of* $\mathcal{S}$*, i.e.* $\mathrm{L} : \Omega \rightarrow \wp(\mathcal{S})$.

We can think of Ω as the "hypothesis space", whence the learner takes her grammatical conjectures, according to the positive examples she has been exposed to up to a certain finite stage of the learning process. Elements of Ω are called *grammars*.

Positive examples presented to the learner belong to the set $\mathcal{S}$ (often we simply have $\mathcal{S} = \Sigma^*$); its elements are called *sentences*, while its subsets are called *languages*. As it will be made clear in the following sections, the nature of elements in $\mathcal{S}$ strongly influences the process of learning: intuitively, we can guess that the more information they bear, the easier the learning process is, if it is possible at all.

The function L maps each grammar G belonging to Ω into a subset of $\mathcal{S}$ which is designated as the *language generated by* G. That's why we often refer to L as the *naming function*. The question of whether $s \in \mathrm{L}(G)$ holds between any $s \in \mathcal{S}$ and $G \in \Omega$ is addressed to as the *universal membership problem*.

Example 3.2 *Let* Σ *be any finite alphabet and let* DFA *be the set of deterministic finite automata whose input alphabet is* Σ*. For every* $M \in DFA$*, let* $\mathrm{L}(M)$ *be the set of strings over* Σ *accepted by* M*. Then* $\langle DFA, \Sigma^*, \mathrm{L} \rangle$ *is a grammar system.*

Example 3.3 *Let* Σ *be any finite alphabet and let* $RegExpr$ *be the set of regular expressions over* Σ*. For every* $r \in RegExpr$*, let* $\mathrm{L}(r)$ *be the regular language represented by* r*. Then* $\langle RegExpr, \Sigma^*, \mathrm{L} \rangle$ *is a grammar system.*

Example 3.4 (Angluin, 1980) *Let* Σ *any finite alphabet, and let* Var *be a countably infinite set of variables, disjoint from* Σ*. A pattern over* Σ *is any element of* $(\Sigma \cup Var)^+$*: let* Pat *be the set of patterns over* Σ*. For every* $p \in Pat$*, let* $\mathrm{L}(p)$ *be the set of strings that can be obtained from* p *by uniformly replacing each variable* x *occurring in* p *by some string* $w \in \Sigma^+$*. The triple* $\langle Pat, \Sigma^+, \mathrm{L} \rangle$ *is a grammar system.*

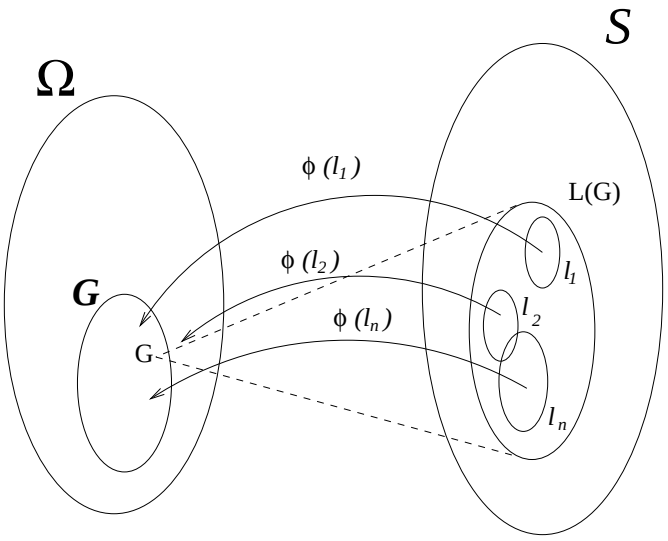


Figure 1: Grammatical Inference

3.2 Learning Functions, Convergence, Learnability

Once formally defined both the set of possible "guesses" the learner can make and the set of the positive examples she is exposed to, we need a formal notion for the mechanism by which the learner formulates hypotheses, on the basis finite sets of well-formed sentences of a given language, about the grammar that generates them.

Definition 3.5 (Learning Function) *Let $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$ be a grammar system. A learning function is a partial function that maps finite sets of sentences to grammars,*

$$\varphi : \bigcup_{k \geq 1} \mathcal{S}^k \rightharpoonup \Omega$$

where $\mathcal{S}^k$ denotes the set of k-ary sequences of sentences.

A learning function can be seen as a formal model of the cognitive process by which a learner conjectures that a given finite set of sentences belongs to the language generated by a certain grammar. Since it's partial, possibly the learner cannot infer any grammar from the stream of sentences she has seen so far.

According to the informal model outlined in section 2.2, in a successful learning process, we require the guesses made by the learner to remain the same from a certain point onward in the infinite process of learning. That is to say, there must be a finite stage (even if we

don't know which one) after which the grammar inferred on the basis of all the positive examples the learner has seen so far is always the same. This informal idea can be made precise by introducing the notion of *convergence* for a learning function:

Definition 3.6 (Convergence) *Let $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$ be a grammar system, φ a learning function,*

$$\langle s_i \rangle_{i \in \mathbb{N}} = \langle s_0, s_1, \ldots \rangle$$

an infinite sequence of sentences belonging to $\mathcal{S}$, and let

$$G_i = \varphi(\langle s_0, \ldots, s_i \rangle)$$

for any $i \in \mathbb{N}$ such that φ is defined on the finite sequence $\langle s_0, \ldots, s_i \rangle$. φ is said to converge to G on $\langle s_i \rangle_{i \in \mathbb{N}}$ *if there exists $n \in \mathbb{N}$ such that, for each $i \geq n$, G_i is defined and $G_i = G$ (equivalently, if $G_i = G$ for all but finitely many $i \in \mathbb{N}$).*

As we've already pointed out, one can never say exactly *if* and *when* convergence of a learning function to a certain grammar has taken place: this is due to the *infinite* nature of the process by which a learner gets to learn a given language in Gold's model. At any finite stage of the learning process there's no way to know whether the next sentence the learner will see causes the current hypothesis to change or not.

We will say that a class of grammars is *learnable* when for each language generated by its grammars there exists a learning function which converges to the correct underlying grammar on the basis of any enumeration of the sentences of the language. Formally:

Definition 3.7 (Learning $\mathcal{G}$) *Let $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$ be a grammar system, and $\mathcal{G} \subseteq \Omega$ a given set of grammars. The learning function φ is said* to learn $\mathcal{G}$ *if the following condition holds:*

- *for every language $L \in \mathrm{L}(\mathcal{G}) = \{\mathrm{L}(G) \mid G \in \mathcal{G}\}$,*
- *and for every infinite sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ that enumerates L (i.e., $\{s_i \mid i \in \mathbb{N}\} = L$)*

there exists a $G \in \mathcal{G}$ such that $\mathrm{L}(G) = L$, such that φ converges to G on $\langle s_i \rangle_{i \in \mathbb{N}}$.

So we will say that a given learning function *converges* to a single grammar, but that it *learns* a class of grammars. The learning for a single grammar, indeed, could be trivially implemented by a learning function that, for any given sequence of sentences as input, always returns that grammar.

Definition 3.8 (Learnability of a Class of Grammars) *A class $\mathcal{G}$ of grammars is called* learnable *if and only if there exists a learning function that learns $\mathcal{G}$. It is called* effectively learnable *if and only if there is a* computable *learning function that learns $\mathcal{G}$.*

Obviously effective learnability implies learnability.

Example 3.9 *Let $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$ be any grammar system and let $\mathcal{G} = \{G_0, G_1, G_2\} \subseteq \Omega$ and suppose there are elements $w_1, w_2 \in \mathcal{S}$ such that $w_1 \in \mathrm{L}(G_1) - \mathrm{L}(G_0)$ and $w_2 \in \mathrm{L}(G_2) - (\mathrm{L}(G_1) \cup \mathrm{L}(G_0))$. Then it's easy to verify that the following learning function learns $\mathcal{G}$:*

$$\varphi(\langle s_0, \ldots, s_i \rangle) = \begin{cases} G_2 & \text{if } w_2 \in \{s_0, \ldots, s_i\}, \\ G_1 & \text{if } w_1 \in \{s_0, \ldots, s_i\} \text{ and } w_2 \notin \{s_0, \ldots, s_i\}, \\ G_0 & \text{otherwise.} \end{cases}$$

Example 3.10 *Let's consider the grammar system $\langle CFG, \Sigma^*, \mathrm{L} \rangle$ of context-free grammars over the alphabet Σ. Let $\mathcal{G}$ be the subclass of CFG consisting of grammars whose rules are all of the form*

$$S \to w,$$

where $w \in \Sigma^*$. We can easily see that $\mathrm{L}(\mathcal{G})$ is exactly the class of finite languages over Σ. Let's define the learning function φ as

$$\varphi(\langle s_0, \ldots, s_i \rangle) = \langle \Sigma, \{S\}, S, P \rangle,$$

where

$$P = \{S \to s_0, \ldots, S \to s_i\}.$$

Then φ learns $\mathcal{G}$.

3.3 Structural Conditions for (Un)Learnability

One of the first important results in learnability theory presented in [Gol67] was a sufficient condition to deduce the unlearnability of a class $\mathcal{G}$ of grammars on the basis of some formal properties of the class of languages $\mathcal{L} = \mathrm{L}(\mathcal{G})$ (see theorem 3.14). We present here some structural conditions sufficient to deduce (un)learnability for a class of grammars. Such results are useful to get a deeper understanding to the general problem of learnability for a class of grammars.

3.3.1 Existence of a Limit Point

Let's define the notion of *limit point* for a class of languages:

Definition 3.11 (Limit Point) *A class $\mathcal{L}$ of languages* has a limit point *if there exists an infinite sequence $\langle L_n \rangle_{n \in \mathbb{N}}$ of languages in $\mathcal{L}$ such that*

$$L_0 \subset L_1 \subset \cdots \subset L_n \subset \cdots$$

and there exists another language $L \in \mathcal{L}$ such that

$$L = \bigcup_{n \in \mathbb{N}} L_n$$

The language L is called limit point *of $\mathcal{L}$.*

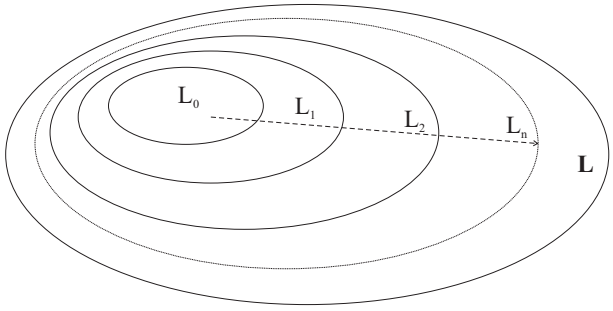


Figure 2: A limit point for a class of languages.

Lemma 3.12 (Blum and Blum's locking sequence lemma, 1975)
Suppose that a learning function φ converges on every infinite sequence that enumerates a language L. Then there is a finite sequence $\langle w_0, \ldots, w_l \rangle$ (called a locking sequence *for φ and L) with the following properties:*

(i) $\{w_0, \ldots, w_l\} \subseteq L$,

(ii) *for every finite sequence $\langle v_0, \ldots, v_m \rangle$, if $\{v_0, \ldots, v_m\} \subseteq L$, then $\varphi(\langle w_0, \ldots, w_l \rangle) = \varphi(\langle w_0, \ldots, w_l, v_0, \ldots, v_m \rangle)$.*

Intuitively enough, the previous lemma (see [BB75]) states that if a learning function converges, then there must exist a *finite* subsequence of input sentences that "locks" the guess made by the learner on the grammar the learning function converges to: that is to say, the learning function returns always the same grammar for any input stream of sentences containing that finite sequence.

The locking sequence lemma proves one of the first unlearnability criterions in Gold's learnability framework:

Theorem 3.13 *If $\mathrm{L}(\mathcal{G})$ has a limit point, then $\mathcal{G}$ is not learnable.*

An easy consequence of the previous theorem is the following

Theorem 3.14 (Gold, 1967) *For any grammar system, a class $\mathcal{G}$ of grammars is not learnable if $\mathrm{L}(\mathcal{G})$ contains all finite languages and at least one infinite language.*

Proof sketch. Let $L_1 \subset L_2 \subset \ldots$ be a sequence of finite languages and let $L_\infty = \bigcup_{i=1}^{\infty} L_i$. Suppose there were a learning function φ that learns the class $\{L \mid L \text{ is finite}\} \cup \{L_\infty\}$. Then φ must identify any finite language in a finite amount of time. But then we can build an infinite sequence of sentences that forces φ to make an infinite number of mistakes: we first present φ with enough examples from L_1 to make it guess L_1; then with enough examples from L_2 to make it guess L_2, and so on. Note that all our examples belong to L_∞.

3.3.2 (In)Finite Elasticity

As we've seen in the previous section, the existence of a limit point for a class of languages implies the existence of an "infinite ascending chain" of languages like the one described by the following, weaker condition:

Definition 3.15 (Infinite Elasticity) *A class $\mathcal{L}$ of languages is said* to have infinite elasticity *if there exists an infinite sequence $\langle s_n \rangle_{n \in \mathbb{N}}$ of sentences and an infinite sequence $\langle L_n \rangle_{n \in \mathbb{N}}$ of languages such that for every $n \in \mathbb{N}$,*

$$s_n \notin L_n,$$

and

$$\{s_0, \ldots, s_n\} \subseteq L_{n+1}.$$

The following definition, although trivial, identifies an extremely useful criterion to deduce learnability for a class of grammars:

Definition 3.16 (Finite Elasticity) *A class $\mathcal{L}$ of languages is said to have* finite elasticity *if it doesn't have infinite elasticity.*

Dana Angluin proposed in [Ang80] a characterization of the notion of learnability in a "restrictive setting" which is of paramount importance in formal learning theory. Such restrictions are about the membership problem and the recursive enumerability for the class of grammars whose learnability is at issue. Let $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$ be a grammar system and $\mathcal{G} \subseteq \Omega$ a class of grammars, let's define:

Condition 3.17 *There is an algorithm that, given $s \in \mathcal{S}$ and $G \in \mathcal{G}$, determines whether $s \in \mathrm{L}(G)$.*

Condition 3.18 *$\mathcal{G}$ is a recursively enumerable class of grammars.*

Condition 3.17 is usually referred to as *decidability for the universal membership problem*, and condition 3.18 as the *recursive enumerability* condition. Such restrictions are not unusual in concrete situations where learnability is at issue, so they don't significantly affect the usefulness of the following characterization of the notion learnability under such restrictive conditions.

Theorem 3.19 (Angluin 1980) *Let $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$ be a grammar system for which both conditions 3.17 and 3.18 hold, and let $\mathcal{G}$ be a recursively enumerable subset of Ω. Then $\mathcal{G}$ is learnable if and only if there exists a computable partial function $\psi : \Omega \times \mathbb{N} \rightharpoonup \mathcal{S}$ such that:*

(i) *for all $n \in \mathbb{N}$, $\psi(G, n)$ is defined if and only if $G \in \mathcal{G}$ and $\mathrm{L}(G) \neq \emptyset$;*

(ii) *for all $G \in \mathcal{G}, T_G = \{\psi(G, n) \mid n \in \mathbb{N}\}$ is a finite subset of $\mathrm{L}(G)$;*

(iii) *for all $G, G' \in \mathcal{G}$, if $T_G \subseteq \mathrm{L}(G')$, then $\mathrm{L}(G') \not\subset \mathrm{L}(G)$.*

Note: From this point onward, unless otherwise stated, we will restrict our attention to classes of grammars that fulfill both condition 3.17 and condition 3.18.

Angluin's theorem introduces the notion of T_G as the *tell-tale set* for a given language. Learnability in the restricted environment is characterized by the existence of a mechanism (the function ψ) to enumerate all the sentences belonging to such a *finite* subset of the target language. Even more, a tell-tale set for a given grammar G is such that if it is included in the language generated by another grammar G', then

- either $\mathrm{L}(G)$ is included in $\mathrm{L}(G')$,
- or $\mathrm{L}(G')$ contains other sentences as well as those belonging to $\mathrm{L}(G)$.

Otherwise stated, it is never the case that $T_G \subseteq \mathrm{L}(G') \subseteq \mathrm{L}(G)$. The point of the tell-tale subset is that once the strings of that subset have appeared among the sample strings, we need not fear overgeneralization in guessing a grammar G. This is because the true answer, even if it is not $\mathrm{L}(G)$, cannot be a proper subset of $\mathrm{L}(G)$. This means that a learner who has seen only the sentences belonging to the tell-tale set for a given grammar G, is justified in conjecturing G as the underlying grammar, since doing so never results in overshooting or inconsistency.

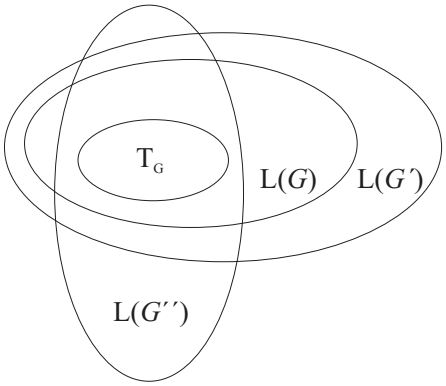


Figure 3: A tell-tale set for $\mathrm{L}(G)$.

As a consequence of Angluin's theorem, Wright proved in [Wri89] the following

Theorem 3.20 (Wright, 1989) *Let $\langle \Omega, \mathcal{S}, \mathrm{L} \rangle$ and $\mathcal{G}$ be as in theorem 3.19. If $\mathrm{L}(\mathcal{G})$ has finite elasticity, then $\mathcal{G}$ is learnable.*

In such a restricted framework, therefore, the task of proving learnability for a certain class of grammars can be reduced to the usually simpler task of proving its finite elasticity.

Due to Wright's theorem we can establish the following useful implications

$$\begin{aligned} \mathrm{L}(\mathcal{G}) \text{ has finite elasticity} &\overset{\dagger}{\Rightarrow} \mathcal{G} \text{ is learnable} \\ \mathrm{L}(\mathcal{G}) \text{ has a limit point} &\Rightarrow \mathcal{G} \text{ is } \textit{un}\text{learnable} \\ \mathcal{G} \text{ is } \textit{un}\text{learnable} &\overset{\dagger}{\Rightarrow} \mathrm{L}(\mathcal{G}) \text{ has } \textit{in}\text{finite elasticity} \end{aligned}$$

The implications indicated by $\overset{\dagger}{\Rightarrow}$ depend on the decidability of universal membership and recursive enumerability of the class of grammars at issue, as defined in conditions 3.17 and 3.18.

3.3.3 Kanazawa's Theorem

The following theorem (see [Kan98]), which is a generalization of a previous theorem by Wright, provides a sufficient condition for a class of grammars to have finite elasticity, and therefore to be learnable. A relation $R \subseteq \Sigma^* \times \Upsilon^*$ is said to be *finite-valued* if and only if for every $s \in \Sigma^*$, the set $\{u \in \Upsilon^* \mid sRu\}$ is finite.

Theorem 3.21 *Let $\mathcal{M}$ be a class of languages over Υ that has finite elasticity, and let $R \subseteq \Sigma^* \times \Upsilon^*$ be a finite-valued relation. Then $\mathcal{L} = \{R^{-1}[M] \mid M \in \mathcal{M}\}$ also has finite elasticity.*

This theorem is a powerful tool to prove finite elasticity (and therefore learnability) for classes of grammars. Once we prove the finite elasticity for a certain class of grammars in the "straight" way, we can get a proof for finite elasticity of other classes of grammars, due to the relatively loose requirements of the theorem. All we have to do is to devise a "smart" finite-valued relation between the first class and a new class of grammars such that the anti-image of the latter under this relation is the class for which we want to prove finite elasticity.

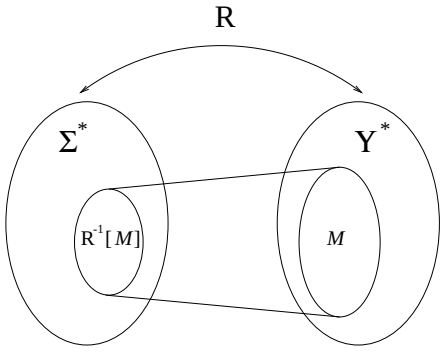


Figure 4: Kanazawa's theorem.

3.4 Constraints on Learning Functions

In the definition of learnability nothing is said about the behaviour of learning functions apart from convergence to a correct grammar. Further constraints can be imposed: one can choose a certain learning strategy. Intuitively, a strategy refers to a policy, or preference, for choosing hypotheses. Formally, a strategy can be analyzed as merely picking a subset of possible learning functions. Strategies can be grouped by numerous properties. We choose to group them by restrictiveness, defined as follows:

Definition 3.22 (Restrictiveness) *If a strategy constrains the class of learnable languages it is said to be* restrictive.

For example, strategies are grouped as computational constraints (computability, time complexity), constraints on potential conjectures (consistency), constraints on the relation between conjectures (conservatism), etc. Since the classes we will be discussing are all classes of recursive languages, "restrictive" will be taken to mean "restrictive for classes of recursive languages".

3.4.1 Non-restrictive Constraints

The proof of theorem 3.19 implies that in a grammar system where universal membership is decidable, a recursively enumerable class of grammars is learnable if and only if there is a computable learning function that learns it *order-independently*, *prudently*, and is *responsive* and *consistent* on this class.

Definition 3.23 (Order-independent Learning) *A learning function φ learns $\mathcal{G}$* order-independently *if for all $L \in \mathrm{L}(\mathcal{G})$, there exists $G \in \mathcal{G}$ such that $\mathrm{L}(G) = L$ and for all infinite sequences $\langle s_i \rangle_{i \in \mathbb{N}}$ that enumerate L, φ converges on $\langle s_i \rangle_{i \in \mathbb{N}}$ to G.*

Intuitively this seems a reasonable strategy. There does not seem to be an a priori reason why either the order of presentation should influence the final choice of hypothesis. On the other hand, it has already been proved (see [JORS99]) that in any grammar system, a class of grammars is learnable if and only if there is a computable learning function that learns it order-independently.

Definition 3.24 (Exact Learning) *A learning function φ learns $\mathcal{G}$* exactly *if for all $\mathcal{G}'$ such that φ learns $\mathcal{G}'$, $\mathrm{L}(\mathcal{G}') \subseteq \mathrm{L}(\mathcal{G})$.*

In other words, the learning function will not hypothesize grammars that are outside its class. This is not really a constraint on learning functions, but on the relation between a class of languages and a learning function. For every learning function there exists a class that it learns exactly. The reason for this constraint is the idea that children only learn languages that have at least a certain minimal expressiveness. If we want to model language learning, we want learning functions to learn a chosen class exactly. There seems to be empirical support for this idea. Some of it comes from studies of children raised in pidgin dialects, some from studies of sensory deprived children (see [Pin94]).

Definition 3.25 (Prudent Learning) *A learning function φ learns $\mathcal{G}$* prudently *if φ learns $\mathcal{G}$ and* range$(\varphi) \subseteq \mathcal{G}$.

Note that prudent learning implies exact learning. This reduces to the condition that a learning function should only produce a hypothesis if the learning function can back up its hypotheses, i.e. if the hypothesis is confirmed by the input, the learning function is able to identify the language.

Definition 3.26 (Responsive Learning) *A learning function φ is* responsive on $\mathcal{G}$ *if for any $L \in \mathrm{L}(\mathcal{G})$ and for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of elements of L ($\{\langle s_0, \ldots, s_i \rangle\} \subseteq L$), $\varphi(\langle s_0, \ldots, s_i \rangle)$ is defined.*

This constraint can be regarded as the complement of prudent learning: if all sentences found in the input are in a language in the class of languages learned, the learning function should always produce a hypothesis.

Definition 3.27 (Consistent Learning) *A learning function φ is* consistent on $\mathcal{G}$ *if for any $L \in \mathrm{L}(G)$ and for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of elements of L, either $\varphi(\langle s_0, \ldots, s_i \rangle)$ is undefined or $\{s_0, \ldots, s_i\} \subseteq \mathrm{L}(\varphi(\langle s_0, \ldots, s_i \rangle))$.*

The idea behind this constraint is that all the data given should be explained by the chosen hypothesis. It should be self-evident that this is a desirable property. Indeed, one would almost expect it to be part of the definition of learning. However, learning functions that are not consistent are not necessarily trivial. If, for example, the input is noisy, it would not be unreasonable for a learning function to ignore certain data because it considers them as unreliable. Also, it is a well known fact that children do not learn languages consistently.

3.4.2 Restrictive Constraints

Definition 3.28 (Set-Drivenness) *A learning function φ learns $\mathcal{G}$* set-driven *if $\varphi(\langle s_0, \ldots, s_i \rangle)$ is determined by $\{s_0, \ldots, s_i\}$ or, more precisely, if the following holds: whenever $\{s_0, \ldots, s_i\} = \{u_0, \ldots, u_j\}$, $\varphi(\langle s_0, \ldots, s_i \rangle)$ is defined if and only if $\varphi(\langle u_0, \ldots, u_j \rangle)$ is defined, and if they are defined, they are equal.*

It is easy to see that set-drivenness implies order-independence. Set-driven learning could be very loosely described as order-independent learning with the addition of ignoring "doubles" in the input. It is obvious that this is a nice property for a learning function to have: one would not expect the choice of hypothesis to be influenced by repeated presentation of the same data. The assumption here is that the order of presentation and the number of repetitions are essentially arbitrary, i.e. they carry no information that is of any use to the learning function. One can devise situations where this is not the case.

Definition 3.29 (Conservative Learning) *A learning function φ is* conservative *if for any finite sequence $\langle s_0, \ldots, s_i \rangle$ of sentences and for any sentence s_{i+1}, whenever $\varphi(\langle s_0, \ldots, s_i \rangle)$ is defined and $s_{i+1} \in \mathrm{L}(\varphi(\langle s_0, \ldots, s_i \rangle))$, $\varphi(\langle s_0, \ldots, s_i, s_{i+1} \rangle)$ is also defined and $\varphi(\langle s_0, \ldots, s_i \rangle) = \varphi(\langle s_0, \ldots, s_i, s_{i+1} \rangle)$.*

At first glance conservatism may seem a desirable property. Why change your hypothesis if there is no direct need for it? One could imagine cases, however, where it would not be unreasonable for a learning function to change its mind, even though the new data fits in the current hypothesis. Such a function could for example make reasonable but "wild" guesses which it could later retract. The function could "note" after a while that the inputs cover only a proper subset of its conjectured language. While such behaviour will sometimes result in temporarily overshooting, such a function could still be guaranteed to converge to the correct hypothesis in the limit.

It is a common assumption in cognitive science that human cognitive processes can be simulated by computer. This would lead one to believe that children's learning functions are computable. The corresponding strategy is the set of all partial and total recursive functions. Since this is only a subset of all possible functions, the computability strategy is a non trivial hypothesis, but not necessarily a restrictive one.

The computability constraint interacts with consistency (see [Ful88]):

Proposition 3.30 *There is a collection of languages that is identifiable by a computable learning function but by no consistent, computable learning function.*

The computability constraint also interacts with conservative learning (see [Ang80]):

Proposition 3.31 (Angluin, 1980) *There is a collection of languages that is identifiable by a computable learning function but by no conservative, computable learning function.*

Definition 3.32 (Monotonicity) *The learning function φ is* monotone increasing *if for all finite sequences $\langle s_0, \ldots, s_n \rangle$ and $\langle s_0, \ldots, s_{n+m} \rangle$, whenever $\varphi(\langle s_0, \ldots, s_n \rangle)$ and $\varphi(\langle s_0, \ldots, s_{n+m} \rangle)$ are defined,*

$$\mathrm{L}(\varphi(\langle s_0, \ldots, s_n \rangle)) \subseteq \mathrm{L}(\varphi(\langle s_0, \ldots, s_{n+m} \rangle)).$$

When a learning function that is monotone increasing changes its hypothesis, the language associated with the previous hypothesis will be (properly) included in the language associated with the new hypothesis. There seems to be little or no empirical support for such a constraint.

Definition 3.33 (Incrementality, Kanazawa 1998) *The learning function φ is* incremental *if there exists a computable function ψ such that*

$$\varphi(\langle s_0, \ldots, s_{n+1} \rangle) \simeq \psi(\varphi(\langle s_0, \ldots, s_n \rangle), s_{n+1}).$$

An incremental learning function does not need to store previous data. All it needs is current input, s_n, and its previous hypothesis. A generalized form of this constraint, called *memory limitation*, limits access for a learning function to only n previous elements of the input sequence. This seems reasonable from an empirical point of view; it seems improbable that children (unconsciously) store all utterances they encounter.

Note that, on an infinite sequence enumerating language L in $\mathrm{L}(\mathcal{G})$, a conservative learning function φ learning $\mathcal{G}$ never outputs any grammar that generates a proper superset of L.

Let φ be a conservative and computable learning function that is responsive and consistent on $\mathcal{G}$, and learns $\mathcal{G}$ prudently. Then, whenever $\{s_0, \ldots, s_n\} \subseteq L$ for some $L \in \mathrm{L}(\mathcal{G})$, $\mathrm{L}(\varphi(\langle s_0, \ldots, s_n \rangle))$ must be a minimal element of the set $\{L \in \mathrm{L}(\mathcal{G}) \mid \{s_0, \ldots, s_n\} \subseteq L\}$. This implies the following condition:

Condition 3.34 *There is a computable partial function ψ that takes any finite set D of sentences and maps it to a grammar $\psi(D) \in \mathcal{G}$ such that $\mathrm{L}(\psi(D))$ is a minimal element of $\{L \in \mathrm{L}(\mathcal{G}) \mid D \subseteq L\}$ whenever the latter set is non-empty.*

Definition 3.35 *Let ψ a computable function satisfying condition 3.34. Define a learning function φ as follows*

$$\varphi(\langle s_0 \rangle) \simeq \psi(\{s_0\}),$$
$$\varphi(\langle s_0, \ldots, s_i + 1 \rangle) \simeq \begin{cases} \varphi(\langle s_0, \ldots, s_i \rangle) & \text{if } s_{i+1} \in \mathrm{L}(\varphi(\langle s_0, \ldots, s_i \rangle)), \\ \psi(\{s_0, \ldots, s_{i+1}\}) & \text{otherwise.} \end{cases}$$

Under certain conditions the function just defined is guaranteed to learn $\mathcal{G}$, one such case is where $\mathrm{L}(\mathcal{G}$ has finite elasticity.

Proposition 3.36 *Let $\mathcal{G}$ be a class of grammars such that $\mathrm{L}(\mathcal{G})$ has finite elasticity, and a computable function ψ satisfying condition 3.34 exists. Then the learning function φ defined in definition 3.35 learns $\mathcal{G}$.*

4 Is Learning Theory Powerful Enough?

4.1 First Negative Results

One of the main and apparently discouraging consequences of the theorem 3.14 proved by Gold in the original article wherein he laid the foundations of Formal Learning Theory was that none of the four classes of Chomsky's Hierarchy is learnable under the criterion of identification in the limit. Such a first negative result has been taken for a long time as a proof that identifying languages from positive data according to his *identification in the limit* criterion was too hard a task. Gold himself looks quite pessimistic about the future of the theory he has just defined along its main directions:

> However, the results presented in the last section show that only the most trivial class of languages considered is learnable... [Gol67]

4.2 Angluin's Results

The first example of non-trivial class of learnable grammars was discovered by Dana Angluin (see [Ang80]). If *Pat* is defined like in example 3.4, we can prove that the class of all pattern languages has finite elasticity and, therefore, it is learnable. Furthermore, such a learnable class of grammars was also the first example of an interesting class of grammars that cross-cuts Chomsky Hierarchy, therefore showing that Chomsky's is not but one of many meaningful possible classifications for formal grammars.

4.3 Shinohara's Results

Initial pessimism about effective usefulness of Gold's notion of identification in the limit was definitely abandoned after an impressive result by Shinohara who proves (see [Shi90]), that *k-rigid context sensitive grammars* (context-sensitive grammars over a finite alphabet Σ with at most k rules), have finite elasticity for any k. Since the universal membership problem for context-sensitive grammars is decidable, that class of grammars is learnable. This is a particular case of his more general result about finite elasticity for what he calls *monotonic formal system.*

4.4 Kanazawa's Results

Makoto Kanazawa in [Kan98] makes another decisive step toward bridging the existing gap between Formal Learning Theory and computational linguistics. Indeed, he gets some important results on the learnability for some non-trivial subclasses of Classical Categorial Grammars (also known as AB Grammars). Analogously to what is done in [Shi90] he proves that as soon as we bound the maximum number of types a classical categorial grammar assigns to a word, we get subclasses which can be effectively learnable: in particular, he proves effective learnability for the class of *k-valued Classical Categorial Grammars*, both from structures and from strings.

In the first case, each string of the language the learner is presented to comes with additional information about the underlying structure induced by the grammar formalism that generates the language. The availability of such additional information for each string is somewhat in contrast with Gold's model of learning and gives rise to weaker results. On the other hand, psychological plausibility of the process is preserved by the fact that such an underlying structure can be seen as some kind of semantic information that could be available to the child learning the language from the very early stages of her cognitive development.

4.5 Our Results

The present work pushes Kanazawa's results a little further in the direction of proving the effective learnability for more and more powerful and expressive classes of formal languages. In particular, we will be able to prove learnability for the class of Rigid Lambek Grammars

(see chapter 9) and to show an effective algorithm to learn them on the basis of a structured input. Much is left to be done along this direction of research, since even a formal theory for Rigid Lambek Grammars is still under-developed. However, our results confirm once again that initial pessimism toward this paradigm of learning was largely unjustified, and that even quite a complex and linguistically motivated formalism like Lambek Grammars can be learnt according to it.

5 Lambek Grammars

In 1958 Joachim Lambek proposed (see [Lam58]) to extend the formalism of Classical Categorial Grammars (sometimes referred to also as Basic Categorial Grammars or BCGs) by a deductive system to derive type-change rules. A BCG is basically as a finite relation between the finite set of symbols of the alphabet (usually referred to as *words*) and a finite set of types. Combinatory properties of each word are completely determined by the shape of its types, which can be combined according to a small set of rules, fixed once and for all BCGs. Lambek's proposal marked the irruption of logics into grammars: Lambek grammars come with a whole deductive system that allows the type of a symbol to be replaced with a weaker type.

It was first realized by van Benthem (in [vB87]) that the proofs of these type changes principles carry important information about their *semantic* interpretation, following the Curry-Howard isomorphism. Thus, the notion of a proof theoretical grammar was proposed that replaces *formal grammars* (see [Cho56]) with *deductive systems* and that includes a systematic semantics for natural languages based on the relationship between proof theory and type theory. Thus, rather than considering grammatical categories as unanalyzed primitives, they are taken to be formulas constructed from atoms and connectives, and rather than defining grammars with respect to rewrite rules, grammars are defined by the rules of inference governing the connectives used in the syntactic categories.

Due to the renewed interest in categorial grammars in the field of computational linguistics, Lambek (Categorial) Grammars (LCGs) are currently considered as a promising formalism. They enjoy the relative simplicity of a tightly constrained formalism as that for BCGs, together with the linguistically attractive feature of full lexicalization.

Besides, although Pentus proved (in [Pen97]) that Lambek grammars generate exactly context-free (string) languages, in [Tie99] it has been shown that their strong generative capacity is greater than that of context-free grammars. These features make them an interesting subject for our inquiry about their properties with respect to Gold's Learnability Theory.

5.1 Classical Categorial Grammars

The main idea which lies behind the theory of Categorial Grammars is to conceive a grammar instead as a set of rules which generate any string of the language, as a system which assigns to each symbol of the alphabet a set of types which can be combined according to a small set of rules, fixed for the whole class of Classical Categorial Grammars.

A context-free grammar *á la* Chomsky is made of a set of rules that generate all the strings of a given language in a "top-down" fashion, starting from an initial symbol which identifies all the well-formed strings. On the contrary, a categorial grammar accepts a sequence of symbols of the alphabet as a well-formed string if and only if a sequence of types assigned to them *reduces* (in a "bottom-up" fashion) according to a fixed set of rules, to a distinguished type which designates well-formed strings.

Definition 5.1 (Classical Categorial Grammar)
A Classical Categorial Grammar (henceforth CCG) is a quadruple $\langle \Sigma, Pr, F, s \rangle$*, such that*

- Σ *is a finite set (the terminal symbols or vocabulary),*
- Pr *is a finite set (the non-terminal symbols or atomic categories),*
- F *is a function from* Σ *to* finite *subsets of* Tp*, where* Tp *is the smallest set such that:*
  1. $Pr \subseteq Tp$
  2. *if* $A, B \in Tp$*, then* $(A/B), (A\backslash B) \in Tp$

  If $F(a) = \{A_1, \ldots, A_n\}$ *we usually write* $G : a \mapsto A_1, \ldots, A_n$*.*
- $s \in Pr$ *is the distinguished atomic category*

In a CCG, combinatory properties are uniquely determined by their structure. There are only two modes of type combination: so-called (according to the notation introduced in [Lam58] and almost universally adopted) *Backward Application*:

$$A, A\backslash B \Rightarrow B$$

and *Forward Application*:

$$B/A, A \Rightarrow B.$$

A non-empty sequence of types $A_1, \ldots, A_n$ is said to *derive* a type B, that is

$$A_1, \ldots, A_n \Rightarrow B,$$

if repeated applications of the rules of Backward and Forward application to the sequence $A_1, \ldots, A_n$ results in B.

In order to define the language generated by a CCG we have to establish a criterion to identify a string belonging to that language. That's what is done by the following

Definition 5.2 *The binary relation*

$$\Rightarrow \subseteq Tp^* \times Tp^*$$

is defined as follows. Let $A, B \in Tp$*, let* $\alpha, \beta \in Tp^*$*,*

$$\alpha\ A\ A\backslash B\ \beta\ \Rightarrow\ \alpha\ B\ \beta$$
$$\alpha\ B/A\ A\ \beta\ \Rightarrow\ \alpha\ B\ \beta$$

The language generated by a CCG G *is the set*

$$\{a_1 \cdots a_n \in \Sigma^* \mid \textit{for } 1 \leq i \leq n,\ \exists A_i,\ G : a_i \mapsto A_i,\ \textit{and } A_1 \ldots A_n \stackrel{*}{\Rightarrow} s\}$$

where $\stackrel{*}{\Rightarrow}$ *is the reflexive, transitive closure of* $\Rightarrow$*.*

Informally, we can say that a string of symbols belongs to the language generated by a CCG if there exists a derivation of the distinguished category s out of at least one sequence of types assigned by the grammar to the symbols of the string.

Example 5.3 *The following grammar generates the language* $\{a^n b^n \mid n > 0\}$*:*

$$\begin{array}{rcl} a & : & s/B, \\ b & : & B,\ s\backslash B \end{array}$$

Here is a derivation for a^3b^3*:*

$$\begin{array}{rcl} s/B\ s/B\ \underline{s/B\ B}\ s\backslash B\ s\backslash B & \Rightarrow & s/B\ s/B\ \underline{s\ s\backslash B}\ s\backslash B\ \Rightarrow \\ s/B\ \underline{s/B\ B}\ s\backslash B & \Rightarrow & s/B\ \underline{s\ s\backslash B}\ \Rightarrow \\ \underline{s/B\ B} & \Rightarrow & s \end{array}$$

Weak generative capacity of CCGs was characterized by Gaifman (see [BH64]):

Theorem 5.4 (Gaifman, 1964) *The set of languages generated by CCGs coincides with the set of context-free languages.*

From the proof of Gaifman's theorem, we immediately obtain the following normal form theorem:

Theorem 5.5 (Gaifman normal form) *Every categorial grammar is equivalent to a categorial grammar which assigns only categories of the form*

$$A, A/B, (A/B)/C.$$

Example 5.6 *A CCG equivalent to that in example 5.3 in Gaifman normal form is the following*

$$\begin{array}{rcl} a & : & s/B,\ (s/B)/s \\ b & : & B \end{array}$$

and here is a derivation for a^3b^3*:*

$$\dfrac{\dfrac{(s/B)/B \qquad \dfrac{\dfrac{s/B \quad B}{s}}{s/B} \qquad B}{s}}{s}$$

In the previous example we make use for the first time of a "natural deduction" notation for derivations, that in the present work will substitute the cumbersome notation used in example 5.3.

5.2 Extensions of Classical Categorial Grammars

As stated in the previous section, CCG formalism comes with only two reduction rules which yield smaller types out of larger ones. Montague's work on semantics (see [Mon97]) led to the definition of two further "type-raising" rules, by which it is possible to construct new syntactic categories out of atomic ones. We can extend the definition of CCGs as presented in the previous section by adding to the former definition two new type change rules:

$$\alpha B\beta \Rightarrow \alpha(A/B)\backslash A\beta$$
$$\alpha B\beta \Rightarrow \alpha A/(B\backslash A)\beta$$

Other type-change rules that were proposed are the composition:

$$\frac{A/B \quad B/C}{A/C} \qquad \frac{C\backslash B \quad B\backslash A}{C\backslash A}$$

and the Geach Rules:

$$\frac{A/B}{(A/C)/(B/C)} \qquad \frac{B\backslash A}{(C\backslash B)\backslash(C\backslash A)}$$

We can extend the formalism of CCG by adding to definition 5.2 any type change rule we need to formalize specific phenomena in natural language. Such a *rule-based* approach was adopted by Steedman (see [Ste93]) who enriches classical categorial grammar formalism with a *finite* number of type-changes rules. On the other hand, as it will be made clear in the following section, Lambek's approach is a *deductive* one: he defines a calculus in which type changes rules spring out as a consequence of the operations performed on the types.

One could ask why we should follow the deductive rather than the rule-based approach. To begin with, as proved in [Zie89], Lambek Calculus *is not finitely axiomatizable*, that is to say that adding a finite number of type-change rules to the formalism of CCG one cannot derive all the type change rules provable in the Lambek Calculus. Moreover, the two approaches are very different under a theoretical viewpoint.

From a linguistic perspective, Steedman pointed out that there is no reason why we should stick to a deductive approach instead of to a rule based one: he underlines the importance of introducing *ad hoc* rules to formalize specific linguistic phenomena. Why should we subordinate the use of specific type change rules to their derivability in some calculus?

One of the most compelling reasons to do so is given by Moortgat (see [Moo97]) who stresses the systematicity of the relation between syntax and semantics provided in a deductive framework. Also, Lambek Calculus enjoys an important property: it is sound and complete with respect to free semigroup model, i.e. an interpretation with respect to formal languages . That is to say, rules that are not deducible in Lambek Calculus are not sound, and so they can be considered as linguistically implausible.

5.3 (Associative) Lambek Calculus

Categorial grammars can be analyzed from a proof theoretical perspective by observing the close connection between the "slashes" of a categorial grammar and implication in intuitionistic logics. The rule that allows us to infer that if w is of type A/B and v is of type B, then wv is of type A, behaves like the *modus ponens* rule of inference in logic. On the basis of this similarity Lambek proposed an architecture for categorial grammars based on two levels:

- a syntactic calculus, i.e. a deductive system in which statement of the form

  $$A_1, \ldots, A_n \vdash B,$$

  to be read "from the types $A_1, \ldots, A_n$ we can infer type B" can be proved;
- a categorial grammar as presented in definition 5.1, wherein the relation $\Rightarrow$ is changed to allow any type change rule that could be deduced at the previous level.

In doing so, instead of adding a finite number of type change rules to our grammar, every type change rule that can be derived in the Lambek Calculus is added to the categorial grammar.

The following formalizations for Lambek Calculus are presented according, respectively, to the formalism of sequent calculus and to the formalism of natural deduction. Note that in the present work we will use the expression *Lambek Calculus* to refer to *product-free Lambek Calculus*: indeed we will never make use of the product '$\cdot$' (which corresponds to the tensor of linear logic).

Definition 5.7 *The sequent calculus formalization of the Lambek calculus contains the axiom [ID] and the rules of inference [/R], [/L], [\R], [\L], and [Cut]:*

$$\frac{}{A \vdash A}\,[ID]$$

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

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

$$\frac{\Delta \vdash B \quad \Gamma, B, \Pi \vdash A}{\Gamma, \Delta, \Pi \vdash A}\,[Cut]$$

Note: in $[/R]$ *and* $[\backslash R]$ *there is a side condition stipulating that* $\Gamma \neq \emptyset$.

The side condition imposed for $[/R]$ and $[\backslash R]$ rules formalizes the fact that in Lambek Calculus one is not allowed to cancel all the premises from the left-hand side of a derivation. Otherwise stated, in Lambek Calculus there are no deductions of the form

$$\vdash A.$$

Coherently with our interpretation of Lambek Calculus as a deductive system to derive the type of a sequence of symbols of the alphabet out of the types of each symbol, such a derivation makes no sense, since it would mean assigning a type to an empty sequence of words.

Definition 5.8 *The natural deduction formalization of the Lambek Calculus is defined as follows:*

$$A \ [ID]$$

$$\frac{\begin{array}{cc}\vdots & \vdots \\ A/B & B\end{array}}{A}[/E] \qquad \frac{\begin{array}{cc}\vdots & \vdots \\ B & B\backslash A\end{array}}{A}[\backslash E]$$

$$\frac{\begin{array}{c}[B] \\ \vdots \\ A\end{array}}{A/B}[/I] \qquad \frac{\begin{array}{c}[B] \\ \vdots \\ A\end{array}}{B\backslash A}[\backslash I]$$

Note: in [/I] and [\I] rules the cancelled assumption is always, respectively, the rightmost and the leftmost uncancelled assumption, and there must be at least another uncancelled hypothesis.

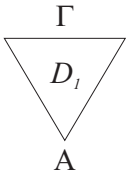
Both formalisms have advantages and disadvantages. However, due to the close connection between natural deduction proofs and λ-terms and because the tree-like structure of deductions resembles derivations trees of grammars, the natural deduction version will be the primary object of study in the present work.

For later purposes we introduce here the notion of *derivation* in Lambek calculus that will be useful later for the definition of the *structure* of a sentence in a Lambek grammar. A derivation of B from $A_1, \ldots, A_n$ is a certain kind of unary-binary branching tree that encodes a proof of $A_1, \ldots, A_n \vdash B$. Each node of a derivation is labeled with a type, and each internal node has an additional label which, for Lambek grammars, is either $/E, \backslash E, /I$, or $\backslash I$ and that indicates which Lambek calculus rule is used at each step of a derivation. For each occurrence of an introduction rule there must be a corresponding previously unmarked leaf type A which must be marked as $[A]$ (that corresponds to "discharging" an assumption in natural deduction).

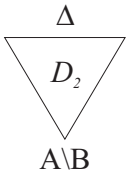
The set of derivations is inductively defined as follows:

Definition 5.9 *Let* $A, B \in \mathrm{Tp}$ *and* $\Gamma, \Delta \in \mathrm{Tp}^+$,

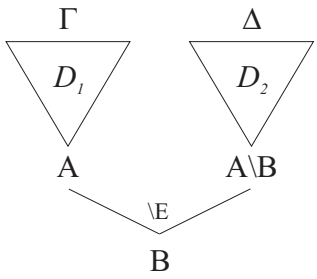
- *A (the tree consisting of a single node labeled by A) is a derivation of A from A.*
- **"Backslash elimination".** *If*



is a derivation of A *from* Γ *and*

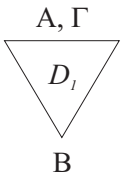


is a derivation of $A\backslash B$ *from* Δ*, then*

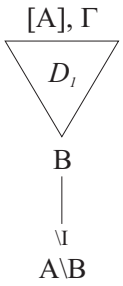


is a derivation of B *from* Γ, Δ.

- **"Backslash introduction".** *If*

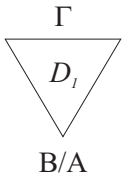


is a derivation of B *from* $\{A, \Gamma\}$*, then*

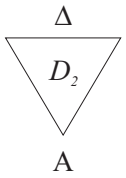


is a derivation of $A\backslash B$ *from* Γ*. The leaf labeled by* [A] *is called a* discharged *leaf.*

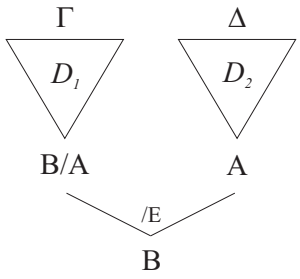
- **"Slash elimination".** *If*



is a derivation of B/A *from* Γ *and*

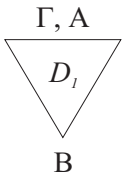


is a derivation of A *from* Δ*, then*

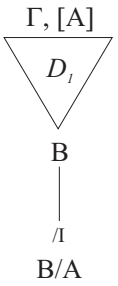


is a derivation of B *from* Γ, Δ.

- **"Slash introduction".** *If*

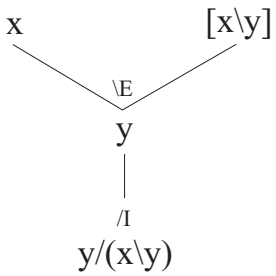


is a derivation of B from $\{\Gamma, A\}$ then



is a derivation of B/A from Γ. The leaf labeled by [A] *is called a* discharged *leaf.*

Example 5.10 *The following example is a derivation of x from $y/(x\backslash y)$ (which proves one of the two type-raising rules in Lambek Calculus):*



5.4 Non-associative Lambek Calculus

Lambek Calculus, as defined in the previous section, is implicitly associative. In order to use Lambek calculus to describe some linguistic phenomena we have to forbid associativity and so the hierarchical embedding of hypotheses is respected. Another linguistically attractive feature of non-associative Lambek calculus is that it provides useful logical to

support semantics, but at the same time it prohibits transitivity, that sometimes leads to overgeneration.

Definition 5.11 *The natural deduction formalization of the non-associative Lambek Calculus (SND) has the following axioms and rules of inference, presented in the sequent format:*

$$\frac{}{A \vdash A}\,[ID]$$

$$\frac{\Gamma \vdash A/B \quad \Delta \vdash B}{(\Gamma,\Delta) \vdash A}\,[/E] \qquad \frac{\Gamma \vdash B \quad \Delta \vdash B\backslash A}{(\Gamma,\Delta) \vdash A}\,[\backslash E]$$

$$\frac{(\Gamma,B) \vdash A}{\Gamma \vdash A/B}\,[/I] \qquad \frac{(B,\Gamma) \vdash A}{\Gamma \vdash B\backslash A}\,[\backslash I]$$

Note: in $[/I]$ *and* $[\backslash I]$ *there is a side condition stipulating that* $\Gamma \neq \emptyset$.

5.5 Normalization and Normal Forms

As one can easily see, in Lambek Calculus there are infinitely many proofs for any deduction $A_1, \ldots, A_n \vdash B$. Since, as it will be extensively explained in section 6, proofs in Lambek Calculus play a decisive role in defining the notion of *structure* for a sentence generated by a Lambek grammar, such an arbitrary proliferation of proofs for deductions is quite undesirable.

The following definition introduces a useful relation between proofs in Lambek Calculus that formalizes our idea of a "minimal" proof for any deduction. It provides two normalization schemes that can be applied to a derivation to produce a "simpler" derivation of the same result.

Definition 5.12 *The relation* $>_1$ *between proofs in the natural deduction formalization of Lambek Calculus is defined in the following way:*

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A\backslash B}\,[\backslash I] \quad \vdots\ A}{B}\,[\backslash E] \;>_1\; \begin{array}{c}\vdots\\ B\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{B/A}\,[/I] \quad \vdots\ A}{B}\,[/E] \;>_1\; \begin{array}{c}\vdots\\ A\\ \vdots\\ B\end{array}$$

$$\dfrac{\dfrac{[B] \quad \begin{array}{c}\vdots\\ B\backslash A\end{array}}{A}\,[\backslash E]}{B\backslash A}\,[\backslash I] \;>_1\; \begin{array}{c}\vdots\\ B\backslash A\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ A/B\end{array} \quad [B]}{A}\,[/E]}{A/B}\,[/I] \;>_1\; \begin{array}{c}\vdots\\ A/B\end{array}$$

The symbol $\geq$ *stands for reflexive and transitive closure of* $>_1$. *Relation* $>_1$ *is usually defined as* β-η*-conversion, while* $\geq$ *as* β-η*-reduction.*

The relation $\geq$ satisfies the following properties (see [Wan93], [Roo91]):

Theorem 5.13 (Wansing, 1993) *The relation $\geq$ is confluent (in the Church-Rosser meaning), i.e. if $\delta_1 \geq \delta_2$ and $\delta_1 \geq \delta_3$, then there exists a δ_4 such that $\delta_2 \geq \delta_4$ and $\delta_3 \geq \delta_4$.*

Theorem 5.14 (Roorda, 1991) *The relation $\geq$ is both weakly and strongly normalizing, that is, every proof can be reduced in normal form and every reduction terminates after at most a finite number of steps.*

Definition 5.15 (β-η-normal form) *A proof tree for the Lambek Calculus is said to be in* β-η-normal form *is none of its subtrees is of the form*

$$\dfrac{\dfrac{\begin{array}{c}[B]\\ \vdots\\ A\end{array}}{A/B}\,[/I] \quad B}{A}\,[/E] \qquad \dfrac{B \quad \dfrac{\begin{array}{c}[B]\\ \vdots\\ A\end{array}}{B\backslash A}\,[\backslash I]}{A}\,[\backslash E]$$

$$\dfrac{\dfrac{A/B \quad [B]}{A}\,[/E]}{A/B}\,[/I] \qquad \dfrac{\dfrac{[B] \quad B\backslash A}{A}\,[\backslash E]}{B\backslash A}\,[\backslash I]$$

5.6 Basic Facts about Lambek Calculus

Let's summarize here some meaningful properties for Lambek calculus, which is:

- *intuitionistic*: only one formula is allowed on the right-hand side of a deduction. This means there is neither involutive negation, nor disjunction;
- *linear*: so-called structural rules of logics are not allowed: two equal hypotheses can't be considered as only one, and on the other hand we are not allowed to "duplicate" hypotheses at will. Lambek calculus is what we call a resource-aware logics, wherein hypotheses must be considered as consumable resources;
- *non-commutative*: hypotheses don't commute among them, that is, the implicit operator "$\cdot$" in this calculus is not commutative. This is what makes possible the existence of the two "implications" ($/$ and $\backslash$), the first one consuming its right argument, the second one its left argument.

Since Lambek proved a cut-elimination theorem for his calculus (see [Lam58]), among the many consequences of the normalization theorems there are the subformula property, that is:

Proposition 5.16 *Every formula that occurs in a normal form natural deduction proof of cut-free sequent calculus proof is either a subformula of the (uncancelled) assumptions or of the conclusion;*

and decidability for Lambek calculus:

Proposition 5.17 *Derivability in the Lambek Calculus is decidable.*

In fact, given a sequent to prove in Lambek calculus, cut-elimination property authorizes us to look for a cut-free proof. But if the sequent comes from the application of a rule other that cut, this can't but be made in a finite number of different ways, and in any case we have to prove one or two smaller (i.e. with less symbols) sequents. This is enough to prove decidability for Lambek calculus.

Theorem 5.14 states that any proof has a normal form and theorem 5.13 that this normal form is unique. This doesn't mean that there is a unique normal form proof for any deduction. The following theorem by van Benthem sheds light on this point:

Theorem 5.18 (van Benthem) *For any sequent*

$$A_1, \ldots, A_n \vdash B$$

there are only finitely many different normal form proofs in the Lambek Calculus.

This is quite an unsatisfactory result: we still have a one-to-many correspondence between a sequent and its normal proofs. This leads to what is generally known as the problem of *spurious ambiguities* for Lambek grammars.

5.7 Lambek Grammars

A Lambek grammar extends the traditional notion of categorial grammars as presented in section 5.1 by a whole deductive system in the following way:

- a lexicon assigns to each word w_i a finite set of types

$$F(w_i) = \{t_i^1, \ldots, t_i^{k_i}\} \subset \wp(Tp);$$

- the language generated by this fully lexicalized grammar is the set of all the sequences $w_1 \cdots w_n$ of words of the lexicon such that for each w_i there exists a type $t_i \in F(w_i)$ such that

$$t_1, \ldots, t_n \vdash s$$

  is provable in Lambek calculus.

Formally:

Definition 5.19 (Lambek grammar) *A Lambek grammar is a triple $G = \langle \Sigma, s, F \rangle$, such that*

- *Σ is a finite set (the vocabulary),*
- *s is the distinguished category (a propositional variable),*
- *$F : \Sigma \to \wp(Tp)$ is a function which maps each symbol of the alphabet into the set if its types. If $F(a) = \{A_1, \ldots, A_n\}$ we write $G : a \mapsto A_1, \ldots, A_n$.*

For $w \in \Sigma^*, w = a_1 \cdots a_n$, we say that G *accepts* w if there is a proof in Lambek calculus of

$$A_1, \ldots, A_n \vdash s$$

with $G : a_i \mapsto A_i$ for each i.

The language generated by a Lambek grammar G is

$$\mathrm{L}(G) = \{a_1 \cdots a_n \in \Sigma^* \mid \text{for } 1 \leq i \leq n,\ \exists A_i,\ G : a_i \mapsto A_i \text{ and } A_1, \ldots, A_n \vdash s\}.$$

Example 5.20 *Let $\Sigma =$* {**Mary, cooked, the, beans**} *be our alphabet and s our distinguished category. Let's take F such that*

Mary	:	np
cooked	:	$(np\backslash s)/np$
the	:	np/n
beans	:	n

Then Mary cooked the beans *belongs to the language generated by this grammar, because in Lambek calculus we can prove:*

$$np, (np\backslash s)/np, np/n, n \vdash s$$

Weak generative capacity for associative Lambek grammars was characterized (see [Pen97]) by the following celebrated theorem, one of the finest and most recent achievements in this field:

Theorem 5.21 (Pentus, 1997) *The languages generated by associative Lambek grammars are exactly the context-free languages.*

Analogously, for non-associative Lambek grammars Buszkowski proved (see [Bus86]):

Theorem 5.22 (Buszkowski, 1986) *The languages generated by non-associative Lambek grammars are exactly the context-free languages.*

6 Proofs as Grammatical Structures

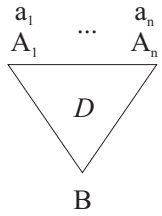
In this section we will introduce the notion of *structure* for a sentence generated by a Lambek grammar. On the basis of a recent work by Hans-Joerg Tiede (see [Tie99]) who proved some important theorems about the tree language of proof trees in Lambek calculus, we will adopt as the underlying structure of a sentence in a Lambek grammar a proof of its well-formedness in Lambek calculus. We will see in section 9 how this choice affects the process of learning a rigid Lambek grammar on the basis of *structured positive data*.

6.1 (Partial) Parse Trees for Lambek Grammars

Just as a derivation encodes a proof of $A_1, \ldots, A_n \vdash B$, the notion of *parse tree* introduced by the following definition encodes a proof of $a_1 \cdots a_n \in \mathrm{L}(G)$ where G is a Lambek grammar and $a_1, \ldots, a_n$ are symbols of its alphabet.

Definition 6.1 *Let $G = \langle \Sigma, s, F \rangle$ be a Lambek grammar, then*

- *if $\mathcal{D}$ is a derivation of B from $A_1, \ldots, A_n$, and $a_1, \ldots, a_n$ are symbols of alphabet Σ such that $G : a_i \mapsto A_i$ for $1 \leq i \leq n$, the result of attaching $a_1, \ldots, a_n$, from left to right in this order, to the undischarged leaf nodes of $\mathcal{D}$ is a* partial parse tree *of G.*



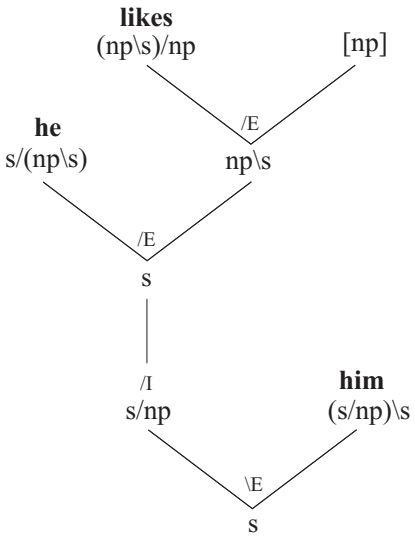
- *A* parse tree *of G is a partial parse tree of G whose root node is labeled by the distinguished category s.*

If $a_1 \cdots a_n$ is the string of symbols attached to the leaf nodes of a partial parse tree $\mathcal{P}$, $a_1 \cdots a_n$ is said to be the *yield* of $\mathcal{P}$. If a parse tree $\mathcal{P}$ of G yields $a_1 \cdots a_n$, then $\mathcal{P}$ is called a *parse* of $a_1 \cdots a_n$ in G.

Example 6.2 *Let* $\Sigma = \{\mathbf{he}, \mathbf{him}, \mathbf{likes}\}$ *be our alphabet and let* G *a Lambek grammar such that*

$$\begin{aligned} G : \mathbf{likes} &\mapsto (np\backslash s)/np, \\ \mathbf{he} &\mapsto s/(np\backslash s), \\ \mathbf{him} &\mapsto (s/np)\backslash s. \end{aligned}$$

Then the following is a parse for **he likes him***:*



6.2 Tree Languages and Automata

In order to fully appreciate the peculiarity of Lambek grammars with respect to their strong generative capacity, we recall here some basic definitions about the notion of *tree language* as presented in [Tie99].

Definition 6.3 (Trees and tree languages) *A* tree *is a term over a finite signature* Σ *containing function and constant symbols. The set of n-ary function symbols in* Σ *will be denoted by* Σ_n*. The set of all terms over* Σ *will be denoted by* T_Σ*; a subset of* T_Σ *is called a* tree language *or a* forest.

Definition 6.4 (Yield of a tree) *The* yield *of a tree t is defined by*

$$\begin{aligned} yield(c) &= c, \quad \textit{for } c \in \Sigma_0 \\ yield(f(t_1, \ldots, t_n)) &= yield(t_1), \ldots, yield(t_n), \quad \textit{for } f \in \Sigma_n, n > 0 \end{aligned}$$

Thus, the yield of a tree is the string of symbols occurring as its leaves.

Definition 6.5 (Root of a tree) *The* root *of a tree t is defined by*

$$\begin{aligned} root(c) &= c, \quad \textit{for } c \in \Sigma_0 \\ root(f(t_1, \ldots, t_n)) &= f, \quad \textit{for } f \in \Sigma_n, n > 0. \end{aligned}$$

In the following subsections three increasingly more powerful classes of tree languages are presented: local, regular and context-free tree languages. Note that even if the names for these classes of tree languages are the same as those for classes of string languages, their meaning is very different.

6.2.1 Local Tree Languages

We can think of a local tree language as a tree language whose membership problem can be decided by just looking at some very simple (local) properties of trees. A formalization of such an intuitive notion is given by the following definitions:

Definition 6.6 (Fork of a tree) *The* fork *of a tree t is defined by*

$$\begin{aligned} fork(c) &= \emptyset, \quad \textit{for } c \in \Sigma_0 \\ fork(f(t_1, \ldots, t_n)) &= \{\langle f, root(t_1), \ldots, root(t_n)\rangle\} \cup \bigcup_{i=1}^{n} fork(t_i) \end{aligned}$$

Definition 6.7 (Fork of a tree language) *For a tree language L, we define*

$$fork(L) = \bigcup_{t \in L} fork(t)$$

Note that, since Σ is finite, $fork(T_\Sigma)$ is always finite.

Definition 6.8 (Local tree language) *A tree language $L \subseteq T_\Sigma$ is* local *if there are sets $R \subseteq \Sigma$ and $E \subseteq fork(T_\Sigma)$, such that, for all $t \in T_\Sigma$, $t \in L$ iff $root(t) \in R$ and $fork(t) \subseteq E$.*

Thatcher (see [Tha67]) characterized the relation between local tree languages and the derivation trees of context-free string grammars by the following

Theorem 6.9 (Thatcher, 1967) *S is the set of derivation trees of some context-free string grammar iff S is local.*

6.2.2 Regular Tree Languages

Among many different equivalent definitions for regular tree languages, we follow Tiede's approach in choosing the following one, based on finite tree automata.

Definition 6.10 (Finite tree automaton) *A finite tree automaton is a quadruple* $\langle \Sigma, Q, q_0, \Delta \rangle$, *such that*

- Σ *is a finite signature,*
- Q *is a finite set of unary states,*
- $q_0 \in Q$ *is the start state,*
- Δ *is a finite set of transition rules of the following type:*

$$\begin{array}{rcl} q(c) & \rightarrow & c \ \ \textit{for } c \in \Sigma_0 \\ q(f(v_1, \ldots, v_n)) & \rightarrow & f(q_1(v_1), \ldots, q_n(v_n)) \ \ \textit{for } f \in \Sigma_n,\ q, q_1, \ldots, q_n \in Q \end{array}$$

We can think of a finite tree automaton as a device which scans non-deterministically a tree from root to frontier. It accepts a tree if it succeeds in reading the whole tree, it rejects it otherwise.

In order to define the notion of tree language accepted by a regular tree automaton we need to define the transition relation for finite tree automata.

Definition 6.11 *A* context *is a term over* $\Sigma \cup \{x\}$ *containing the zero-ary term* x *exactly once.*

Definition 6.12 *Let* $M = \langle \Sigma, Q, q_0, \Delta \rangle$ *be a finite tree automaton, the derivation relation*

$$\Rightarrow_M \subseteq T_{Q \cup \Sigma} \times T_{Q \cup \Sigma}$$

is defined by $t \Rightarrow_M t'$ *if for some context* s *and some* $t_1, \ldots, t_n \in T_\Sigma$, *there is a rule in* Δ

$$q(f(v_1, \ldots, v_n)) \rightarrow f(q_1(v_1), \ldots, q_n(v_n))$$

and

$$\begin{array}{rcl} t & = & s[x \mapsto q(f(t_1, \ldots, t_n))] \\ t' & = & s[x \mapsto f(q_1(t_1), \ldots, q_n(t_n))]. \end{array}$$

If we use $\Rightarrow_M^*$ *to denote the reflexive, transitive closure of* $\Rightarrow_M$, *we say that a finite automaton* M accepts *a term* $t \in T_\Sigma$ *if* $q_0(t) \Rightarrow_M^* t$. *The tree language accepted by a finite tree automaton* M *is*

$$\{t \in T_\Sigma \mid q_0(t) \Rightarrow_M^* t\}.$$

Definition 6.13 (Regular tree language) *A tree language is* regular *if it is accepted by a finite tree automaton.*

The following theorem (see [Koz97]) defines the relation between local and regular tree languages:

Theorem 6.14 *Every local tree language is regular.*

while the following (see [GS84]) establishes a relation between regular tree languages and context-free string languages:

Theorem 6.15 *The yield of any regular tree language is a context-free string language.*

6.2.3 Context-free Tree Languages

The final step in the definition of more and more powerful tree language classes is made possible by introducing the notion of *pushdown tree automaton*. Again, we stick to Tiede's approach in choosing Guesserian's useful definition (see [Gue83]):

Definition 6.16 (Pushdown tree automaton) *A pushdown tree automaton is a system* $\langle \Sigma, \Gamma, Q, q_0, Z_0, \Delta \rangle$*, such that*

- Σ *is a finite signature (the input signature),*
- Γ *is a finite signature (the pushdown signature; we assume* $\Sigma \cap \Gamma = \emptyset$*),*
- Q *is a finite set of binary states,*
- $q_0 \in Q$ *is the start state,*
- $Z_0 \in \Gamma$ *is the initial stack symbol,*
- Δ *is a finite set of rules of the form*

$$\begin{aligned} q(f(v_1, \ldots, v_n), E(x_1, \ldots, x_m)) &\rightarrow f(q_1(v_1, \gamma_1), \ldots, q_n(v_n, \gamma_n)), \\ q(v, E(x_1, \ldots, x_m)) &\rightarrow q'(v, \gamma'), \\ q(c) &\rightarrow c \end{aligned}$$

  with

  - $q, q', q_1, \ldots, q_n \in Q$,
  - $c \in \Sigma_0$,
  - $f \in \Sigma_n$, $n > 0$,
  - $E \in \Gamma_m$,
  - $\gamma', \gamma_1, \ldots, \gamma_n \in T_{\Gamma \cup \{x_1, \ldots, x_m\}}$.

The transition relation for pushdown tree automata $\Rightarrow$ can be defined straightforwardly as a generalization of definition 6.12. A term t is accepted by a pushdown automaton if $q_0(t, Z_0) \Rightarrow^* t$, where $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$.

Definition 6.17 (Context-free tree language) *The language accepted by a pushdown tree automaton is called a* context-free tree language.

The relationship between regular and context-free tree languages is exemplified by the following proposition:

Proposition 6.18 *The intersection of a regular and a context-free tree language is context-free.*

We know that the yield of a regular tree language is a context-free string language: there is a similar connection between the class of context-free tree languages and the class of indexed languages, as stated by the following

Proposition 6.19 *The yield of any context-free tree language is an indexed string language.*

Indexed languages have been proposed as an upper bound of the complexity of natural languages, after it was shown that certain phenomena in natural languages cannot be described with context-free grammars (see [Gaz88]).

6.3 Proof Trees as Structures for Lambek Grammars

In [Tie99] Hans-Joerg Tiede proposes, in contrast with a previous approach by Buszkowski, to take as the structure underlying a sentence generated by a Lambek grammar, one of the infinite proof trees of the deduction $A_1, \ldots, A_n \vdash s$, where $A_1, \ldots, A_n$ is a sequence of types assigned by the grammar to each symbol, and s is the distinguished atomic category.

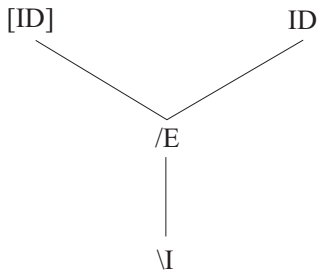
Following Tiede's approach, we give the following

Definition 6.20 (Proof tree) *A* proof tree *for a Lambek grammar is a term over the signature* $\Sigma = \{[/E], [\backslash E], [/I], [\backslash I], [ID]\}$ *where*

- $[ID]$ *is the 0-ary function symbol,*
- $[/E]$ *and* $[\backslash E]$ *are the binary function symbols,*
- $[/I]$ *and* $[\backslash I]$ *are the unary function symbols.*

The terms over this signature represent proof trees that neither have information about the formulas for which they are a proof, nor about the strings that are generated by a grammar using this proof. These terms represent proofs unambiguously, since the assumption discharged by an introduction rule is univocally determined by the position of the corresponding $[/I]$ or $[\backslash I]$ function symbol in the proof tree.

Example 6.21 *The term $t = [\backslash I]([/E]([ID], [ID]))$ is an example of well-formed term over this signature. There's no need for additional information about the discharged assumption since, as we can see from the tree-like representation of the term, the discharged assumption is unambiguously identified.*



The following terms are examples of not well-formed proof trees for the tree language generated by any Lambek grammar:

- $[\backslash E](x, [/I](y))$. Since the major premise of the $\backslash E$ function symbol is something with a $(\ldots)\backslash(\ldots)$ shape, there's no way to reduct that term by a $\backslash E$ rule;
- $[/E]([\backslash I](x), y)$. Analogous to the previous situation;
- $[\backslash I]([\backslash E](x, [ID]))$ if the term x does not contain at least two uncancelled assumptions;
- $[/I]([/E]([ID], x))$, if the term x does not contain at least two uncancelled assumptions.

By taking a proof tree as the structure of a sentences generated by Lambek grammars, Tiede proved some important results about their strong generative capacity, that is, the set of the structures assigned by a grammar to the sentences it generates. Since strong generative capacity can provide a formal notion of the linguistic concept of *structure* of a sentence, this result justifies the current interest toward Lambek Grammars as a promising mathematical tool for linguistic purposes.

Theorem 6.22 (Tiede, 1999) *The set of well-formed proof trees of the Lambek Calculus is not regular.*

Theorem 6.23 (Tiede, 1999) *The set of proof trees of the Lambek Calculus is a context-free tree language.*

These two theorems show that the language of proof trees is properly a context-free tree language.

In particular, these theorems show that Lambek grammars are more powerful, with respect to strong generative capacity, than context-free grammars, whose structure language

is a local tree language as shown in theorem 6.9.

We can easily introduce the notion of *normal form proof tree* by simply extending the notion of normal form proof as presented in definition 5.15. We can say that for normal form trees, in addition to the rules that prohibit terms of the form

$$[\backslash E](x, [/I](y)),$$
$$[/E]([\backslash I](x), y),$$

we have rules that prohibit terms of the form

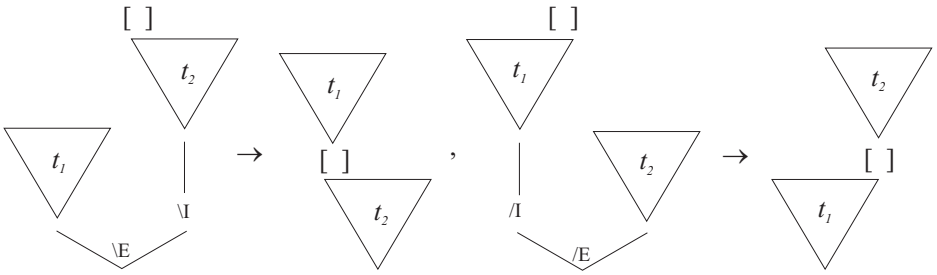
$$[\backslash E](x, [\backslash I](y))$$
$$[/E]([/I](x), y)$$

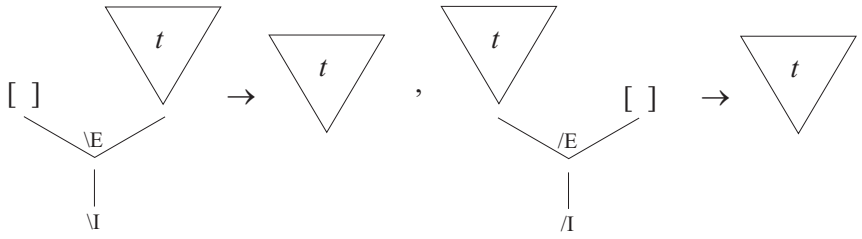
and terms of the form

$$[/I]([/E](x, [ID]))$$
$$[\backslash I]([\backslash E]([ID], y))$$

which correspond to β-redexes and η-redexes, respectively, as one can easily see from definition 5.15.

We can easily extend to the formalism of proof trees the "reduction rules" we've seen in section 5.5 to get a normal form proof tree out of a non-normal one.





As a corollary of theorem 6.22, Tiede proves that

Theorem 6.24 (Tiede, 1999) *The set of normal form proof trees of the Lambek Calculus is not regular,*

which, together with

Theorem 6.25 *The set of normal form proofs of the Lambek Calculus is a context-free tree language*

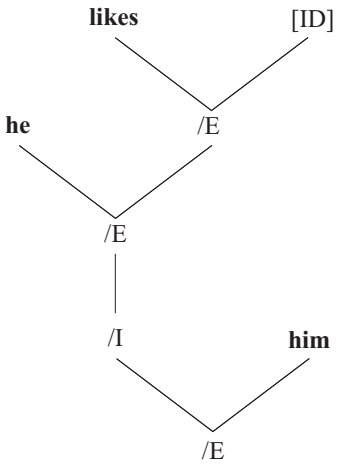
shows that the tree language of normal form proof trees of Lambek Calculus is properly a context-free tree language.

6.4 Proof-tree Structures

Given a Lambek grammar G, a *proof-tree structure* over its alphabet Σ is a unary-binary branching tree whose leaf nodes are labeled by either $[ID]$ (these are called "discharged leaf nodes") or symbols of Σ and whose internal nodes are labeled by either $\backslash E, /E, \backslash I$, or $/I$.

The set of proof-tree structures over Σ is denoted Σ^P. Often we will simply say 'structure' to mean proof-tree structure. A set of proof-tree structures over Σ is called a *structure language* over Σ.

Example 6.26 *The following is an example of a proof-tree structure for the sentence he likes him seen in example 6.2:*



Let G be a Lambek grammar, and let $\mathcal{P}$ be a partial parse tree of G. The result of stripping $\mathcal{P}$ of its type labels is a proof-tree structure, that is called the proof-tree structure of $\mathcal{P}$. If T is the structure of a parse tree $\mathcal{P}$, we say that $\mathcal{P}$ is a *parse* of T.

We say that a Lambek grammar G *generates* a structure T if and only if for some parse tree $\mathcal{P}$ of G, T is the structure of $\mathcal{P}$. The set of structures generated by G is called the *(proof-tree) structure language* of G and is denoted $\mathrm{PL}(G)$. In order to distinguish $\mathrm{L}(G)$, the language of G, from $\mathrm{PL}(G)$, its structure language, we often call the former the *string language* of G.

The *yield* of a proof-tree structure T is the string of symbols $a_1, \ldots, a_n$ labeling the *undischarged* leaf nodes of T, from left to right in this order. The yield of T is denoted $yield(T)$. Note that $\mathrm{L}(G) = \{yield(T) \mid T \in \mathrm{PL}(G)\}$.

6.5 Decidable and Undecidable Problems about Lambek Grammars

Since, as stated in by theorem 5.17, Lambek calculus is decidable, the universal membership problem "$s \in \mathrm{L}(G)$" is decidable for any sentence s and any Lambek grammar G.

On the other hand, the questions "$\mathrm{L}(G_1) = \mathrm{L}(G_2)$" and "$\mathrm{L}(G_1) \subseteq \mathrm{L}(G_2)$" for arbitrary Lambek grammars G_1 and G_2 are undecidable, because the same questions are undecidable for context-free grammars and there exists an effective procedure for converting a context-free grammar G' to a Lambek grammar G such that $L(G') = L(G)$.

Given a proof-tree structure t the question "$t \in \mathrm{PL}(G)$" is decidable. In fact, as shown by Tiede in 6.23, every proof tree language of a Lambek Grammar is a context-free tree language; and that problem is decidable for context-free tree languages (you just have to run the pushdown tree automata on t).

Unfortunately, the question "PL(G_1) $\subseteq$ PL(G_2)" has been proved decidable only for G_1, G_2 non-associative Lambek grammars. Whether it is decidable or not for (associative) Lambek grammars is still an open question and the subject of active research in this field.

6.6 Substitutions

In this section we introduce the notion of a Lambek grammar being a *substitution instance* of another. Besides, we define a notion of *size* of a Lambek grammar that will be decisive in our proof of learnability for Rigid Lambek Grammars presented in section 9.4.

First of all, let's define what we mean when we say that a Lambek grammar is subset of another one:

Definition 6.27 *Let G_1, G_2 be Lambek grammars; we say that $G_1 \subseteq G_2$ if and only if for any $a \in \Sigma$ such that $G_1 : a \mapsto A$ we have also $G_2 : a \mapsto A$.*

Example 6.28 *Let* $\{\mathbf{Francesca}, \mathbf{loves}, \mathbf{Paolo}\} \subseteq \Sigma$ *and let*

$$\begin{array}{rrcl} G_1: & \mathbf{Francesca} & \mapsto & np \\ & \mathbf{loves} & \mapsto & np\backslash s \\ G_2: & \mathbf{Francesca} & \mapsto & np \\ & \mathbf{loves} & \mapsto & np\backslash s, np\backslash(s/np) \\ & \mathbf{Paolo} & \mapsto & np \end{array}$$

Obviously, $G_1 \subseteq G_2$

Definition 6.29 *A substitution is a function $\sigma : Var \to Tp$ that maps variables to types. We can extend it to a function from types to types by setting*

$$\begin{array}{rcl} \sigma(t) & = & t \\ \sigma(A/B) & = & \sigma(A)/\sigma(B) \\ \sigma(A\backslash B) & = & \sigma(A)\backslash\sigma(B) \end{array}$$

for all $A, B \in Tp$.

We use the notation $\{x_1 \mapsto A_1, \ldots, x_n \mapsto A_n\}$ to denote the substitution σ such that $\sigma(x_1) = A_1, \ldots, \sigma(x_n) = A_n$ and $\sigma(y) = y$ for all other variables y.

Example 6.30 *Let $\sigma = \{x \mapsto x\backslash y, y \mapsto s, z \mapsto s/(s/x)\}$. Then*

$$\sigma((s/x)\backslash y) = (s/(x\backslash y))\backslash t$$

and

$$\sigma(((s/x)\backslash y)/(x/z)) = ((s/(x\backslash y))\backslash s)/((x\backslash y)/(s/(s/x))).$$

The following definition introduce the notion of a Lambek grammar being a *substitution instance* of another:

Definition 6.31 *Let $G = \langle \Sigma, s, F \rangle$ be a Lambek grammar, and σ a substitution. Then $\sigma[G]$ denotes the grammar obtained by applying σ in the type assignment of G, that is:*

$$\sigma[G] = \langle \Sigma, s, \sigma \cdot F \rangle$$

$\sigma[G]$ is called a substitution instance *of G.*

It easy to prove also for Lambek grammars this straightforward but important fact that was first proved for CCGs in [BP90]

Proposition 6.32 *If $\sigma[G_1] \subseteq G_2$, then the set of proof-tree structures generated by G_1 is a subset of the set of proof-tree structures generated by G_2, that is* $\mathrm{PL}(G_1) \subseteq \mathrm{PL}(G_2)$.

Proof. Suppose $\sigma[G_1] \subseteq G_2$. Let $T \in \mathrm{PL}(G_1)$ and let $\mathcal{P}$ be a parse of T in G_1. Let $\sigma[\mathcal{P}]$ the result of replacing each type label A of $\mathcal{P}$ by $\sigma(A)$. Then it is easy to see that $\sigma[\mathcal{P}]$ is a parse of T in G_2. Therefore, $T \in \mathrm{PL}(G_2)$.

Corollary 6.33 *If $\sigma[G_1] \subseteq G_2$, then* $\mathrm{L}(G_1) \subseteq \mathrm{L}(G_2)$.

Proof. Immediate from the previous proposition and the remark at the end of section 6.4.

A substitution that is a one-to-one function from Var to Var is called a *variable renaming*. If σ is a variable renaming, then G and $\sigma[G]$ are called *alphabetic variants*. Obviously grammars that are alphabetic variants have exactly the same shape and are identical for all purposes. Therefore, grammars that are alphabetic variants are treated as identical.

Proposition 6.34 *Suppose $\sigma_1[G_1] = G_2$ and $\sigma_2[G_2] = G_1$. Then G_1 and G_2 are alphabetic variants and thus are equal.*

Proof. For each symbol $c \in \Sigma$, σ_1 and σ_2 provide a one-to-one correspondence between $\{A \mid G_1 : c \mapsto A\}$ and $\{A \mid G_2 : c \mapsto A\}$. Indeed, if it didn't and, say, $\{\sigma_1(A) \mid G_1 : c \mapsto A\} \subset \{A \mid G_2 : c \mapsto A\}$, then $\sigma_2[G_2] = \sigma_2[\sigma_1[G_1]]$ couldn't be equal to G_1, and likewise for σ_2. Then, it is easy to see that $\sigma_1 \uparrow Var(G_1)$ is a one-to-one function from $Var(G_1)$ onto $Var(G_2)$, and $\sigma_2 \uparrow Var(G_2) = (\sigma_1 \uparrow Var(G_1))^{-1}$. One can extend $\sigma_1 \uparrow Var(G_1)$ to a variable renaming σ. Then $\sigma[G_1] = \sigma_1[G_1] = G_2$.

6.7 Grammars in Reduced Form

Definition 6.35 *A substitution σ is said to be* faithful to *a grammar G if the following condition holds:*

for all $c \in dom(G)$, if $G_1 : c \mapsto A$, $G_1 : c \mapsto B$, and $A \neq B$, then $\sigma(A) \neq \sigma(B)$.

Example 6.36 *Let G be the following grammar*

$$\begin{array}{rrcl} G: & \textbf{Francesca} & \mapsto & x, \\ & \textbf{dances} & \mapsto & x\backslash s, y \\ & \textbf{well} & \mapsto & y\backslash(x\backslash s). \end{array}$$

Let

$$\begin{aligned} \sigma_1 &= \{y \mapsto x\}, \\ \sigma_2 &= \{y \mapsto x\backslash s\}. \end{aligned}$$

Then σ_1 is faithful to G, while σ_2 is not.

Definition 6.37 *Let $\sqsubseteq$ be a binary relation on grammars such that $G_1 \sqsubseteq G_2$ if and only if there exists a substitution σ with the following properties:*

- *σ is faithful to G_1;*
- *$\sigma[G_1] \subseteq G_2$.*

From the definition above and proposition 6.34 it's immediate to prove the following:

Proposition 6.38 *$\sqsubseteq$ is reflexive, transitive and antisymmetric.*

Definition 6.39 *For any grammar G, define the* size *of G, $size(G)$, as follows:*

$$size(G) = \sum_{c \in \Sigma} \sum_{G : c \mapsto A} |A|,$$

where, for each type A, $|A|$ is the number of symbol occurrences in A.

Lemma 6.40 *If $G_1 \sqsubseteq G_2$, then $size(G_1) \leq size(G_2)$,*

Proof. For any type A and any substitution σ, $|A| \leq |\sigma(A)|$. Then the lemma is immediate from the definition of $\sqsubseteq$.

Corollary 6.41 *For any grammar G, the set $\{G' \mid G' \sqsubseteq G\}$ is finite.*

Proof. By lemma 6.40, $\{G' \mid G' \sqsubseteq G\} \subseteq \{G' \mid size(G') \leq size(G)\}$. The latter set must be finite, because for any $n \in \mathbb{N}$, there are only finitely many grammars G such that $size(G) = n$.

If we write $G_1 \sqsubset G_2$ to mean $G_1 \sqsubseteq G_2$ and $G_1 \neq G_2$, we have

Corollary 6.42 *$\sqsubset$ is well-founded.*

Definition 6.43 *A grammar G is said to be in reduced form if there is no G' such that $G' \sqsubset G$ and $\mathrm{PL}(G) = \mathrm{PL}(G')$.*

7 Lambek Grammars as a Linguistic Tool

7.1 Lambek Grammars and Syntax

As explicitly stated in the original paper wherein Lambek laid the foundations of the Lambek Calculus, his aim was

> [...] to obtain an effective rule (or algorithm) for distinguishing sentences from nonsentences, which works not only for the formal languages of interest to the mathematical logician, but also for natural languages such as English, or at least for fragments of such languages. ([Lam58])

That's why, even if Lambek grammars can be simply considered as interesting mathematical objects, it will be useful to underline here some properties that make them also an interesting tool to formalize some phenomena in natural languages.

The importance of Lambek's approach to grammatical reasoning lies in the development of a uniform *deductive* account of the composition of form and meaning in natural language: formal grammar is presented as a *logic*, that is a system to reason about structured linguistic structures.

The basic idea underlying the notion of Categorial Grammar on which Lambek based his approach is that a grammar is a formal device to assign to each word (a symbol of the alphabet of the grammar) or expression (an ordered sequence of words) one or more *syntactic types* that describe their function. Types can be considered as a formalization of the linguistic notion of *parts of speech*.

CCGs assign to each symbol a fixed set of types, and provide two composition rules to derive the type of a sequence of words out of the types of its components. Such a "fixed types" approach leads to some difficulties: to formalize some linguistic phenomena we should add further rules to the two elimination rules defined for CCGs as described in section 5.2. In the following subsections we present some examples where the deductive approach of Lambek grammars leads to more an elegant and consistent formalization of such linguistic phenomena.

In the following subsections we take s as the primitive type of *well-formed sentences* in our language and np as the primitive type for *noun phrases* (such as John, Mary, he).

7.1.1 Transitive verbs

Transitive verbs require a name both on their left and right hand sides, as it is apparent from the well-formedness of the following sentences.

$$\overset{np}{\text{John}}\ (\ \overset{(np\backslash s)/np}{\text{likes}}\ \overset{np}{\text{Mary}})$$

$$(\overset{np}{\text{John}}\ \overset{np\backslash(s/np)}{\text{likes}}\ )\ \overset{np}{\text{Mary}}$$

Both parenthesizations lead to a derivation of s as type of the whole expression. This would mean that in an CCG we should assign to any transitive verb at least two distinct types: $(np\backslash s)/np$ and $np\backslash(s/np)$.

On the contrary, in a Lambek grammar, since we can prove both

$$(np\backslash s)/np \vdash np\backslash(s/np)$$

and

$$np\backslash(s/np) \vdash (np\backslash s)/np$$

we can simply assign to a transitive verb the type $np\backslash s/np$ without any further parenthesizations.

7.1.2 Pronouns

If we try to assign a proper type to the personal pronoun *he* we notice that its type is such that the following sentences are well-formed:

$$\overset{}{\text{he}}\ \overset{np\backslash s}{\text{works}},$$

$$\text{he}\ \overset{np\backslash s/np}{\text{likes}}\ \overset{np}{\text{Jane}}$$

We have two choices: either we give *he* the same type as a name (that is, np) or we give it the type $s/(np\backslash s)$. In the first case there is a problem: expressions like *Jane likes he* are considered as well-formed sentences. So, we assign to *he* the type $s/(np\backslash s)$.

Analogously, since the personal pronoun *him* makes the following sentences well-formed:

$$\overset{np}{\text{Jane}}\ \overset{np\backslash s/np}{\text{likes}}\ \text{him}$$

$$\overset{np}{\text{Jane}}\ \overset{np\backslash s}{\text{works}}\ \overset{s\backslash s/np}{\text{for}}\ \text{him},$$

we assign to *him* the type $(s/np)\backslash s$ (and not type np, since expressions like *him likes John* would be well-formed).

Since a pronoun is, according to its own definition, something that "stands for a noun", we wish that in our grammar each occurrence of a pronoun could be replaced by a name (while the converse is not always true): but this means that any name (say, John, of type np) should also be assigned the type of *he* and *him*, that is, respectively, type $s/(np\backslash s)$ and type $(s/np)\backslash s$. In other words, we need something that accounts for a type-raising. But since in Lambek Calculus we can prove

$$np \vdash s/(np\backslash s)$$
$$np \vdash (s/np)\backslash s$$

for any np and s, a Lambek grammar provides a very natural formalization of the relationship between names and pronouns: while a name can always be substituted to a pronoun in a

sentence (and the type-raising derivation guarantees that a name can always "behave like" a pronoun if we need it to), the converse is not true (the converse of the type-raising proof doesn't hold in Lambek Calculus). The proof of the first deduction is reported in example 5.10 as a derivation in a Lambek grammar.

7.1.3 Adverbs

If we look for the proper type for adverbs like *here* we can consider the well-formed sentence *John works here*. We can choose between two possible parenthesizations here, that is:

$$(\overset{np}{\text{John}}\ \overset{np\backslash s}{\text{works}}\ )\ \text{here}$$

$$\overset{np}{\text{John}}\ (\overset{np\backslash s}{\text{works}}\ \text{here})$$

The first one suggests for *here* the type $s\backslash s$, while the second one the type $(np\backslash s)\backslash(np\backslash s)$. The good news is that, while in a CCG we should assign each adverb at least two different types, in a Lambek grammar we can prove that

$$s\backslash s \vdash (np\backslash s)\backslash(np\backslash s)$$

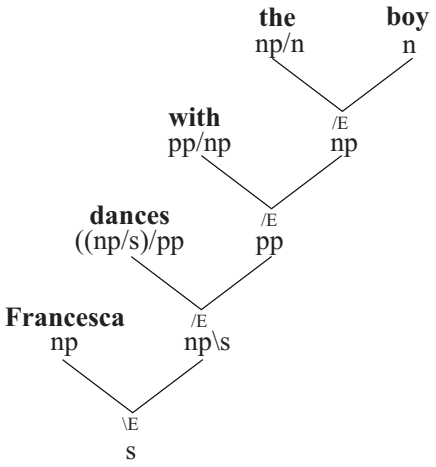
that is to say, in Lambek grammars any adverbial expression of type $s\backslash s$ has also type $(np\backslash s)\backslash(np\backslash s)$. More generally, we can show that in Lambek Calculus

$$x\backslash y \vdash (z\backslash x)\backslash(y\backslash x)$$
$$x/y \vdash (x/z)/(y/z).$$

7.1.4 Hypothetical reasoning

In the following example, sentences s, noun phrases np, common nouns n, and propositions phrases pp are taken to be "complete expressions", whereas the verb *dances*, the determiner *the* and the preposition *with* are categorized as incomplete with respect to these complete phrases.

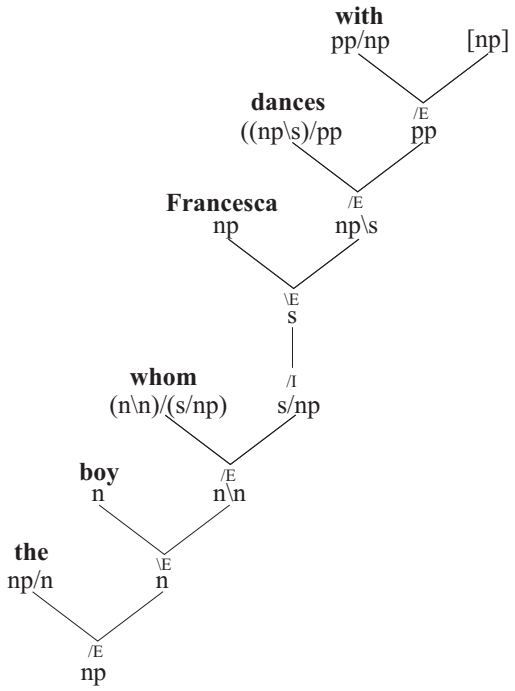
Example 7.1 *Here is the derivation for the sentence* Francesca dances with the boy.



This is an example of grammatical reasoning where, on the basis of the types we assigned to each word, we infer the well-formedness of a sequence of words. On the other hand we can assume a different perspective: knowing that a sentence is well-formed, what can be said about the type of its components? In the words of Lambek: "Given the information about the categorization of a composite structure, what conclusions could be draw about the categorization of its parts?" ([Lam58]). That's where the following inference patterns come into play:

$$\begin{aligned} &\text{from } \Gamma, B \vdash A, \qquad \text{infer } \Gamma \vdash A/B, \\ &\text{from } B, \Gamma \vdash A, \qquad \text{infer } \Gamma \vdash B\backslash A \end{aligned}$$

which gives a linguistic interpretation of the role of the "introduction" rules. That's what is done in the following derivation which allows us to infer that the expression *the boy Francesca dances with* is of type *np*:



Since the relative pronoun *whom* (of type $(n\backslash n)/(s/np)$) wants to enter into composition on its right with the relative clause body, we'd like to assign type s/np to the latter. In order to show that *Francesca dances with* is indeed of type s/np, we make a *hypothetical assumption* and suppose to have a "ghost word" of type np on its right. It's easy to derive the category s for the sentence *Francesca dances with* np. By withdrawing the hypothetical np assumption, we conclude that *Francesca dances with* has type s/np.

We can say that the cancelled hypothesis is the analogous of a "trace" *à la* Chomsky moving *whom* before *Francesca*.

7.1.5 Transitivity

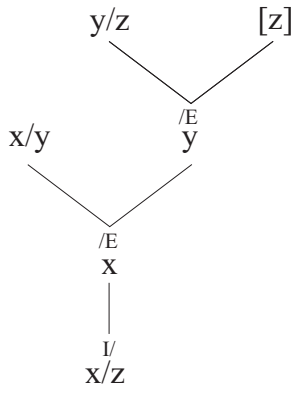
In the framework of CCGs a difficulty arises when we try to show the well-formedness of

$$\underset{\text{he}}{s/(np\backslash s)}\quad \underset{\text{likes}}{np\backslash s/np}\quad \underset{\text{him}}{(s/np)\backslash s}$$

so some authors proposed to introduce two new rules, which are often referred to as 'transitivity rules':

$$\begin{aligned}(x/y)(y/z) &\rightarrow x/z,\\ (x\backslash y)(y\backslash z) &\rightarrow x\backslash z\end{aligned}$$

It's easy to show that such rules are derivable in Lambek Calculus, as we can easily see from the following proof tree:



7.2 Lambek Grammars and Montague Semantics

From a linguistic point of view, one of the main reasons of interest in Lambek grammars lies in the natural interface that proof-tree structures provide for Montague-like semantics. Just like Curry-Howard isomorphism shows that simply typed λ-terms can be seen as proofs in intuitionistic logics, and vice-versa, syntactical analysis of a sentence in a Lambek grammar is a proof in Lambek calculus, which is naturally embedded into intuitionistic logics. Indeed, if we read B/A and $A\backslash B$ like the intuitionistic implication $A \to B$, every rule in Lambek calculus is a rule of intuitionistic logics.

In order to fully appreciate this relation between syntax and semantics which is particularly strong for Lambek grammars, we define a morphism between syntactic types and semantic types: the latter are formulas of a minimal logics (where the only allowed connector is $\to$, that is, intuitionistic implication) built on the two types e (entity) and t (truth values).

(Syntactic type)*	=	Semantic type
s^*	=	t (a sentence is a proposition)
sn^*	=	e (a nominal sintagma denotes an entity)
n^*	=	$e \to t$ (a noun is a subset of entities)

$(A\backslash B)^* = (B/A)^* = A^* \to B^*$ extends $(\_)^*$ to every types.

The lexicon associates also to every word w a λ-term τ_k for every syntactic type $t_k \in \mathcal{L}(w)$, such that the type of τ_k is precisely t_k^*, the semantic type corresponding to that syntactic type. We introduce some constants for representing logical operations of quantification,

conjunction etc:

Constant	Type
$\exists$	$(e \to t) \to t$
$\forall$	$(e \to t) \to t$
$\wedge$	$t \to (t \to t)$
$\vee$	$t \to (t \to t)$
$\supset$	$t \to (t \to t)$

Let the following be given:

- a syntactical analysis of $w_1 \dots w_n$ in Lambek calculus, that is to say, a derivation $\mathcal{D}$ of $t_1, \dots, t_n \vdash s$ and
- the semantics for every word $w_1, \dots, w_n$, that is to say, λ-terms $\tau_i : t_i^*$,

then we get the semantics of the sentence by simply applying the following algorithm:

- Substitute in $\mathcal{D}$ every syntactic type with its corresponding semantic image; since intuitionistic logics is an extension of Lambek calculus, we get a derivation $\mathcal{D}^*$ into intuitionistic logic of $t_1^*, \dots, t_n^* \vdash t = s^*$;
- this derivation in intuitionistic logic due to Curry-Howard isomorphism can be seen as a simply typed λ-term $\mathcal{D}_\lambda^*$, containing a free variable x_i of type t_i^* for every word w_i;
- in $\mathcal{D}_\lambda^*$ replace each variable x_i with λ-term τ_i, equally typed with t_i^*;
- reduce the λ-term resulting at the end of the previous step, and we get the semantic representation of the analyzed sentence.

Let's consider the following example (taken from [Ret96]):

word	**Syntactic type** t **Semantic type** t^* **Semantic representation: a** λ**-term of type** t^*
some	$(s/(sn\backslash s))/n$ $(e \to t) \to ((e \to t) \to t)$ $\lambda P : e \to t\ \lambda Q : e \to t(\exists(\lambda x : e(\wedge(Px)(Qx))))$
sentences	n $e \to t$ $\lambda x : e(\texttt{sentence}\ x)$
talkabout	$sn\backslash(s/sn)$ $e \to (e \to t)$ $\lambda x : e\ \lambda y : e((\texttt{talkabout}\ x)y)$
themselves	$((sn\backslash s)/sn)\backslash(sn\backslash s)$ $(e \to (e \to t)) \to (e \to t)$ $\lambda P : e \to (e \to t)\lambda x : e((Px)x)$

First of all, we'll prove that *Some sentences talk about themselves* is a well formed-sentence, that is, it belongs to the language generated by the lexicon at issue. This means building a natural deduction of:

$$(s/(sn\backslash s))/n, n, sn\backslash(s/sn), ((sn\backslash s)/sn)\backslash(sn\backslash s) \vdash s.$$

If we indicate with S, N, T, M the left-hand side of syntactic types we get

$$\dfrac{\dfrac{S \vdash (s/(sn\backslash s))/n \quad N \vdash n}{S, N \vdash s/(sn\backslash s)}\,[/E] \qquad \dfrac{T \vdash (sn\backslash s)/sn \quad M \vdash ((sn\backslash s)/sn)\backslash(sn\backslash s)}{T, M \vdash sn\backslash s}\,[\backslash E]}{S, N, T, M \vdash s}\,[\backslash E]$$

By applying the isomorphism between syntactic and semantic types, we get the following intuitionistic proof, where S^*, N^*, T^*, M^* are the abbreviations for semantic types associated to S, N, T, M:

$$\dfrac{\dfrac{S^* \vdash (e \to t) \to (e \to t) \to t \quad N^* \vdash e \to t}{S^*, N^* \vdash (e \to t) \to t}\,[\to E] \qquad \dfrac{T^* \vdash e \to e \to t \quad M^* \vdash (e \to e \to t) \to e \to t}{T^*, M^* \vdash e \to t}\,[\to E]}{S^*, N^*, T^*, M^* \vdash t}\,[\to E]$$

The λ-term coding this proof is simply $((sn)(tm))$ of type t, where s, n, t, m are variables of types respectively S^*, N^*, T^*, M^*.

By replacing these variables with λ-terms of the same types associated by the lexicon to the words, we get the following λ-term of type t:

$$\begin{array}{c}((\lambda P\ \lambda Q\ (\exists\ (\lambda x(\wedge(P\ x)(Q\ x))))))(\lambda x\ (\texttt{sentence}\ x)))\\ ((\lambda P\ \lambda x\ ((P\ x)x))(\lambda x\ \lambda y\ ((\texttt{talkabout}\ x)y)))\end{array}$$

$$\downarrow \beta$$

$$(\lambda Q\ (\exists(\lambda x(\wedge(\texttt{sentence}\ x)(Q\ x)))))(\lambda x((\text{talkabout}\ x)x))$$

$$\downarrow \beta$$

$$(\exists(\lambda x(\wedge(\texttt{sentence}\ x)((\texttt{talkabout}\ x)x))))$$

If we recall that the x in this last term is of type e, the latter reduced term represents the following formula in predicate calculus:

$$\exists x : e(\text{sentence}\ (x) \wedge \text{talkabout}(x, x))$$

which is the semantic representation of the previously analyzed sentence.

8 Rigid Lambek Grammars

In the present section we introduce the notion of *rigid Lambek grammar* (often referred to as RLG), whose learnability properties will be the subject of our inquiry in section 9. Basic notions and results presented here are almost trivial extensions of what has already been done for rigid CCGs (see [Kan98]), since a specific a specific theory for rigid Lambek grammars is still missing.

8.1 Rigid and k-Valued Lambek Grammars

A *rigid Lambek grammar* is a triple $G = \langle \Sigma, s, F \rangle$, where Σ and s are defined like in definition 5.19, while $F : \Sigma \rightharpoonup Tp$ is a partial function that assigns to each symbol of the alphabet *at most one type*. We can easily generalize the notion of rigid Lambek grammar to the notion of *k-valued* Lambek grammar by a function F that assigns to each symbol of the alphabet *at most k types*. Formally, $F : \Sigma \rightharpoonup \bigcup_{i=1}^{k} Tp^k$.

Let an alphabet Σ be given. We call $\mathcal{G}_{rigid}$ the class of rigid Lambek grammars over Σ, and $\mathcal{G}_{k-valued}$ the class of k-valued Lambek grammars over Σ.

Let's define two classes of proof-tree structures:

$$\begin{aligned} \mathcal{PL}_{rigid} &= \{\mathrm{PL}(G) \mid G \in \mathcal{G}_{rigid}\}, \\ \mathcal{PL}_{k-valued} &= \{\mathrm{PL}(G) \mid G \in \mathcal{G}_{k-valued}\}. \end{aligned}$$

Members of $\mathcal{PL}_{rigid}$ are called *rigid (proof-tree) structure languages*, and members of $\mathcal{PL}_{k-valued}$ are called *k-valued (proof-tree) structure languages*.

Let's define two classes of strings:

$$\begin{aligned} \mathcal{L}_{rigid} &= \{\mathrm{L}(G) \mid G \in \mathcal{G}_{rigid}\}, \\ \mathcal{L}_{k-valued} &= \{\mathrm{L}(G) \mid G \in \mathcal{G}_{k-valued}\}. \end{aligned}$$

Members of $\mathcal{L}_{rigid}$ are called *rigid (string) languages*, and members of $\mathcal{L}_{k-valued}$ are called *k-valued (string) languages*.

Example 8.1 *Let* $\{\mathbf{well}, \mathbf{Francesca}, \mathbf{dances}\} \subseteq \Sigma$ *and let* G_1, G_2 *be the following Lambek grammars:*

$$\begin{aligned} G_1 : \mathbf{Francesca} &\mapsto x, \\ \mathbf{dances} &\mapsto x\backslash s,\ y, \\ \mathbf{well} &\mapsto y\backslash(x\backslash s), \\ G_2 : \mathbf{Francesca} &\mapsto x, \\ \mathbf{dances} &\mapsto x\backslash s, \\ \mathbf{well} &\mapsto (x\backslash s)\backslash(x\backslash s). \end{aligned}$$

Then G_2 is a rigid grammar, while G_1 is not. G_1 is a 2-valued grammar.

Definition 8.2 *Any type A can be written uniquely in the following form:*

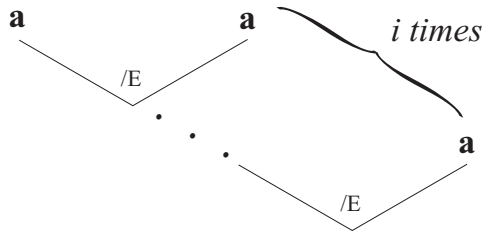
$$(\ldots((p|A_1)|A_2)|\ldots)|A_n$$

where $B|C$ stands for either B/C or $C\backslash B$ and $p \in Pr$. For $0 \leq i \leq n$, we call the subtype $(\ldots(p|A_1)|\ldots)|A_i$ of A a head subtype *of A. p is the* head *of A and is denoted $head(A)$. A_i's are called* argument subtypes *of A. The number n is called the* arity *of A.*

The following propositions are almost trivial extensions to rigid Lambek grammars of analogous results proved by Kanazawa for CCGs in [Kan98]. However, they deserve some attention since they can provide a first superficial insight about properties of RLGs.

First of all we prove a *hierarchy theorem* about strong generative capacity of k-valued Lambek grammars.

Proposition 8.3 *Let $\mathtt{a} \in \Sigma$. For each $i \geq 1$, let T_i be the following proof-tree structure:*



Then for each $k \geq 1$,

$$\{T_1, \ldots, T_k\} \in \mathcal{PL}_{k+1-valued} - \mathcal{PL}_{k-valued}.$$

Thus, for each $k \in \mathbb{N}$, $\mathcal{PL}_{k-valued} \subset \mathcal{PL}_{k+1-valued}$.

Proof. (See [Kan98]) Let G_k be the following k+1-valued grammar:

$$\begin{aligned} G_k : a \quad \mapsto \quad & x, \\ & s/x, \\ & (s/x)/x, \\ & \vdots \\ & (\ldots((s/\underbrace{x)/x)/\ldots)/x}_{k \text{ times}}. \end{aligned}$$

Then one can easily verify that $\{T_1, \ldots, T_k\} \subset \mathrm{PL}(G_k)$.

Let G be a grammar such that $\{T_1, \ldots, T_k\} \subset \mathrm{PL}(G)$: we will show that G is at least k+1-valued.

Let $\mathcal{P}_i$ be a parse of T_i in G for $1 \leq i \leq k$. Then the leftmost leaf of $\mathcal{P}_i$ is the ultimate functor of $\mathcal{P}_i$, and if we call A_i the type labeling it, we can easily verify that the its arity must be exactly i. Thus, $i \neq j$ implies $A_i \neq A_j$.

We show that there is at least one type B such that $G : a \mapsto B$ and $B \notin \{A_1, \ldots, A_k\}$. Since the relation "is an argument subtype of" is well-founded, there is at least one i such that the argument subtypes of A_i are not in $\{A_1, \ldots, A_k\}$. But in order to produce $\mathcal{P}_i$, any argument subtype of A_i must be a type assigned to a by G. Therefore G must be at least k+1-valued.

The proof of proposition 8.3 shows

Corollary 8.4 *There is no Lambek grammar G such that* $\mathrm{PL}(G) = \Sigma^P$.

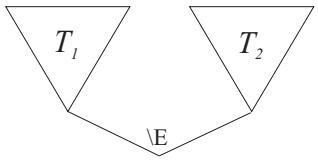
Lemma 8.5 *Let G be a rigid Lambek grammar. Then for each proof-tree structure T, there is at most one partial parse tree $\mathcal{P}$ such that T is the structure of $\mathcal{P}$.*

Proof. By induction on the construction of T.

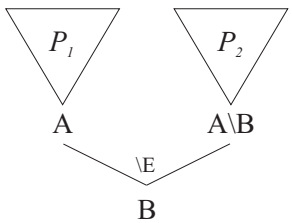
Induction basis. $T = c \in \Sigma$. Any partial parse tree $\mathcal{P}$ whose structure is T is a height 0 tree whose only node is labeled by the symbol c and a type A such that $G : c \mapsto A$. Since G is rigid, there is at most one such type A. Then $\mathcal{P}$, if it exists, is unique.

Induction step. There are 4 cases to consider:

1. T is the following proof-tree structure:



Then any partial parse tree of G whose structure is T has the form where $\mathcal{P}_1$ and $\mathcal{P}_2$

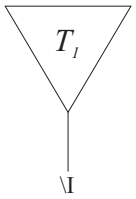


are partial parse trees of G whose structures are T_1 and T_2, respectively. By induction

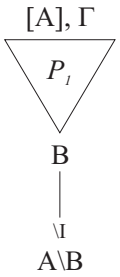
hypothesis, $\mathcal{P}_1$ and $\mathcal{P}_2$ are unique. This means that the type label B is also uniquely determined, so $\mathcal{P}$ is also unique.

2. Exactly like Case 1, with $/E$ in place of $\backslash E$.

3. T is the following proof-tree structure:



Then any partial parse tree of G whose structure is T has the form where $\mathcal{P}_1$ is a



partial parse tree of G whose structure is T_1. By induction hypothesis, $\mathcal{P}_1$ is unique. This means the the type label $A\backslash B$ is uniquely determined, so $\mathcal{P}$ is also unique.

4. Exactly like Case 3, with $/I$ in place of $\backslash I$.

Corollary 8.6 *If G is a rigid Lambek grammar, each proof-tree structure $T \in \mathrm{PL}(G)$ has a unique parse.*

Note that last corollary doesn't state that if G is rigid, then each string $s \in L(G)$ has a unique parse: in general for each sentence there are infinitely many proof trees, as extensively shown in section 6.

Lemma 8.7 *Let G be a rigid Lambek grammar. Then for each incomplete proof-tree structure T, there is at most one incomplete parse tree $\mathcal{P}$ of G such that T is the structure of $\mathcal{P}$.*

Proof. See [Kan98] trivially extended to Lambek grammars.

8.2 Most General Unifiers and $\sqcup$ Operator

Unification plays a crucial role in automated theorem proving in classical first-order logic and its extensions (see, for example, [Fit96] for an exposition of its use in first-order logic). Since types are just a special kind of terms, the notion of unification applies straightforwardly to types.

Definition 8.8 *Let A and B be types. A substitution σ is a* unifier *of A and B if $\sigma(A) = \sigma(B)$. A unifier σ is a* most general unifier *of A and B, if for any other unifier τ of A and B, there exists a substitution η, such that $\tau = \sigma \circ \eta$, i.e. $\tau(C) = \eta(\sigma(C))$, for $C = A$ or $C = B$.*

A substitution σ is said to *unify* a set $\mathbf{A}$ of types if for all $A_1, A_2 \in \mathbf{A}$, $\sigma(A_1) = \sigma(A_2)$. We say that σ unifies a family of sets of types, if σ unifies each set in the family.

A most general unifier is unique up to 'renaming of variables'.

Example 8.9 *Let $\mathcal{A}$ consist of the following sets:*

$$\begin{aligned} A_1 &= \{x_1/x_2, x_3/x_4\}, \\ A_2 &= \{x_5\backslash(x_3\backslash t\}, \\ A_3 &= \{x_1\backslash t, x_5\}. \end{aligned}$$

Then the most general unifier of $\mathcal{A}$ is:

$$\sigma = \{x_3 \mapsto x_1, x_4 \mapsto x_2, x_5 \mapsto x_1\backslash t\}.$$

There are many different efficient algorithms for unification, which decide whether a finite set of types has a unifier and, if it does, compute a most general unifier for it. For illustration purposes, we present here a non-deterministic version of an unification algorithm.

Our algorithm uses the notion of *disagreement pair*. The easiest way to define disagreement pair is to consider the types to be tree-like:

Definition 8.10 *Let A and B be two types. A* disagreement pair *for A and B is a pair of subterms of A and B, A', B', such that $A' \neq B'$ and the path from the root of A to the root of A' is equal to the path from the root of B to the root of B'.*

The following, non-deterministic version of the unification algorithm is taken from [Fit96]:

UNIFICATION ALGORITHM.

- **input:** two types A and B;
- **output:** a most general unifier σ of A, B, if it exists, or a correct statement that A and B are not unifiable.

Let $\sigma := \epsilon$
While $\sigma(A) \neq \sigma(B)$ do
begin
choose a disagreement pair A', B' for $\sigma(A), \sigma(B)$;
if neither A' nor B' is a variable, then FAIL;
let x be whichever of A', B' is a variable (if both are, choose one)
and let C be the other one of A', B'
if x occurs in C, then FAIL;
let $\sigma := \sigma \circ \{x \mapsto C\}$;
end

The previous algorithm present one of many efficient algorithms for unification, so we the following is a well-defined notion:

Definition 8.11 *We define a computable partial function mgu that maps a finite family $\mathcal{A}$ of finite sets of types to a most general unifier $mgu(\mathcal{A})$, if $\mathcal{A}$ is unifiable.*

The set $\mathcal{G}_{rigid}$ of all rigid Lambek grammars is partially ordered by $\sqsubseteq$.

Definition 8.12 *Let $\mathcal{G} \subseteq \mathcal{G}_{rigid}$, and let $G \in \mathcal{G}$. Then G is called an* upper bound *of $\mathcal{G}$ if for every $G' \in \mathcal{G}$, $G' \sqsubseteq G$.*

We introduce here a new operator among rigid grammars that will be used to prove an interesting property for our learning algorithm at the end of the fifth chapter.

Definition 8.13 *Let G_1 and G_2 be rigid Lambek grammars. We can assume that G_1 and G_2 have no common variables (if they do, we can always choose a suitable alphabetic variant of one of them such that $Var(G_1) \cap Var(G_2) = \emptyset$). Let*

$$\mathcal{A} = \{\{A \mid G_1 \cup G_2 : c \mapsto A\} \mid c \in dom(G_1 \cup G_2)\}$$

and let

$$\sigma = mgu(\mathcal{A}).$$

Note that $G_1 \cup G_2$ is a 2-valued grammar. Then we define $G_1 \sqcup G_2$ as follows:

$$G_1 \sqcup G_2 = \sigma[G_1 \cup G_2].$$

If $\mathcal{A}$ is not unifiable, then $G_1 \sqcup G_2$ is undefined.

Example 8.14 *Let G_1 and G_2 be the following rigid Lambek grammars:*

$$\begin{array}{rcl} G_1 : a & \mapsto & s/x, \\ b & \mapsto & x, \\ G_2 : b & \mapsto & y\backslash s, \\ c & \mapsto & y. \end{array}$$

Then

$$\begin{array}{rcl} G_1 \sqcup G_2 : a & \mapsto & s/(y\backslash s), \\ b & \mapsto & y\backslash s, \\ c & \mapsto & y. \end{array}$$

Obviously, from definition 8.13, we have

Lemma 8.15 *If $G_1 \sqcup G_2$ exists, then $G_1 \sqsubseteq G_1 \sqcup G_2$ and $G_2 \sqsubseteq G_1 \sqcup G_2$.*

Proposition 8.16 (Kanazawa, 1998) *Let $G_1, G_2 \in \mathcal{G}_{rigid}$. If $\{G_1, G_2\}$ has un upper bound, then $G_1 \sqcup G_2$ exists and it's the least upper bound of $\{G_1, G_2\}$.*

Proof. (See [Kan98]).

9 Learning Rigid Lambek Grammars from Structures

In the present chapter we will explore a model of learning for Rigid Lambek Grammars based on *positive structured data*. In addition to the standard model where sentences are presented to the learner as flat sequences of words, in this somewhat enriched model, strings come with additional information about their "deep structure". Following the approach sketched in section 6, largely indebted with Tiede's study on proof trees in Lambek calculus as grammatical structures for Lambek grammars (see [Tie99]), in our model each sentence comes to the learner with a structure in the form of a *proof tree structure* as extensively described in section 6.

Formally, given a finite alphabet Σ, we will present a learning algorithm for the grammar system $\langle \mathcal{G}_{rigid}, \Sigma^P, \mathrm{PL} \rangle$: that is to say, samples to which the learner is exposed to are proof-tree structures over the alphabet Σ, and guesses are made about the set of rigid Lambek grammars that can generate such a set of structures.

We follow the advice of Kanazawa (see [Kan98]) who underlines how such an approach, which turns out to be quite logically independent from an approach based on flat strings of words, seems to make the task of learning easier but doesn't trivialize it. If, on one hand, in the process of learning from structures the learner is provided with more information, on the other hand the criterion for successful learning is stricter. It is not sufficient that the string language of G contains exactly the *yields* of the structures in the input sequence, the learning function is required to converge to a grammar G that generates all the grammatical *structures* which appear in the input sequence. We could say that the learning function must converge to a grammar that is both weakly and strongly equivalent to the grammar that generated the input samples.

Clearly, from a psycholinguistic point of view, both learning from flat strings and from proof tree structures are quite unrealistic models of first language acquisition by human beings. In the first case, experimental evidences (see [Pin94]) show that children can't acquire a language simply by passively listening to flat strings of words. First of all, we can think that prosody (or punctuation, in written text) can provide "structural" information to the children on the syntactic bracketing of the sentences she is exposed to (although they do not always coincide) and it is known that prosody is needed to learn a language for a child. Furthermore, another interesting evidence of the fact that a child needs something more to learn her mother tongue is given by the fact that no children can improve their grammatical skills during the early stages of their language acquisition process by watching TV: it seems very likely they need "richer data" than simple sentences uttered by an adult. Some researchers (see [Tel99]) hypothesize this additional information comes to the children as the semantic content of the first sentences she is exposed to, whose she could have a first, primitive grasp through first sensory-motor experiences.

On the other hand, it is also highly unlikely that a child can have access to something like a proof tree structure of the sentence she is exposed to. Our belief is that a good formal model for the process of learning should rely on something "halfway" between flat strings of words and highly structured and complete information coming from the proof tree structure of the sentence. However, since, as we've already seen in section 7.2, proof tree structures

provide a very natural support for a Montague-like semantics, we think that our model for learning a rigid Lambek grammar from structured data represents a first, simple but meaningful approximation of a more plausible model of learning.

In any case, even though in most of real-world applications only unstructured data are available, we are often interested not only in the sentences that a grammar derives, but also in derivation strings that grammar assigns to sentences. That is, we generally want a grammar that makes structural sense.

9.1 Grammatical Inference as Unification

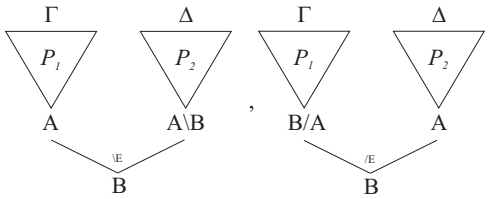
We set our inquiry over the learnability for rigid Lambek grammars in the more general logical framework of the Theory of Unification. We will stick to the approach described in [Nic99] based on the attempt to reduce the process of inferring a categorial grammar to the problem of unifying a set of terms. This approach establishes a fruitful connection between Inductive Logic Programming techniques and the field of Grammatical Inference, a connection that has already been proved successful in devising efficient algorithms to infer k-valued CCGs from positive structured data (see [Kan98]). Our aim is to exploit as much as possible what has already been done in this direction by exploring the possibility of adapting existing algorithms for CCGs to rigid Lambek grammars.

9.2 Argument Nodes and Typing Algorithm

Our learning algorithm is based on a process of labeling for the nodes of a set of proof tree structures. We introduce here the notion of *argument node* for a normal form proof tree. We will be a bit sloppy in defining such a notion, and sometimes we will use the same notation to indicate a node and the type it's labeled by, when this doesn't engender confusion, and much will be left to the graphical" interpretation of trees and their nodes. However, we can always think of a node as a De Bruijn-like object (see [dB72]) without substantially affecting the meaning of what will be proved.

Definition 9.1 *Let $\mathcal{P}$ be a normal form partial parse tree. Let's define inductively the set $Arg(\mathcal{P})$ of argument nodes of $\mathcal{P}$. There are three cases to consider:*

- *$\mathcal{P}$ is a single node labeled by a type x, which is the only member of $Arg(\mathcal{P})$.*
- *$\mathcal{P}$ looks like one of the following*



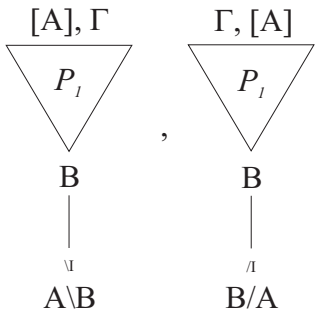
then in the first case

$$Arg(\mathcal{P}) = \{Root(\mathcal{P})\} \cup Arg(\mathcal{P}_1) \cup Arg(\mathcal{P}_2) - \{Root(\mathcal{P}_2)\},$$

and in the second case

$$Arg(\mathcal{P}) = \{Root(\mathcal{P})\} \cup Arg(\mathcal{P}_1) \cup Arg(\mathcal{P}_2) - \{Root(\mathcal{P}_1)\}.$$

- *$\mathcal{P}$ looks like one of the following*



then $Arg(\mathcal{P}) = Arg(\mathcal{P}_1)$.

The following proposition justifies our interest for argument nodes for a normal form proof tree structure:

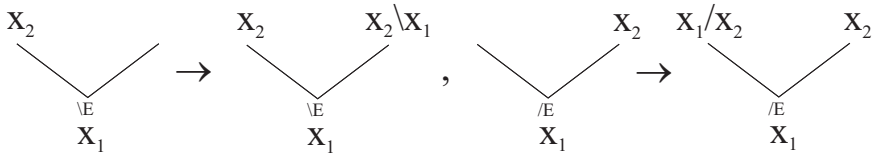
Proposition 9.2 *Let t be a well formed normal form proof tree structure. If each argument node is labeled, then any other node in t can be labeled with one and only one type.*

Proof. We prove that, once argument nodes are labeled, any other node can be labeled, by providing a typing algorithm; uniqueness of typing follows from the rules applied.

By induction on the height h of t:

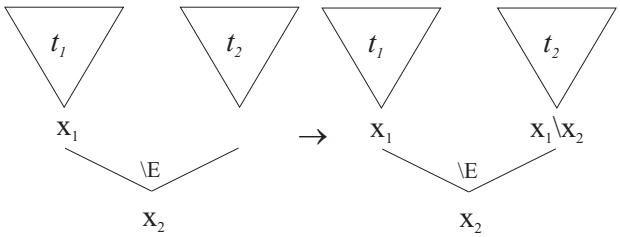
Induction Basis. There are two cases to consider:

1. $h = 0$. Trivially, by definition 9.1, t is a single argument node, the result of the application of a single axiom rule $[ID]$ and by definition it's already typed.
2. $h = 1$. Then t must be the result of a single application of a $[/E]$ or $[\backslash E]$ rule. By hypothesis and definition 9.1, its two argument nodes are labeled with, say, x_1 and x_2, and the remaining node must be labeled according to one of the following rules:



Induction Step. Let t be a normal form proof tree structure of height $h > 1$. There are 3 cases to consider:

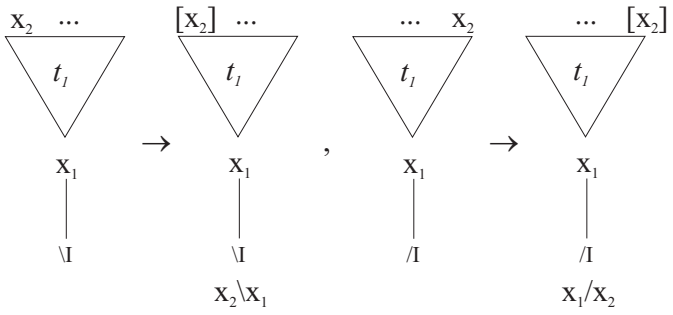
1. $t \equiv \backslash E(t_1, t_2)$. Since, by hypothesis, each node in $Arg(t) = \{Root(t)\} \cup Arg(t_1) \cup Arg(t_2) - \{Root(t_2)\}$, is labeled, then also $Root(t)$ is labeled with, say, x_2. For the same reason, any node of $Arg(t_1)$ is labeled, too, and so, by induction hypothesis, t_1 is fully (and uniquely) labeled. In particular its root is labeled with, say, x_1. Since t is well formed, t_2 cannot be the result of the application of a $[/I]$ rule, and since t is normal, t_2 cannot be the result of the application of a $[\backslash I]$ rule, so its root node is an argument node of its, too. By hypothesis, each node in $Arg(t_2) - \{Root(t_2)\}$ has a type, so we can apply the following rule:



and t_2 has all of its argument nodes (uniquely) labeled. So, by induction hypothesis, its fully and uniquely labeled, and so is t.

2. $t \equiv /E(t_1, t_2)$. Analogous to case 1.

3. $t \equiv \backslash I(t_1)$ or $t \equiv /I(t_1)$. By definition, $Arg(t) = Arg(t_1)$, then by hypothesis, any argument node in t_1 is labeled. Then, by induction hypothesis, t_1 is fully (and uniquely) labeled, and since t is well-formed, there must be at least two undischarged leaves in t_1. So t can be fully labeled according, respectively, to the following rules:



where x_2 labels, respectively, the leftmost and the rightmost undischarged leaf.

The proof of the previous proposition has implicitly defined an algorithm for labeling in the most general way the nodes of a normal form proof tree structure.

Definition 9.3 *A* principal parse *of a proof tree structure t is a partial parse tree $\mathcal{T}$ of t, such that for any other partial parse tree $\mathcal{T}'$ of t, there exists a substitution σ such that, if a node of t is labeled by type A in $\mathcal{T}$, it's labeled by $\sigma(A)$ in $\mathcal{T}'$.*

From the proof of proposition 9.2 it's easy to devise an algorithm to get a principal parse for any well formed normal form proof tree structure.

Principal Parse Algorithm

- **Input:** a well formed normal form proof tree structure t;
- **Output:** a principal parse $\mathcal{T}$ of t in a Lambek grammar G.

Step 1. Label with distinct variables each argument node in t;

Step 2. Compute the types for the remaining nodes according to the rules described in the proof of proposition 9.2.

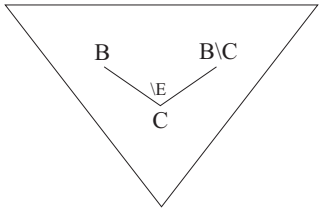
Obviously, this algorithm always terminates. If $\mathcal{T}$ is the resulting parse, we can easily prove it's principal. If $\mathcal{T}'$ is another parse for t, let's define a substitution σ in the following way: for each variable $x \in Var(G)$, find the (unique, for construction) node in $\mathcal{T}$ labeled by x, and let $\sigma(x)$ be the type labeling the same node in $\mathcal{T}'$. By induction on $A \in Tp(G)$ (where $Tp(G)$ is the set of all subtypes appearing in a Lambek Grammar G), we prove that

if A labels a node of $\mathcal{T}$, $\sigma(A)$ labels the corresponding node of $\mathcal{T}'$.

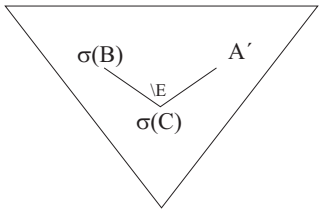
Induction Basis. If $A \in Var$, this holds by definition.

Induction Step. Let $A = B\backslash C$ labels a node of $\mathcal{T}$. Then the relevant part of $\mathcal{T}$ must look like one of the following cases:

- First case:

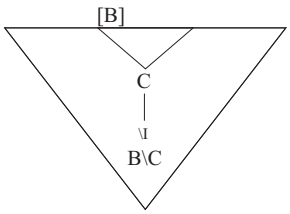


By induction hypothesis, the corresponding part of $\mathcal{T}'$ looks like:

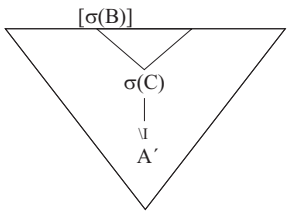


Then $A' = \sigma(B)\backslash\sigma(C) = \sigma(B\backslash C) = \sigma(A)$.

- Second case:



By induction hypothesis, the corresponding part of $\mathcal{T}'$ looks like:



Then $A' = \sigma(B)\backslash\sigma(C) = \sigma(B\backslash C) = \sigma(A)$.

The case $A = C/B$ is entirely similar, thus completing the induction.

It follows that if a node of $\mathcal{T}$ is labeled by A, then the corresponding node of $\mathcal{T}'$ is labeled by $\sigma(A)$. That is to say, with a small abuse of notation, $\mathcal{T}' = \sigma(\mathcal{T})$.

9.3 RLG Algorithm

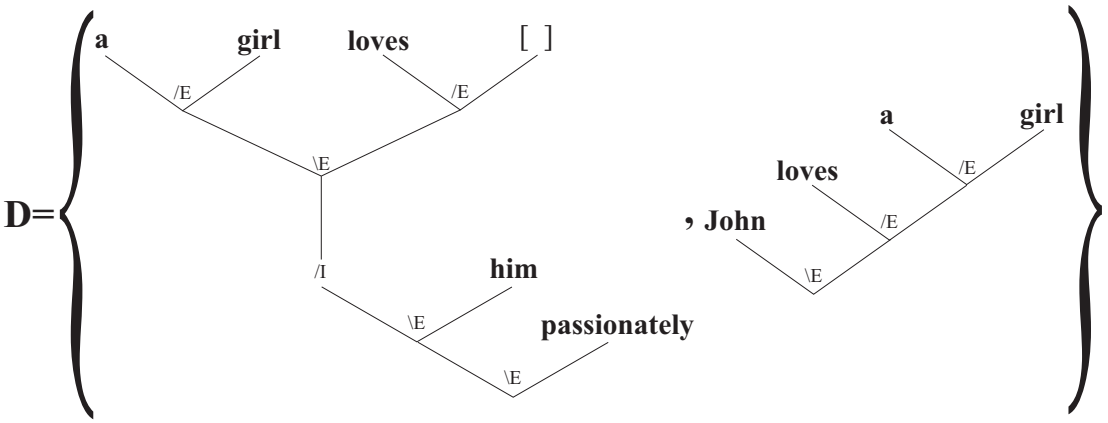
Our algorithm (called RLG from Rigid Lambek Grammar) takes as its input a finite set D of proof tree structures over a finite alphabet Σ and returns a rigid Lambek grammar G over the same alphabet whose structure language contains (properly) D, if it exists; a correct statement that there's no such a rigid Lambek grammar otherwise.

Our algorithm is based on the type algorithm described in section 9.2 and on the unification algorithm described in section 8.2.

RLG Algorithm.

- **input:** a finite set D of proof tree structures.
- **output:** a rigid Lambek grammar G such that $D \subset \mathrm{PL}(G)$, if there is one.

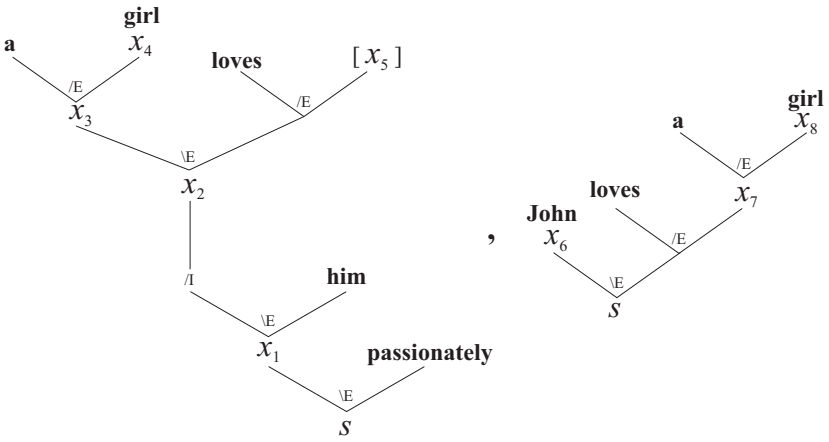
We illustrate the algorithm using the following example:



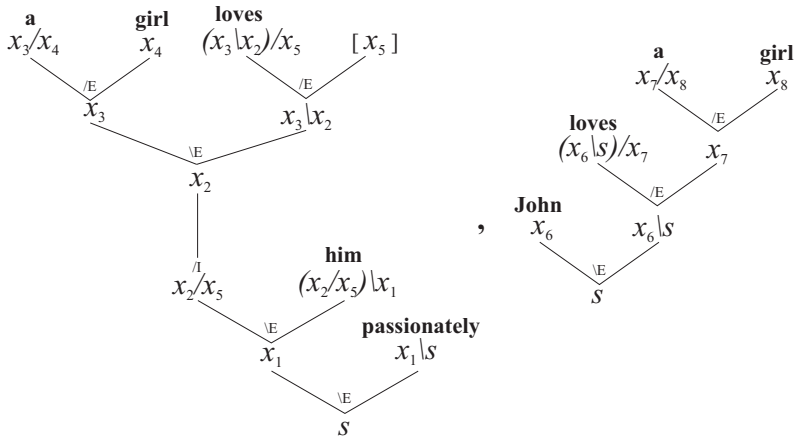
Step 1. Normalize all the proof tree structures in D, if they are not normal, according to the rules described in section 6.3.

Step 2. Assign a type to each node of the structure in D as follows:

1. Assign s to each root node.
2. Assign distinct variables to the argument nodes.



3. Compute types for the remaining nodes according to the rules described in proposition 9.2.



Step 3. Collect the types assigned to the leaf nodes into a grammar $GF(D)$ called the *general form* induced by D. In general, $GF(D) : c \mapsto A$ if and only if the previous step assigns A to a leaf node labeled by symbol c.

$$\begin{aligned} GF(D) : \textbf{passionately} &\mapsto x_1\backslash s \\ \textbf{him} &\mapsto (x_2/x_5)\backslash x_1 \\ \textbf{a} &\mapsto x_3/x_4, x_7/x_8 \\ \textbf{girl} &\mapsto x_4, x_8 \\ \textbf{loves} &\mapsto (x_3\backslash x_2)/x_5, (x_6\backslash s)/x_7 \\ \textbf{John} &\mapsto x_6 \end{aligned}$$

Step 4. Unify the types assigned to the same symbol. Let $\mathcal{A} = \{\{A \mid GF(D) : c \mapsto A\} \mid c \in dom(GF(D))\}$, and compute $\sigma = mgu(\mathcal{A})$. The algorithm fails if unification fails.

$$\sigma = \{x_7 \mapsto x_3, x_8 \mapsto x_4, x_6 \mapsto x_3, x_2 \mapsto s, x_5 \mapsto x_3\}$$

Step 5. Let $RLG(D) = \sigma[GF(D)]$.

$$\begin{aligned} RLG(D) : \textbf{passionately} &\mapsto x_1\backslash s \\ \textbf{him} &\mapsto (s/x_3)\backslash x_1 \\ \textbf{a} &\mapsto x_3/x_4 \\ \textbf{girl} &\mapsto x_4 \\ \textbf{loves} &\mapsto (x_3\backslash s)/x_3 \\ \textbf{John} &\mapsto x_3 \end{aligned}$$

Our algorithm is based on the "principal parse algorithm" described in the previous section, which has been proved to be correct and terminate, and the unification algorithm described in section 8.2. The result is, intuitively, the most general rigid Lambek Grammar which can generate all the proof tree structures appearing in the input sequence.

9.4 Properties of RLG

In the present section we prove some properties of the RLG algorithm that will be helpful to study its behaviour in the limit.

The following lemma is almost trivial but it will play an important role in the convergence proof for the RLG algorithm. It simply states that the tree language of the grammar inferred just after the labeling of the structures properly contains the sample structures.

Lemma 9.4 *Let D be the input set of proof tree structures for the RLG algorithm. Then the set of the proof tree structures generated by the 'general form' grammar contains properly D. That is, $D \subset \mathrm{PL}(GF(D))$.*

Proof. Let $D = \{T_1, \ldots, T_n\}$. The labeling of the nodes of the structures in D that precedes the construction of $GF(D)$ in fact forms a parse tree $\mathcal{P}_i$ of $GF(D)$ for each structure T_i in D. This shows $D \subseteq PL(GF(D))$. The proper inclusion follows trivially from the fact that D is by hypothesis a finite set, while PL(G), the set of proof tree structures generated by a Lambek grammar G, is always infinite.

Lemma 9.5 *Each variable $x \in Var(GF(D))$ labels a unique node in a unique parse tree of D.*

Proof. Obviously, by construction, if $x \in Var(GF(D))$, then there must be an $i \in \mathbb{N}$ such that x labels one of the nodes of a parse tree $\mathcal{P}_i$. Since, by construction, for each $i \neq j$ the sets of variables that label $\mathcal{P}_i$ are disjoint, x appears in one and only one $\mathcal{P}_i$. Besides, since variables are assigned only during the first phase of the type-assignment process of our algorithm, again by construction each variable labels only one node in the deduction tree.

The following lemma makes explicit the relation between the grammar inferred just after the labeling of the structures in the algorithm RLG, and the structure language of the rigid grammar we are trying to infer.

Lemma 9.6 *Let D be a finite set of proof tree structures. Then, for any Lambek grammar G, the following are equivalent:*

(i) $D \subseteq$ PL(G)

(ii) *There is a substitution σ such that $\sigma[GF(D)] \subseteq G$.*

Proof. (ii)⇒(i). Suppose there is a substitution σ such that $\sigma[GF(D)] \subseteq G$. Then, from proposition 6.32, we have that PL($GF(D)$) $\subseteq$ PL(G). This, together with lemma 9.4 proves (i).

(i) ⇒(ii). Let $D = \{T_1, \ldots, T_n\}$ and let $\mathcal{P}_i$ be $GF(D)$'s parse of T_i for $1 \leq i \leq n$. Assume $D \subseteq$ PL(G). Then G has a parse $\mathcal{Q}_i$ of each T_i. Define a substitution σ as follows: for each variable $x \in Var(GF(D))$, find a (unique, due to lemma 9.5) $\mathcal{P}_i$ that contains a (unique, again due to lemma 9.5) node labeled by x, and let $\sigma(x)$ be the type labeling the corresponding node of $\mathcal{Q}_i$. We show that

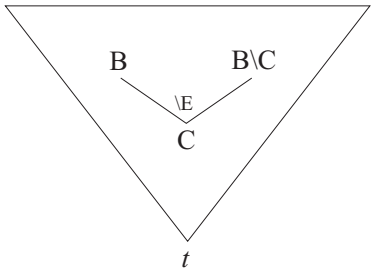
if A labels a node of some $\mathcal{P}_i$, then $\sigma(A)$ labels the corresponding node of $\mathcal{Q}_i$.

Proof. By induction on $A \in Tp(GF(D)) = \{T \mid T \text{ is a subtype of some } B \in range(GF(D))\}$):

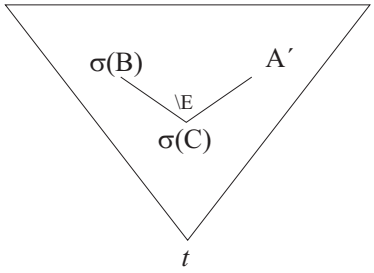
Induction basis. If $A \in Var$, this holds by definition. If $A = t$, then any node labeled by A in $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ is the root node of some $\mathcal{P}_i$. Since $\mathcal{Q}_i$ is a parse tree of G, the root node of $\mathcal{Q}_i$ must be labeled by t.

Induction step. Let $A = B\backslash C$ labels a node of $\mathcal{P}_i$. Then the relevant part of $\mathcal{P}_i$ must look like one of the two following cases:

- First case

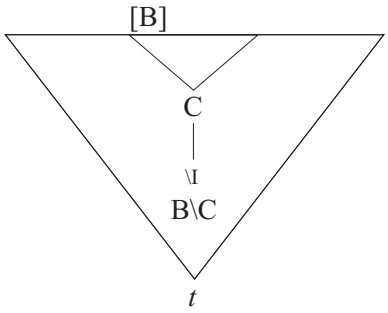


By induction hypothesis, the corresponding part of $\mathcal{Q}_i$ looks like:

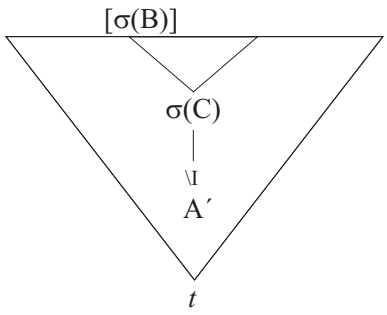


Then $A' = \sigma(B)\backslash\sigma(C) = \sigma(B\backslash C) = \sigma(A)$.

- Second case



By induction hypothesis, the corresponding part of $\mathcal{Q}_i$ looks like:



Then again $A' = \sigma(B)\backslash\sigma(C) = \sigma(B\backslash C) = \sigma(A)$.

The case $A = C/B$ is entirely similar, thus completing the induction. It follows that if $GF(D) : c \mapsto A$, then $G : c \mapsto \sigma(A)$. Therefore, $\sigma[GF(D)] \subseteq G$.

The following proposition establishes an "if and only if" relation between the inclusion of our set of positive samples D in a tree language generated by a rigid grammar G and the successful termination of the RLG algorithm when it has D as its input set. Even more, we have that the rigid grammar inferred by the algorithm is not "larger" than the rigid grammar G.

Proposition 9.7 *Let D be a finite set of proof tree structures. Then, for any rigid grammar G, the following are equivalent:*

(i) $D \subseteq \mathrm{PL}(G)$*;*

(ii) *$RLG(D)$ exists and $RLG(D) \sqsubseteq G$ (equivalently, there is a substitution τ such that $\tau[RLG(D)] \subseteq G$).*

Proof. (ii) $\Rightarrow$ (i) follows from lemma 9.6 and the fact that $RLG(D)$ is a substitution instance of $GF(D)$.

(i) $\Rightarrow$ (ii). Assume that G is a rigid grammar such that $D \subseteq PL(G)$. By lemma 9.6 there is a substitution σ such that $\sigma[GF(D)] \subseteq G$. Since G is a rigid grammar, $\sigma[GF(D)]$ is also a rigid grammar. Then σ unifies the family $\mathcal{A} = \{\{A \mid GF(D) : c \mapsto A\} \mid c \in dom(GF(D))\}$. This means that $RLG(D)$ exists and $RLG(D) = \sigma_0[GF(D)]$, where $\sigma_0 = mgu(\mathcal{A})$. Then there is a substitution τ such that $\sigma = \tau \circ \sigma_0$. Therefore, $\tau[RLG(D)] = \tau[\sigma_0[GF(D)]] = (\tau \circ \sigma_0)[GF(D)] = \sigma[GF(D)]$. By assumption, $\sigma[GF(D)] \subseteq G$, so $\tau[RLG(D)] \subseteq G$.

Corollary 9.8 *Let D_1 and D_2 be two finite sets of proof tree structures such that $D_1 \subseteq D_2$. If $RLG(D_2)$ exists, $RLG(D_1)$ also exists and $RLG(D_1) \sqsubseteq RLG(D_2)$ and* $\mathrm{PL}(RLG(D_1)) \subseteq \mathrm{PL}(RLG(D_2))$.

Proof. Immediate from proposition 9.7, noting that if $D_1 \subseteq D_2$, then $\{G \in \mathcal{G}_{rigid} \mid D_1 \subseteq \mathrm{PL}(G)\} \supseteq \{G \in \mathcal{G}_{rigid} \mid D_2 \subseteq \mathrm{PL}(G)\}$.

Definition 9.9 *Let φ_{RLG} be the learning function for the grammar system $\langle G_{rigid}, \Sigma^P, \mathrm{PL}\rangle$ defined as follows:*[1]

$$\varphi_{RLG}(\langle T_0, \ldots, T_n\rangle) \simeq RLG(\{T_0, \ldots, T_n\}).$$

Thanks to previous propositions and lemmas we are able to prove the convergence for the RLG algorithm:

Theorem 9.10 *φ_{RLG} learns $\mathcal{G}_{rigid}$ from structures.*

Proof. We prove that φ_{RLG} learns the class of rigid Lambek grammars from proof tree structures.

Let G be any rigid Lambek grammar and let $\langle T_i\rangle_{i\in\mathbb{N}}$ be an infinite sequence enumerating $\mathrm{PL}(G)$. For each $i \in \mathbb{N}$, $\{T_0, \ldots, T_i\} \subseteq \mathrm{PL}(G)$, so by proposition 9.7 $\varphi_{RLG}(\langle T_0, \ldots, T_i\rangle) = RLG(\{T_0, \ldots, T_i\})$ is defined and

$$\varphi_{RLG}(\langle T_0, \ldots, T_i\rangle) \sqsubseteq \varphi_{RLG}(\langle T_0, \ldots, T_{i+1}\rangle),$$

by corollary 9.8, and

$$\varphi_{RLG}(\langle T_0, \ldots, T_i\rangle) \sqsubseteq G.$$

Since, by corollary 6.41, there are only finitely many Lambek grammars $G'' \sqsubseteq G$, φ_{RLG} must converge on $\langle T_i\rangle_{i\in\mathbb{N}}$ to some G'. Then $\mathrm{PL}(G) = \{T_i \mid i \in \mathbb{N}\} \subseteq \mathrm{PL}(G')$. Since $G' \sqsubseteq G$, by proposition 6.32, $\mathrm{PL}(G') \subseteq \mathrm{PL}(G)$. Therefore, $\mathrm{PL}(G') = \mathrm{PL}(G)$.

When RLG is applied successively to a sequence of increasing set of proof tree structures $D_0 \subset D_1 \subset D_2 \subset \cdots$, it is more efficient to make use of the previous value $\mathrm{RLG}(D_{i-1})$ to compute the current value $\mathrm{RLG}(D_i)$.

Definition 9.11 *If G is a rigid Lambek grammar and D is a finite set of proof tree structures, then let*

$$RLG^{(2)}(G, D) \simeq G \sqcup RLG(D).$$

Lemma 9.12 *If D_1 and D_2 are two finite sets of proof tree structures,*

$$RLG^{(2)}(RLG(D_1), D_2) \simeq RLG(D_1 \cup D_2).$$

[1] Recall that the symbol $\simeq$ means that either both sides are defined and are equal, or else both sides are undefined

Proof. (See [Kan98]). Suppose that $RLG^{(2)}(RLG(D_1), D_2)$ is defined. By lemma 8.15, $RLG(D_1) \sqsubseteq RLG^{(2)}(RLG(D_1), D_2)$ and $RLG(D_2) = RLG^{(2)}(RLG(D_1), D_2)$. This implies that $D_1 \cup D_2 \subseteq \mathrm{PL}(RLG^{(2)}(RLG(D_1), D_2))$, so by proposition 9.7, $RLG(D_1 \cup D_2)$ exists and $RLG(D_1 \cup D_2) \sqsubseteq RLG^{(2)}(RLG(D_1), D_2)$.

Suppose now that $RLG(D_1 \cup D_2)$ is defined. By corollary 9.8, $RLG(D_1)$ and $RLG(D_2)$ exist and $RLG(D_1) \sqsubseteq RLG(D_1 \cup D_2)$ and $RLG(D_2) \sqsubseteq RLG(D_1 \cup D_2)$. Then $RLG(D_1 \cup D_2)$ is an upper bound of $\{RLG(D_1), RLG(D_2)\}$. By proposition 8.16, $RLG(D_1) \sqcup RLG(D_2) = RLG^{(2)}(RLG(D_1), D_2)$ exists and $RLG^{(2)}(RLG(D_1), D_2) \sqsubseteq RLG(D_1 \cup D_2)$.

Thus it has been proved that if one of $RLG^{(2)}(RLG(D_1), D_2)$ and $RLG(D_1 \cup D_2)$ is defined the other is defined and they are equal.

Proposition 9.13 *φ_{RLG} has the following properties:*

(i) *φ_{RLG} learns $\mathcal{G}_{rigid}$ prudently.*

(ii) *φ_{RLG} is responsive and consistent on $\mathcal{G}_{rigid}$.*

(iii) *φ_{RLG} is set-driven.*

(iv) *φ_{RLG} is conservative.*

(v) *φ_{RLG} is monotone increasing.*

(vi) *φ_{RLG} is incremental.*

Proof.

(i) Since $range(\varphi_{RLG}) \subseteq \mathcal{G}_{rigid}$, φ_{RLG} learns $\mathcal{G}_{rigid}$ prudently.

(ii) If $D \subseteq L$ for some $L \in \mathcal{PL}_{rigid}$, then by proposition 9.7 $RLG(D)$ exists and by lemma 9.6 $D \subseteq \mathrm{PL}(RLG(D))$. This means that φ_{RLG} is responsive and consistent on $\mathcal{G}_{rigid}$.

(iii) φ_{RLG} is set-driven by definition.

(iv) Let $T \in \mathrm{PL}(RLG(D))$. Then $D \cup \{T\} \subset \mathrm{PL}(RLG(D))$. By proposition 9.7, $RLG(D \cup \{T\})$ exists and $RLG(D \cup \{T\}) \sqsubseteq RLG(D)$. By corollary 9.8 we have also $RLG(D) \sqsubseteq RLG(D \cup \{T\})$. This shows that φ_{RLG} is conservative.

(v) Trivial from corollary 9.8.

(vi) Define a computable function $\psi : \mathcal{G}_{rigid} \times \Sigma^P \to \mathcal{G}_{rigid}$ as follows:

$$\psi(G, T) \simeq \begin{cases} RLG^{(2)}(G, \{T\}) & \text{if } G \in \mathcal{G}_{rigid} \text{ and } RLG(\{T\}) \text{ is defined,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Then by lemma 9.12, $\varphi_{RLG}(\langle T_0, \ldots, T_{i+1} \rangle) \simeq \psi(\varphi_{RLG}(\langle T_0, \ldots, T_i \rangle), T_{i+1})$.

10 Conclusion and Further Research

This work aims at making a further step in the direction of bridging the gap which still separates any formal/computational theory of learning from a meaningful formal linguistic theory.

We have introduced the basic notions of Formal Learnability Theory as first formulated by E.M. Gold in 1967, and of Lambek Grammars, which appeared for the first time in an article of 1958.

The former, which is one of the first completely formal descriptions of the process of grammatical inference, after an initial skepticism about its effective applicability, is at present to object of a renewed interest due to some meaningful and promising learnabiluty results.

Even the latter, long neglected by the linguistic community, is experiencing a strong renewed interest as a consequence of recent linguistics achievements which point at formal grammars completely lexicalized, as Lambek grammars are. Even if they're still far from being the ultimate formal device for the formalization of human linguistics competence, they're universally looked at as a promising tool for further developments of computational linguistics.

In the present work we've drawn the attention to a particular class of Lambek grammars called *rigid Lambek grammars*, and we've proved that they are learnable in Gold's framework from a structured input. We've used most recent results by Hans-Joerg Tiede for formally define our notion of *structure* for a sentence: he has recently proved that the proof tree language generated by a Lambek grammar strictly contains the tree language generated by context-free grammars. His notion of a proof as the grammatical structure of a sentence in a categorial grammar is also useful in providing a natural support to a Montagovian semantics for that sentence. Therefore, our choice for a structured input for our learning algorithm in the form of proof tree structures is not gratuitous, but it's coherent with the mainstream of (psycho-)linguistics theories about first language learning which stress the importance of providing the learner with informatioannly and semantically rich input in the process of her language acquisition.

We believe it to be a partial but meaningful result, which once more shows how versatile and powerful can be this learning theory, once neglected because it was widely held that it couldn't but account for the learnability of most trivial classes of grammars.

Much is left to be done along many directions. First of all, there's still no real theory of rigid, or k-valued, Lambek grammars: we still know very few formal properties of such grammars which seem to have an undisputable linguistic interest. We still lack, for example, a hierarchy theorem for languages generated by k-valued Lambek grammars.

Another important point which is still unanswered lies in the decidibility for $PL(G_1) \subseteq PL(G_2)$ for G_1, G_2 Lambek grammars, that is deciding whether the tree language generated by a grammar is contained in the tree language generated by another one, for any two grammars. Such a question is decidable for the non-associative variant of Lambek grammars. Proving this question decidable would allow as to very esaily devise a learning algorithm for

k-valued Lambek grammars.

Our learnability result is in our opinion a first step toward a more convincing and linguistically plausible model of learning for k-valued Lambek grammars from less and less structurally rich input. Needless to say, learning from such an informationally rich input like proof-tree structures are hardly has any linguistic plausibility. On the other hand the deep connections between proof tree structures for a sentence in Lambek grammars and its "Montague-like" semantics seems to address to a more convincing model for learning based both on syntactic and semantic information.

References

[Ang80] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[BB75] Lenore Blum and Manuel Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[BH64] Yehoshua Bar-Hillel. *Language and Information*. Addson-Wesley, Reading, 1964.

[BP90] Wojciech Buszkowski and Gerald Penn. Categorial grammars determined from linguistic data by unification. *Studia Logica*, 49:431–454, 1990.

[Bus86] Woiciech Buszkowski. Generative capacity of non-associative lambek calculus. *Bulletin of the Polish Academy of Science and Mathematics*, 1986.

[Cho56] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory IT-2*, 3:113–124, 1956.

[Cho75] Noam Chomsky. *Reflections on Language*. Pantheon, 1975.

[dB72] N. G. de Bruijn. Lambda calculus notation with nameless dummies. *Indagationes Mathematicae*, pages 381–392, 1972.

[Fit96] Melvin Fitting. *First-Order Logic and Automatic Theorem Proving*. Berlin: Springer, 1996.

[Ful88] M. Fulk. Saving the phenomenon: Requirements that inductive machines not contradict known data. *Information and Computation*, 79(3):193–209, 1988.

[Gaz88] Gerald Gazdar. Applicability of indexed grammars to natural languages. In Uwe Reyle and Christian Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. Dordrecht:Reidel, 1988.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[GS84] Ferenc Gécseg and Magnus Steinby. *Tree Automata*. Akadémiai Kiadó, 1984.

[Gue83] Irène Guesserian. Pushdown tree automata. *Mathematical Systems Theory*, 16(4):237–263, 1983.

[JORS99] Sanjay Jain, Daniel N. Osherson, James S. Royer, and Arun Sharma. *Systems that Learn*. MIT Press, Cambridge, Massachusetts, second edition, 1999.

[Kan98] Makoto Kanazawa. *Learnable Classes of Categorial Grammars*. Center for the Study of Language and Information (CSLI), Stanford, 1998.

[Koz97] Dexter Kozen. *Automata and Computability*. Berlin: Springer, 1997.

[Lam58] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

[Mon97] Richard Montague. The proper treatment of quantification in ordinary english. In Jakko Hintikka, editor, *Approaches to Natural Language*, pages 221–242. Reidel, 1997.

[Moo97] Michael Moortgat. Categorial type logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. North Holland, Amsterdam, 1997.

[Nic99] Jacques Nicolas. Grammatical inference as unification. Technical Report 3632, IRISA, Unitede Recherche INRIA Rennes, 1999.

[OGL95] Daniel N. Osherson, Lila R. Gleitmann, and Mark Liberman, editors. *An Invitation to Cognitive Science*, volume 1, "Language". The MIT press, Massachusetts Institute of Technology, Cambridge, Massachusetts, second edition, 1995.

[OWdJM97] Daniel N. Osherson, Scott Weinstein, Dick de Jongh, and Eric Martin. Formal learning theory. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*. North Holland, Amsterdam, 1997.

[Pen97] Mati Pentus. Product-free lambek calculus and context-free grammars. *The Journal of Symbolic Logic*, 62(2):648–660, 1997.

[Pin94] Steven Pinker. *The Language Instict*. Penguin Press, 1994.

[Ret96] Christian Retoré. Proof-nets for the lambek calculus - an overview. In Michele Abrusci and Claudia Casadio, editors, *Proceedings of the 1996 Roma Workshop "Proofs and Linguistic Categories - Applications of Logic to the Analysis and Implementation of Natural Language*, pages 241–262, Bologna, April 1996. CLUEB.

[Roo91] Dirk Roorda. *Resource Logics: Proof-Theoretical Investigations*. PhD thesis, University of Amsterdam, 1991.

[Shi90] T. Shinohara. Inductive inference from positive data is powerful. In *The 1990 Workshop on Computational Learning Theory*, pages 97–110, San Mateo, California, 1990. Morgan Kaufmann.

[Ste93] Mark Steedman. Categorial grammar. *Lingua*, 90:221–258, 1993.

[Tel99] Isabelle Tellier. Rôle de la compositionnalité dan l'acquisition d'une langue. In *Actes de CAP99*, pages 107–114, Palaiseau, 1999.

[Tha67] James W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer Systems Scinces*, pages 317–322, 1967.

[Tie99] Hans-Joerg Tiede. *Deductive Systems and Grammars: Proofs as Grammatical Structures*. PhD thesis, Indiana University, July 1999.

[vB87] Johan van Benthem. Logical syntax. *Theoretical Linguistics*, 14(2/3):119–142, 1987.

[Wan93] Heinrich Wansing. *The Logic of Information Structures*. Berlin: Springer Verlag, 1993.

[Wri89] K. Wright. Identifications of unions of languages drawn from an identifiable class. In *The 1989 Workshop on Computational Learning Theory*, pages 328–333, San Mateo, California, 1989. Morgan Kaufmann.

[Zie89] Wojciech Zielonka. A simple and general method for solving the finite axiomatizability problems for Lambek's syntactic calculi. *Studia Logica*, 48(1):35–39, 1989.

Contents

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399