A Study of Learnability of Lambek Grammars from Typed Examples

D. Dudau Sofronie, I. Tellier

Grappa, Université Lille 3 & INRIA Futurs,
59 653 Villeneuve d'Ascq Cedex, France

Abstract

This paper presents learnability results from Typed Examples for some classes of Lambek Grammars, in the context of Gold's model of identification in the limit. A learning strategy is then presented and exemplified.

Key words: Lambek Grammars, learnability in Gold's model, Typed Examples

1 Introduction

Although categorial grammars have been known for a long time, the study of their learnability is only a recent issue. Kanazawa (9), continuing the work of Buzskowski and Penn (3), opened new ways of research in this domain. He proved that large subclasses of AB-Categorial Grammars (or CGs) are learnable in Gold's model of *identification in the limit* (8). These classes are called k-valued: they contain every CG assigning at most k distinct categories to each member of their vocabulary. For any $k \geq 1$, the class of k-valued CGs is learnable from Structural Examples (i.e. syntactic analysis structures where rules are preserved but intermediate categories are deleted) and from strings (or sentences). Unfortunately, the only tractable (polynomial) case is the learning of rigid (i.e. 1-valued) CGs from Structural Examples.

The question naturally arose of the adaptation of these results to known variants of k-valued Lambek Grammars (or LGs). But these results are not easily adaptable. It has been proved that it is possible to learn the class of rigid LGs from proof structures of a certain normal form (2) but, on the contrary, this

Email address: dudau@grappa.univ-lille3.fr,tellier@univ-lille3.fr (D. Dudau Sofronie, I. Tellier).

class is not learnable from strings alone (7). Thus, for LGs, learnability results crucially rely on the available input data. Another learnability result relies on additional restrictions on grammars (1).

We have introduced the concept of learnability from Typed Examples in the context of CGs (5). Typed Examples can be considered as intermediary input data, richer than strings but less informative than Structural Examples. They can also be interpreted as coming from semantic information. We identified interesting subclasses of CGs learnable from Typed Examples (6). But this result was a trivial consequence of the learnability of rigid CGs from strings. For LGs, the situation is different as rigid LGs are learnable from Structural Examples but not from strings. Furthermore, the learnability result for CGs was associated with a learning algorithm inspired by syntactic analysis procedures. This strategy can also be applied to LGs. For all these reasons, learnability from Typed Examples is worth being studying in the context of LGs.

After some preliminary definitions, this paper concentrates on subclasses of LGs. The learnability of this class in the Gold's model is proved and a learning strategy is proposed. It is illustrated in detail on an example.

2 Preliminaries

2.1 Categorial Grammars

Every categorial grammar share the same notion of categories we recall here.

Definition 1 (Categories) *Let $\mathcal{B}$ be a countably infinite set of basic categories containing a distinguished category $S \in \mathcal{B}$, called the axiom. We note $Cat(\mathcal{B})$ the term algebra built over the two binary symbols $/$, $\backslash$ which is the smallest set such that $\mathcal{B} \subset Cat(\mathcal{B})$ and for any $A, B \in Cat(\mathcal{B})$ we have: $/(A, B) \in Cat(\mathcal{B})$ and $\backslash(A, B) \in Cat(\mathcal{B})$* [1] *.*

Definition 2 (AB-Categorial Grammars and Lambek Grammars) *For every finite vocabulary Σ and for every set of basic categories $\mathcal{B}$ ($S \in \mathcal{B}$), a **categorial grammar** is a finite relation G over $\Sigma \times Cat(\mathcal{B})$.*
AB-Categorial Grammars (CGs) are categorial grammars where the syntactic rules take the form of two rewriting schemes: $\forall A, B \in Cat(\mathcal{B})$

- *FA (Forward Application) : $/(A, B)$ $A \rightarrow B$*

[1] for reasons that will become clear further, we give up here the classical notations B/A and $A\backslash B$: in our notation, terms $/(A, B)$ and $\backslash(A, B)$ are both functors whose first component A is the argument and whose second component B is the result

- *BA (Backward Application) :* $A \ \backslash(A,B) \rightarrow B$

The language generated by a CG G is: $L(G)=\{u_1 \ldots u_n \in \Sigma^* \mid \forall i \in \{1,\ldots,n\}$, $\exists A_i \in Cat(\mathcal{B})$ *such that* $\langle u_i, A_i\rangle \in G$ *and* $A_1,\ldots,A_n \rightarrow^* S\}$.
Lambek Grammars (LGs) are categorial grammars in which syntactic analysis is described by a logical calculus defined by:

- *axioms:*

$$[ID]\ A \vdash A\ ;$$

- *inference rules:*

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

$$[/L]\ \frac{\Gamma \vdash A \ \ \Delta, B, \Pi \vdash C}{\Delta, /(A,B), \Gamma, \Pi \vdash C} \quad [\backslash L]\ \frac{\Gamma \vdash A \ \ \Delta, B, \Pi \vdash C}{\Delta, \Gamma, \backslash(A,B), \Pi \vdash C}$$

where $A, B, C \in Cat(\mathcal{B})$ *and* Γ, Δ, Π *are finite concatenations of categories from* $Cat(\mathcal{B})$, $\Gamma \neq \emptyset$. *The language* $L(G)$ *generated by* G *is:* $L(G)=\{u_1 \ldots u_n \in \Sigma^* \mid \forall i \in \{1,\ldots,n\}$, $\exists A_i \in Cat(\mathcal{B})$ *such that* $\langle u_i, A_i\rangle \in G$ *and* $A_1,\ldots,A_n \vdash S\}$. *In the following, elements of* Σ *will be called words and elements of* $L(G)$ *will be called sentences.*

The variant of the Lambek calculus we will deal with is the associative one, without product and with non-empty antecedent. This variant is the one which has received the greatest linguistic attentions. The Lambek calculus can also be expressed with the help of natural deduction, but here we only use the sequent calculus which can be easier linked with parse algorithms.

Example 1 *Let* $\Sigma = \{a,$ *song, sings, John*$\}$ *be a vocabulary and let* G *be a Lambek Grammar over* $\Sigma \times Cat(\mathcal{B})$ *with* $\mathcal{B} = \{S, T, CN\}$ *the set of basic categories (where* T *stands for "term" and* CN *for "common noun"). G is defined by the following assignments:* $\{\langle a, /(CN, \backslash(/(T,S),S))\rangle$, $\langle song, CN\rangle$, $\langle sings, \backslash(T,S)\rangle$, $\langle sings, /(T, \backslash(T,S))\rangle$, $\langle John, T\rangle\}$. *G recognises sentences like "John sings" or "John sings a song", as displayed by the following proof:*

$$\dfrac{\dfrac{CN \vdash CN \quad [\backslash L]\dfrac{\dfrac{T \vdash T \quad [\backslash L]\dfrac{T \vdash T \quad S \vdash S}{T, \backslash(T,S) \vdash S}}{[/L]\dfrac{}{T, /(T, \backslash(T,S)), T \vdash S}}}{[/R]\dfrac{}{T, /(T, \backslash(T,S)) \vdash /(T,S)}} \quad S \vdash S}{[/L]\dfrac{}{T, \quad /(T, \backslash(T,S)),\ /(CN, \backslash(/(T,S),S)),\ CN \vdash S}}$$

$$John \qquad sings \qquad\qquad a \qquad\qquad song$$

It is easy to observe that this sentence is not recognized by the CG with the same assignment of categories. To be able to analyse this example in this context, a solution would be to assign an extra category to the word "sings", for example: $\backslash(T, /(T, S))$. The inference rules of the Lambek calculus simulate multiple category assignments. But the price to pay for this higher flexibility is a higher complexity of parse algorithms. Pentus proved that the membership problem for LGs is NP-complete (13), whereas it is polynomial for CGs.

We denote by $\mathcal{G}$ the class of CGs and by $\mathcal{LG}$ the class of LGs. For every non null integer k, the set of CGs (resp. LGs) assigning at most k distinct categories to each of its word is the class of k-valued CG (resp. LG) and is noted $\mathcal{G}_k$ (resp. $\mathcal{LG}_k$). When $k = 1$ the grammars are also called rigid.

2.2 Semantic Types

Montague (11) was the first to propose a typed logic to represent natural language semantics. The associated notion of semantic type became so forth classical. It is this (generalised) notion of type that will be considered here.

Definition 3 (Semantic Types) *Θ is a finite set of basic types which contains a distinguished type $t \in \Theta$. The set of all possible types $Types(\Theta)$ is the smallest set such that : $\Theta \subset Types(\Theta)$ and for any $u, v \in Types(\Theta)$, $(u, v) \in Types(\Theta)$. The type (u, v) is a functor expecting as argument the type u and providing as result the type v.*

Example 2 *The usual set of basic types is $\Theta = \{e, t\}$, where e is the type of elementary entities of an associated model and t is the type of truth values. In a logical-based semantic representation, identifiers for individuals like "John" can be represented by a logical constant of type e while common nouns like "man" and intransitive verbs like "run" both denote one place predicates, of type (e, t). Transitive verbs like "loves" denote two-place predicates, of type $(e, (e, t))$. Verbs like "sing" can have both a transitive and an intransitive use.*

There is a very close connection between categories used in categorial grammars and semantic types. Both are represented by binary terms. This connection can be characterized by the notion of **typing function**.

Definition 4 (Typing Function) *For any set of basic categories $\mathcal{B}$ ($S \in \mathcal{B}$) and any set of basic types Θ ($t \in \Theta$), a typing function h is a morphism from $Cat(\mathcal{B})$ to $Types(\Theta)$ satisfying the following conditions:*

(1) $h(S) = t$;
(2) $\forall A, B \in Cat(\mathcal{B})$: *if* $h(A) = h(B) \in \Theta$ *then* $A = B$;
(3) $\forall A, B \in Cat(\mathcal{B})$: $h(/(A, B)) = h(\backslash(A, B)) = (h(A), h(B))$.

As h is a morphism and $Cat(\mathcal{B})$ is built over the set $\mathcal{B}$, it is enough to define h on $\mathcal{B}$ to deduce its values on $Cat(\mathcal{B})$.

Example 3 *If we set: $h(T) = e, h(S) = t, h(CN) = (e,t)$, we define a typing function for the categories of the grammar of Example 1 perfectly compatible with its semantics, defined in Example 2. Note that a basic category (for example CN) can be associated with a non-basic type and that two different categories can correspond to an identical (non-basic, otherwise it would contradicts condition (2)) type as: $h(\backslash(T,S)) = (h(T), h(S)) = (e,t) = h(CN)$.*

The Principle of Compositionality asserts that the meaning of a sentence only depends of the meaning of its parts and of its syntactic structure (12) (where the "parts" are more or less assimilated to words). This principle is usually translated as a similarity of structure between syntactic and semantic trees. As categories and types are *lexicalised structures*, the typing function can be considered as the *lexicalised version of the Principle of Compositionality.*

Definition 5 (The h-Typed Language of a Lambek Grammars) *For any sets Σ, $\mathcal{B}$ and Θ, any LG $G \in \Sigma \times Cat(\mathcal{B})$ and any typing function h from $Cat(\mathcal{B})$ to $Types(\Theta)$, we define the h-*Typed Language *of G by: $TL_h(G) = \{\langle u_1, \tau_1\rangle ... \langle u_n, \tau_n\rangle \ | \forall i \in \{1, ..., n\} \exists A_i$ so that $\langle u_i, A_i\rangle \in G$, $\tau_i = h(A_i)$ and $A_1, ..., A_n \vdash S\}$. An element of $TL_h(G)$ is called a h-Typed Example of G.*

Example 4 *The h-Typed Example corresponding with the sentence analysed in Example 1 and the typing function of Example 3 is the following:*
$\langle John, e\rangle\langle sings, (e,(e,t))\rangle\langle a, ((e,t),((e,t),t))\rangle\langle song, (e,t)\rangle$

Definition 6 (Lengths) *For any category $A \in Cat(\mathcal{B})$ (resp. for any type $\tau \in Types(\Theta)$), the length of A, noted $|A|$ (resp. the length of τ noted $|\tau|$) is the number of basic categories (resp. of basic types) it contains:*

- *if $A \in \mathcal{B}$ (resp. $\tau \in \Theta$) then $|A| = 1$ (resp. $|\tau| = 1$);*
- *$\forall A, B \in Cat(\mathcal{B})$ (resp. $\forall u, v \in Types(\Theta)$) $|/(A,B)| = |\backslash(A,B)| = |A| + |B|$ (resp. $|(u,v)| = |u| + |v|$).*

Lemma 1 *For any set of basic categories $\mathcal{B}$, any set of basic types Θ and any typing function h from $Cat(\mathcal{B})$ to $Types(\Theta)$ we have:*
$\forall C \in Cat(\mathcal{B})$, $|C| \leq |h(C)|$

Proof 1 *The proof is trivial and relies on an induction on the length of C:*

- *if $|C| = 1$ then $C \in \mathcal{B}$ and we always have $1 \leq |h(C)|$: the property is true;*
- *if the property is supposed to be true for every category of length at most n, then let $C \in Cat(\mathcal{B})$ with $|C| = n + 1$. Necessarily, $C = /(A,B)$ or $C = \backslash(A,B)$ with $|A| \leq n$ and $|B| \leq n$. We thus have $|C| = |A| + |B| \leq |h(A)| + |h(B)| = |h(C)|$.*

To deal with questions of learnability, Kanazawa (9) introduced the notion of grammar system, allowing a reformulation of the classical Gold's model of *identification in the limit from positive examples* (8). We recall this notion here and known learnability results concerning categorial grammars.

Definition 7 (Grammar System) *A **grammar system** is a triple $\langle \Omega, \Lambda, L \rangle$:*

- *Ω is the hypothesis space (here, Ω will be a set of grammars),*
- *The sample space Λ is a recursive subset of A^*, for some fixed alphabet A (elements of Λ are sentences and subsets of Λ are languages);*
- *$L : \Omega \rightarrow pow(\Lambda)$ is a naming function. The question of whether $s \in L(G)$ holds between $s \in \Lambda$ and $G \in \Omega$, is supposed to be computable.*

The main grammar systems we will deal with in the following of this paper are $\langle \mathcal{LG}, \Sigma^*, L \rangle$ and $\langle \mathcal{LG}, (\Sigma \times Types(\Theta))^*, TL_h \rangle$.

Definition 8 (Learnability Criterion) *Let $\langle \Omega, \Lambda, L \rangle$ be a grammar system and $\phi : \bigcup_{k \geq 1} \Lambda^k \rightarrow \Omega$ be a computable function. We say that ϕ **converges** to $G \in \Omega$ on a sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ of elements of Λ if $G_i = \phi(\langle s_0, ..., s_i \rangle)$ is defined and equal to G for all but finitely many $i \in \mathbb{N}$ - or equivalently if there exists $n_0 \in \mathbb{N}$ such that for all $i \geq n_0$, G_i is defined and equal to G. Such a function ϕ is said to **learn** $\mathcal{G} \subseteq \Omega$ if for every language L in $L(\mathcal{G}) = \{L(G) | G \in \mathcal{G}\}$ and for every infinite sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ that enumerates the elements of L (i.e. so that $\{s_i | i \in \mathbb{N}\} = L$), there exists some G in $\mathcal{G}$ such that $L(G) = L$ and ϕ converges to G on $\langle s_i \rangle_{i \in \mathbb{N}}$.*

A useful sufficient condition of learnability is known as finite elasticity.

Definition 9 (Infinite and Finite Elasticity) *A class of languages $\mathcal{L}$ has **infinite elasticity** iff there exists an infinite sequence of sentences $\langle s_n \rangle_{n \in \mathbb{N}}$ and an infinite sequence of languages of $\mathcal{L}$: $\langle L_n \rangle_{n \in \mathbb{N}}$ such that for all $n \in \mathbb{N}$, $s_n \notin L_n$ and $\{s_0, \ldots, s_n\} \subseteq L_{n+1}$. A class of languages has **finite elasticity** iff it has not infinite elasticity.*

The finite elasticity of a class of languages implies the learnability of the corresponding class of grammars. Kanazawa (9), proved that for every $k \geq 1$, the class $\mathcal{G}_k$ is learnable in the grammar system $\langle \mathcal{G}, \Sigma^*, L \rangle$. As a consequence subclasses of CGs are also learnable in the grammar system $\langle \mathcal{G}, (\Sigma \times Types(\Theta))^*, TL_h \rangle$ (6). For LG, we also know that the class of rigid LGs is learnable in the grammar system $\langle \mathcal{LG}, \Sigma^F, FL \rangle$ (i.e. from Structural Examples where the structure is provided by a special normal form of proofs (2)) but is not learnable in the grammar system $\langle \mathcal{LG}, \Sigma^*, L \rangle$ (7). We will now concentrate on the learnability of LGs in the grammar system $\langle \mathcal{LG}, (\Sigma \times Types(\Theta))^*, TL_h \rangle$.

3 Learning Lambek Grammars from Typed Examples

In this section, we first prove a learnability result for subclasses of $\mathcal{LG}$ from Typed Examples. We then describe an algorithm which provides the set of LGs compatible with a set of Typed Examples.

3.1 Learnability Theorem

Definition 10 (h-typed Lambek Grammars) *For every vocabulary Σ, every set of basic categories $\mathcal{B}$, every set of basic types Θ and every typing function h from $Cat(\mathcal{B})$ to $Types(\Theta)$, the* class of h-typed LGs $\mathcal{LG}_{h-type}$ *is the set of LGs G over $\Sigma \times Cat(\mathcal{B})$ satisfying the condition:*
$\forall \langle a, A\rangle \in G$ *and* $\langle a, A'\rangle \in G$, $h(A) = h(A') \Rightarrow A = A'$.

These classes are similar to the ones we have studied in the context of CGs, called $\mathcal{G}_{h-type}$. In this context, we have proved interesting language theoretical results concerning the class $\mathcal{G}_{h-type}$ for the special case where h defines a one to one correspondence between $\mathcal{B}$ and Θ. As a matter of fact, for every CG, there exists a member of this class $\mathcal{G}_{h-type}$ recognizing the same structure language (6). We do not have any similar result for LGs and any class $\mathcal{LG}_{h-type}$ but note that for any h, the class $\mathcal{LG}_{h-type}$ is larger than $\mathcal{LG}_1$. For example, the LG of Example 1 is 2-valued but still in $\mathcal{LG}_{h-type}$ because the distinct categories assigned to the word "sings" are associated with distinct types. Nevertheless the restriction expressed in Definition 10 prohibits to assign both $/(CN, \backslash(/(T, S), S))$ and $/(CN, /(\backslash(T, S), S))$ to determinants (the first one for determinants introducing direct objects, the second one for determinants introducing subjects) because both lead to the same type $((e, t), ((e, t), t))$. We will evoke at the end of the paper how to extend our results (and our algorithm) to treat such cases.

Theorem 1 *The class $\mathcal{LG}_{h-type}$ is learnable from h-Typed Examples, i.e. in the grammar system $\langle \mathcal{LG}, (\Sigma \times Types(\Theta))^*, TL_h\rangle$.*

Lemma 2 *For every vocabulary Σ, every sets $\mathcal{B}$ and Θ, every typing function h from $Cat(\mathcal{B})$ to $Types(\Theta)$, every LG $G \in \mathcal{LG}_{h-type}$ without useless category (i.e. without any category never appearing as a node in any syntactic parse), every infinite sequence $\langle s_i\rangle_{i\in\mathbb{N}}$ that enumerates the elements of $TL_h(G)$, $\exists N \in \mathbb{N}$ such that from $\{s_i | 0 \leq i \leq N\}$ it is possible to compute:*

- *the least integer k such that G is k-valued;*
- *a bound on the maximal length of categories assigned to elements of G;*
- *a bound on the maximal number of distinct basic categories used to define the categories assigned to elements of G.*

Proof 2 (proof of the lemma) *Let us compute each of these values:*

- *To compute the least integer k such that G is k-valued, it is enough to note that, as G has no useless category, the condition for G to belong to $\mathcal{LG}_{h-type}$ is precisely that there are exactly the same number of distinct couples $\langle word, category\rangle$ in G than there are corresponding distinct couples $\langle word, type = h(category)\rangle$ in elements of $TL_h(G)$. After all such couples have been presented at least once in elements of $\langle s_i\rangle_{i\in\mathbb{N}}$, k is equal to the maximal number of distinct types associated with the same word.*

- *To compute a bound on the maximal length of categories assigned to elements of G it is, similarly, enough to take the maximal length of types appearing in elements of $\langle s_i\rangle_{i\in\mathbb{N}}$. Lemma 1 ensures that the bound on the lengths of types is also a bound on the lengths of categories. Let us call L such a bound.*

- *Finally, to compute a bound on the maximal number of distinct basic categories used to define the categories assigned to elements of G, it is enough to take advantage of both previous results. This number is bounded by $k\times L\times|\Sigma|$ where $|\Sigma|$ stands for the number of words in the vocabulary of G, available as soon as each word has been presented at least once.*

Proof 3 (proof of the theorem) *The theorem is a direct consequence of lemma 2. As a matter of fact, the lemma implies that there is a finite computable number of LGs without useless categories (up to a renaming of basic categories) compatible with any sequence enumerating the elements of some $TL_h(G)$ and that this set is recursively enumerable. This situation implies finite elasticity and thus learnability of the class $\mathcal{LG}_{h-type}$ in the grammar system $\langle\mathcal{LG}, (\Sigma\times Types(\Theta))^*, TL_h\rangle$.*

The proof suggests an enumerative learning algorithm. This is not, of course, a tractable strategy. We define in the following another algorithm.

3.2 Learning Strategy

The strategy proposed to infer LGs from Typed Examples takes as input a set of h-Typed Examples and provides as output the set of LGs compatible with the input. The key point of the entire strategy is to observe that types provide indications on the functor or argument nature of the elements of vocabulary with which they are associated, but where the directions of the operators / and \ are lost. So, types are "poor" versions of categories with which the typing function link them and the goal of the learning strategy is to rebuild the categories from the available types. This will be done in three steps described below: *variabilisation*, *constraint inference* and *categories deduction*.

3.2.1 Variabilisation

First, a step of *variabilisation* is necessary to introduce variables in type expressions at the operator positions, i.e. before every opening parenthesis of every type. This step is applied once for all Typed Examples in the input. The introduction of variables respects the following constraint: *all occurrences of the same word with the same associated type receive the same variables.* This constraint is a direct consequence of the fact that the target grammar of the algorithm belongs to $\mathcal{LG}_{h-type}$: each time the same couple $\langle word, type \rangle$ is present in a Typed Example, we know that it refers to a unique couple $\langle word, category \rangle$ in this target grammar.

Definition 11 (types with variables) *Let $\mathcal{X}$ be a infinite countable set of variables. The set of types with variables over the set of basic types*[2] *Θ is denoted $VarType(\Theta)$ and is defined as the smallest set such that: $\Theta \subset VarType(\Theta)$ and for any $u, v \in VarType(\Theta)$ and $x_i \in \mathcal{X} \cup \{/, \backslash\}$, $x_i(u, v) \in VarType(\Theta)$.*

The variables take their value in $\mathcal{X} \cup \{/, \backslash\}$. Note that, because of the constraint on the introduction of identical variables, this variabilisation step can only be defined relatively to a *set* of Typed Examples.

Example 5 *We illustrate the variabilisation step for the h-Typed Example given in Example 4:*

John	*sings*	*a*	*song*
e	$x_0(e, x_1(e, t))$	$x_2(x_3(e, t), x_4(x_5(e, t), t))$	$x_6(e, t)$

3.2.2 Constraint Inference

This step consists in deducing constraints over the variables introduced. These constraints will be stored in substitutions.

Definition 12 *A **substitution** is a mapping from $\mathcal{X}$ to $\mathcal{X} \cup \{/, \backslash\}$. For any substitution σ, σ is extended over the set $VarType(\Theta)$ as follows: (1) $\sigma(u) = u$, $\forall u \in Types(\Theta) \cup \{/, \backslash\}$; (2) $\sigma(x_i(u, v)) = \sigma(x_i)(\sigma(u), \sigma(v))$.*

For any substitution $\sigma : VarType(\Theta) \rightarrow VarType(\Theta)$, the sequences of the Typed Examples are proved one by one to deduce the special type $t \in \Theta$ using an adapted σ-dependant Lambek-inspired sequent calculus defined by:

- axioms : $[ID]^{\sigma} \ \frac{}{A \Vdash_{\sigma} A}$ iff $\sigma(A) = \sigma(A')$, for any $A, A' \in VarType(\Theta)$;

[2] For the examples in this presentation of the learning strategy we consider the set of basic types $\Theta = \{e, t\}$

- inference rules :

$$[/R]\ \frac{\Gamma, A \Vdash_\sigma B}{\Gamma \Vdash_\sigma x(A,B)},\ \sigma(x) = / \qquad [\backslash R]\ \frac{A, \Gamma \Vdash_\sigma B}{\Gamma \Vdash_\sigma x(A,B)},\ \sigma(x) = \backslash$$

$$[/L]\ \frac{\Gamma \Vdash_\sigma A\ \Delta, B, \Pi \Vdash_\sigma C}{\Delta, x(A,B), \Gamma, \Pi \Vdash_\sigma C},\ \sigma(x) = / \quad [\backslash L]\ \frac{\Gamma \Vdash_\sigma A\ \Delta, B, \Pi \Vdash_\sigma C}{\Delta, \Gamma, x(A,B), \Pi \Vdash_\sigma C},\ \sigma(x) = \backslash$$

where $A, B, C \in VarType(\Theta)$ and Γ, Δ, Π are concatenations of types with variables from $VarType(\Theta)$, $\Gamma \neq \emptyset$. Making a proof in this system implies defining constraints on the values of a substitution on a subset of $\mathcal{X}$.

For a given Typed Example, the purpose is to prove that the sequent having the corresponding variabilised sequence of types with variable as antecedent and the type t as consequent is valid in the σ-sequent calculus. As the sequence of types with variables comes from a Typed Example (i.e. a sequence of categories provable in the Lambek calculus), it is easy to see that there exists at least a proof in the σ-sequent calculus. Each proof provides a set of constraints on σ. Solutions of this set of constraints are the resulting substitutions.

Example 6 *Applied to the variabilised h-Typed Example of Example 5, the search for a proof in the σ-sequent calculus gives rise to the search tree of Figure 1, where only the branches leading to valid proofs are displayed. In ovals are given the constraints applying on substitutions and in rectangles the sequents obtained after applying the rules (not themselves represented in the Figure). 7 substitutions are obtained, among which only 5 are distinct (we have: $\sigma_2 = \sigma_3 = \sigma_4$). They are summed-up in Table 1.*

Table 1
The substitutions inferred for the h-Typed Example of Example 5

Variables	σ_1	σ_2	σ_5	σ_6	σ_7
x_0	$\backslash$	$\backslash$	$/$	$\backslash$	$\backslash$
x_1	$\sigma_1(x_3)$	$\sigma_2(x_5)$	$\backslash$	$/$	$\backslash$
x_2	$\backslash$	$/$	$\backslash$	$\backslash$	$\backslash$
x_3	$\sigma_1(x_1)$	$\sigma_2(x_6)$	$/$	$/$	$\backslash$
x_4	$/$	$\backslash$	$/$	$/$	$/$
x_5	$\sigma_1(x_6)$	$\sigma_2(x_1)$	$\sigma_5(x_6)$	$\sigma_6(x_6)$	$\sigma_7(x_6)$
x_6	$\sigma_1(x_5)$	$\sigma_2(x_3)$	$\sigma_5(x_5)$	$\sigma_6(x_5)$	$\sigma_7(x_5)$

3.2.3 Category Deduction

Each distinct substitution will give rise to a distinct LG, associated with its specific typing function. To obtain grammars and typing functions from substitutions, a last step of category deduction is necessary. As a non-basic type can be associated by the typing function with a basic category (for example the type (e,t) and CN) a simple renaming is not enough. The deduction of categories is performed as follows:

(1) every smallest subtype with variables that is used as argument in the proof is associated with a new basic category;
(2) the type t, as a result, is associated with the axiomatic category S.

The definition of the typing functions naturally follows from these definitions.

Example 7 *For the same previous example, the substitution σ_1 applied to the variabilised type assignments of Example 5 gives: $\langle John, e\rangle$, $\langle sings, \backslash(e, x_1(e,t))\rangle$, $\langle a, \backslash(x_1(e,t), /(x_5(e,t), t))\rangle$ and $\langle song, x_5(e,t)\rangle$. The first step of the deduction assigns a basic category A_1 to e (with $h(A_1) = e$), and two new basic categories A_2 and A_3 to the chunks $x_1(e,t)$ and $x_5(e,t)$ respectively (with $h(A_2) = (e,t) = h(A_3)$). After the second step of deduction, the final LG obtained is: $G = \{\langle John, A_1\rangle, \langle sings, \backslash(A_1, A_2)\rangle, \langle a, \backslash(A_2, /(A_3, S))\rangle, \langle song, A_3\rangle\}$.*

In Gold's model, the learning function takes as input a *set of examples*, not a single one. The inference strategy applied to a set of h-Typed Examples is given in Algorithm 1. At each step are kept only the substitutions that are compatibles with the new example being analysed. The compatibility relation is a composition denoted by $\ominus$. For any substitutions σ_1 and σ_2 we define:

$$\forall u \in Types(\Theta) \cup \{/, \backslash\}, (\sigma_1 \ominus \sigma_2)(u) = u,$$

$$\forall x_i \in \mathcal{X}, (\sigma_1 \ominus \sigma_2)(x_i) = \begin{cases} \sigma_1(x_i) \text{ if } \sigma_1(x_i) = \sigma_2(x_i) \\ \sigma_1(x_i) \text{ if } \sigma_2(x_i) = x_i \\ \sigma_2(x_i) \text{ if } \sigma_1(x_i) = x_i \\ \text{not defined on } \mathcal{X} \text{ if } \sigma_1(x_i) \neq \sigma_2(x_i) \end{cases}$$

Algorithm 1 Inference strategy for a set of Typed Examples

Require: $TE = \{te_1, \ldots, te_n\}$ a set of Typed Examples;
1: $\mathcal{S} = \{Id_{VarTypes(\Theta)}\}$;
2: **for** every Typed Example $te_i \in TE$ **do**
3: introduce variables respecting the condition of 3.2.1 to obtain tv_i;
4: **end for**
5: **for** every sequence of types with variables tv_i **do**
6: prove the sequent $tv_i \Vdash_\sigma t$ in the σ-sequent calculus to obtain a set of substitutions $\mathcal{S}_i$;
7: $\mathcal{S} = \{s \ominus s_i | s \in \mathcal{S}, s_i \in \mathcal{S}_i\}$;
8: **end for**
9: **for** every substitution s from $\mathcal{S}$ **do**
10: apply s over types with variables;
11: apply rules (1) and (2) of deducing categories to obtain the function h
12: **end for**
Ensure: $\mathcal{G}_r(TE) = \{\langle G_i, h_i\rangle | 1 \leq i \leq |\mathcal{S}|\}$

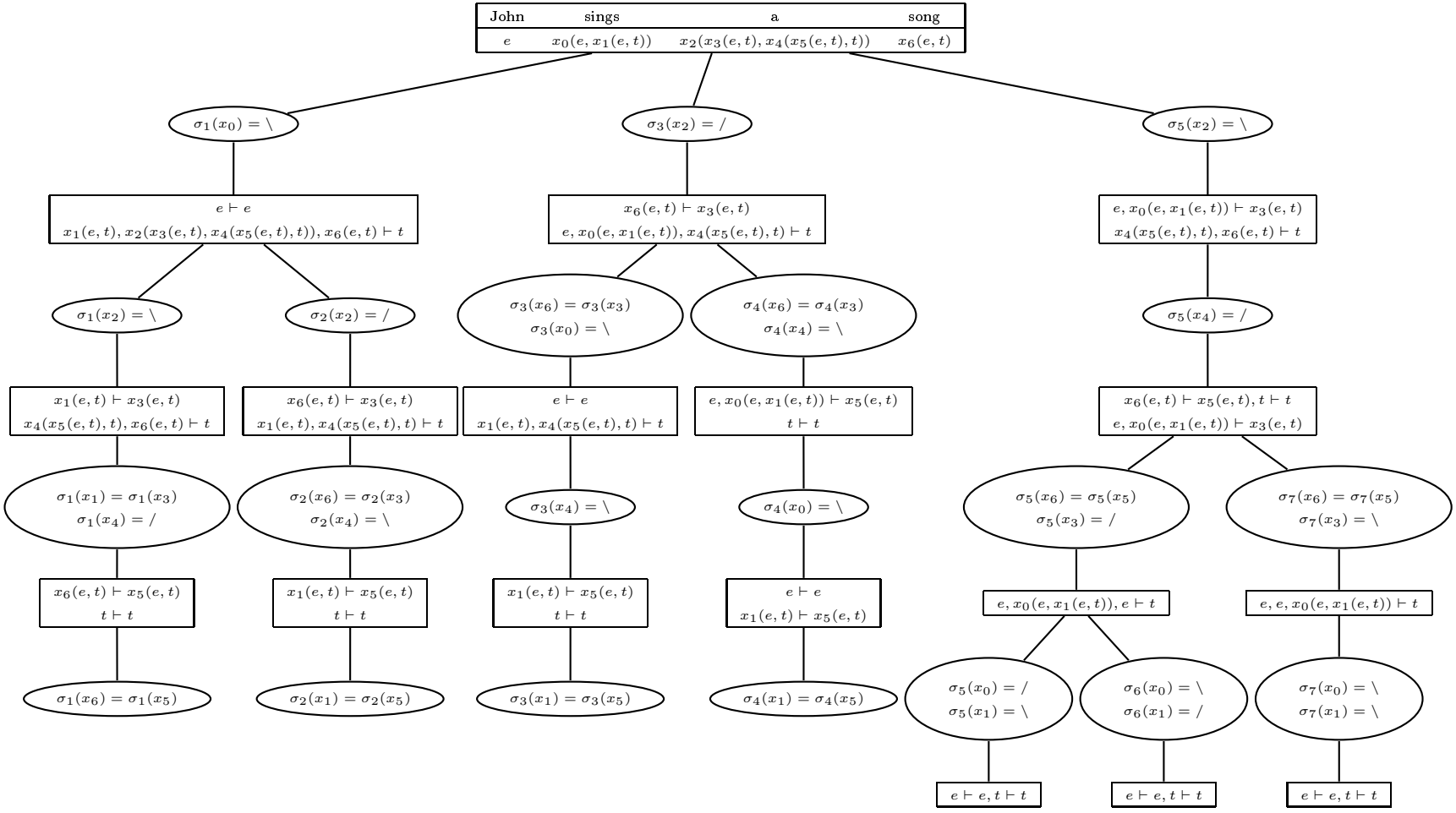

Fig. 1. The substitution searching tree

3.2.4 *Exemple of the Global Strategy*

John	sings	a	song
e	$x_0(e, x_1(e,t))$	$x_2(x_3(e,t), x_4(x_5(e,t),t))$	$x_6(e,t)$
Mary	follows	a	man
e	$x_7(e, x_8(e,t))$	$x_2(x_3(e,t), x_4(x_5(e,t),t))$	$x_9(e,t)$

For this set of h-Typed Examples, the variabilisation has been performed following the constraint explained in section 3.2.1. So, both occurrences of the determinant "a" receive the same variables. After treating the first h-Typed Example, the resulting substitutions are those given in Table 1. The second Example "Mary follows a man" is treated exactly the same way as the first one and also gives rise to 5 different substitutions (replace x_0 by x_7, x_1 by x_8 and x_6 by x_9 in Table 1 to obtain their values). Among the 25 possible compositions between one σ_i and one of these new substitutions, only 11 are defined, and 7 distinct. The 7 distinct LGs of Table 2 are finally obtained:

Table 2
The final Lambek Grammars inferred

	John	sings	a	song	Mary	follows	man
G_1	A_1	$\backslash(A_1, \backslash(A_1, S))$	$\backslash(\backslash(A_1, S), /(A_2, S))$	A_2	A_1	$\backslash(A_1, \backslash(A_1, S))$	A_2
G_2	A_1	$\backslash(A_1, A_2)$	$\backslash(A_2, /(A_3, S))$	A_3	A_1	$\backslash(A_1, A_2)$	A_3
G_3	A_1	$\backslash(A_1, A_2)$	$/(A_3, \backslash(A_2, S))$	A_3	A_1	$\backslash(A_1, A_2)$	A_3
G_4	A_1	$\backslash(A_1, /(A_1, S))$	$\backslash(/(A_1, S), /(A_2, S))$	A_2	A_1	$\backslash(A_1, /(A_1, S))$	A_2
G_5	A_1	$\backslash(A_1, /(A_1, S))$	$\backslash(/(A_1, S), /(A_2, S))$	A_2	A_1	$/(A_1, \backslash(A_1, S))$	A_2
G_6	A_1	$/(A_1, \backslash(A_1, S))$	$\backslash(/(A_1, S), /(A_2, S))$	A_2	A_1	$/(A_1, \backslash(A_1, S))$	A_2
G_7	A_1	$/(A_1, \backslash(A_1, S))$	$\backslash(/(A_1, S), /(A_2, S))$	A_2	A_1	$\backslash(A_1, /(A_1, S))$	A_2

First, note that every solution grammar assigns the same basic category to "John" and "Mary". This is a direct consequence of their assignment to the same basic type (Montague assigns them a more complicated type) and of condition (2) of Definition 4. More interestingly, every solution grammar also assigns the same basic category to "song" and "man". This results from an equality condition between the values of every substitution on x_6 and x_9, both equal to their value on x_3 or x_5. Fundamentally, it is because both words are introduced by the same determinant "a", whose type is variabilised only once. Grammatical categories can thus be defined as equivalence classes between types that have the same combinatorial behaviour. The substitutability at the type level is the criterion our algorithm uses to infer grammatical categories.

It can also be noted that the 7 solution grammars obtained can be reduced to 3 classes. G_1 is clearly apart, allowing a non-linguistically valid combination between the determinant and the verb. G_2 and G_3 are such that $\forall w \in \Sigma$, $\exists C_2, C_3 \in Cat(\mathcal{B})$ with: $\langle w, C_2\rangle \in G_2$, $\langle w, C_3\rangle \in G_3$, $C_2 \vdash C_3$ and $C_3 \vdash C_2$. It is easy to prove (using the Cut rule) that this condition implies that $\forall h$, $TL_h(G_2) = TL_h(G_3)$. So, G_2 and G_3 are equivalent with respect to the

convergence criterion and it is enough to keep memory of only one of them. The same situation occurs for the four grammars G_4, G_5, G_6 and G_7. Unfortunately, the previous condition is only a sufficient one for Typed Languages to be equal.

3.3 Properties of the Algorithm and Extensions

Theorem 2 *For every sets Σ, $\mathcal{B}$ and Θ, every typing function h from $Cat(\mathcal{B})$ to $Types(\Theta)$, every $G \in \mathcal{LG}_{h-type}$ and every set TE of h-Typed Examples of G, the result $\mathcal{G}_r(TE)$ of our algorithm contains every possible couple $\langle G_i, h_i \rangle$ such that G_i is without useless category, $G_i \in \mathcal{LG}_{h_i-type}$ and $TE \subset TL_{h_i}(G_i)$.*

The proof of this correctness and completeness theorem (not given here) is a direct adaptation of the one for CGs (4). But it is not enough to make our algorithm a learning algorithm in the sense of Gold, as it does not allow to select a unique solution grammar. It is possible that $\exists \langle G_i, h_i \rangle$, $\langle G_j, h_j \rangle \in \mathcal{G}_r(TE)$ such that $TL_{h_i}(G_i) \subsetneq TL_{h_j}(G_j)$ (4). In this case, to avoid over-generalisation, G_i should be chosen. But it is not known whether the inclusion of Typed Language is computable. The inclusion can nevertheless be checked for every Typed Example of bounded length (that's Kanazawa's strategy for learning CGs from strings) but, of course, makes the strategy intractable.

The complexity of our strategy is already exponential in the number of words in the input. To reduce it in practice, a valid heuristics based on a *Count* function can be proposed (14). Applied to (sequence of) categories, and similarly applicable to (sequences of) types, *Count* computes the exponent of a basic category on both sides of a sequent. *Count* is defined for elements of $VarType(\Theta)$ as follows: (1) $Count_\tau(\tau) = 1$, $\forall \tau \in \Theta$, (2) $Count_\tau(\tau_1) = 0$ if $\tau \neq \tau_1$, $\forall \tau, \tau_1 \in \Theta$, (3) $Count_\tau(x_i(\alpha_1, \alpha_2)) = Count_\tau(\alpha_2) - Count_\tau(\alpha_1)$. It is naturally extended to sequences: $Count_\tau(\tau_1.\tau_2) = Count_\tau(\tau_1) + Count_\tau(\tau_2)$. It is easy to prove that for any sequences of types with variables Γ and Δ and every substitution σ: if $\Gamma \Vdash_\sigma \Delta$ then $\forall \tau \in \Theta$, $Count_\tau(\Gamma) = Count_\tau(\Delta)$. This equality can be checked in linear time and allows to prune unfruitful branches.

Finally, the learnability results exposed in this paper rely on a special kind of rigidity: a unique correspondence between couples $\langle word, category \rangle$ and couples $\langle word, type \rangle$. Like every other result based on rigidity, it can be extended to k-valued variants. That is, if it is known that at most k distinct types can be associated with a category, learnability results are preserved. Furthermore, the same algorithm can be adapted to treat this case: only the composition between substitutions needs to be modified to allow at most k distinct values for every set of variables appearing in a unique type. This variant allows to treat the case of determinants, which need the assignment of two distinct categories corresponding with a unique type, but it introduces a higher complexity.

4 Conclusion

Before we started our study, positive learnability results in Gold's model for LGs were still rare and only concerned rigid LGs (2; 1). We present here a result of learnability for larger classes of LGs, provided that adapted data are available. The advantages of our strategy are very similar to the ones we already argued for CGs: types are lexical information, easier to justify than structural information. Despite bad theoretical complexity, the algorithm has been implemented and applied on small sets of natural language sentences.

References

[1] D. Béchet and A. Foret. Apprentissage des grammaires de Lambek rigides et d'arité bornée pour le traitement automatique des langues. in *CAP 2003*, p. 155–168, 2003.

[2] R. Bonato and C. Rétoré. Learning rigid Lambek grammars and minimalist grammars from structured sentences. in *Proc. of Learning Language in Logic (LLL)*, p. 23–34, 2001.

[3] W. Buszkowki and G. Penn. Categorial grammars determined from linguistic data by unification, *Studia Logica*, p. 431–454, 1990.

[4] D. Dudau-Sofronie, Apprentissage de grammaires catégorielles pour simuler l'acquisition du langage naturel PhD thesis, université de Lille, 2004.

[5] D. Dudau-Sofronie, I. Tellier, and M. Tommasi. From logic to grammars via types. in *Proc. of Learning Language in Logic (LLL)*, p. 35–46, 2001.

[6] D. Dudau, I. Tellier, and M. Tommasi. A learnable class of ccg from typed examples. in *Proc. of the 8th conference on Formal Grammar*, p. 77–88,2003.

[7] A. Foret and Y. le Nir. On limit points for some variants of rigid lambek grammars. in M. Van Zaanen P. Adriaans, H. Fernau (eds) *Proc. ICGI 2002*, LNAI vol. 2484, p. 106–119, 2002.

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

[9] M. Kanazawa. *Learnable Classes of Categorial Grammars*, CSLI Publ. 1998.

[10] J. Lambek. The mathematics of sentence structure. (65):154–170, 1958.

[11] R. Montague. *Formal Philosophy; Selected papers of Richard Montague.* 1974.

[12] B. Partee. *Mathematical methods in Linguistics.* Number 30. 1990.

[13] Mati Pentus. Lambek calculus is np-complete. Draft, 2003.

[14] J. van Benthem. The Lambek Calculus in *Categorial Grammars and Natural Language Structures* Reidel, Dordrecht, p. 35–68, 1988.