Une classe de grammaires catégorielles apprenable à partir d'Exemples Typés

Daniela Dudau-Sofronie, Isabelle Tellier, Marc Tommasi

Grappa et Université Charles de Gaulle-Lille3
59 653 Villeneuve d'Ascq Cedex
dudau@grappa.univ-lille3.fr
tellier, tommasi@univ-lille3.fr
http://www.grappa.univ-lille3.fr

Abstract

La classe des grammaires catégorielles dites AB ou classiques a donné lieu ces dernières années à des résultats d'apprenabilité au sens de Gold (principalement dus à Kanazawa) intéressants. Cette classe mérite d'être étudiée parce que ses membres permettent de générer l'ensemble des langages algébriques et parce que l'interface qu'elle permet avec une interprétation sémantique la rend apte à modéliser certaines particularités des langues naturelles. Mais les résultats d'apprenabilité connus ne concernent que les classe des grammaires k-valuées avec $k \geq 1$. Nous définissons dans cet article une nouvelle sous-classe de grammaires catégorielles classiques à la fois intéressante du point de vue de la théorie des langages (puisque ses représentants permettent de générer l'ensembles des langages de structures de toutes les grammaires categorielles classiques) et du point de vue de l'apprentissage (puisqu'elle est apprenable au sens de Gold à condition de fournir des données adaptées).

Mots clef inférence grammaticale, grammaires catégorielles, apprentissage à la limite, exemples typés

1 Introduction

Categorial Grammars are well known lexicalized formalisms, often used to model natural languages. Their main interest is their expressivity and the fact that they allow good connections with formal semantics in Montague's tradition. Furthermore, this grammatical formalism seems well adapted to a learning process because the rules are expressed by a limited set of schemes and it remains only to infer the syntactic categories associated with lexical items. That's way, nowadays the interest in studying various models and variants of such grammars is growing [Adr92, Kan98] and [BR01, FlN02] [1].

Classical Categorial Grammars (CCG in the following) is one well known instance of the family of Categorial Grammars. Although too rudimentary

[1] This research was done during the cooperative research action of INRIA, ArcGracq 2001-2002

to model subtle linguistic phenomena, this family is interesting to study because some formal learnability results (in Gold's model) have recently been proved for large subclasses of CCGs [Kan98]. But these results do not provide tractable learning algorithms, as most of the problems to be solved are NP-hard [Flo01, Flo02]. The only favorable case is when rigid CCGs are to be learned from Structural Examples because this can be performed in linear time. Besombes and Marion [BM02] have shown that the class of grammars generating the same languages as reversible tree automata, strictly larger then the class of rigid CCGs is also efficiently learnable from Structural Examples. But the class of rigid grammars has poor expressive power with respect to natural language. Moreover, learning algorithms must have access to Structural Examples which corresponds to derivation trees of sentences and it seems to us that this requirement is not natural.

We define a new subclass of CCGs with good properties from a language-theoretic point of view. Our main result is that for every CCG, another CCG producing the same structure language as (i.e. strongly equivalent with) the first one and belonging to this new subclass can be built. This new subclass is then proved learnable from Typed Examples. Typed Examples are sentences enriched with lexicalized information which can be interpreted as coming from semantics, and can be thus considered more "naturally available" than Structural Examples.

We argue the availability of the Typed Examples firstly from a theoretical point of view, exploiting the compositionality and secondly by the efforts we do to constitute a natural language resource (corpus) from texts enriched with semantic information. As a matter of fact, we have already noticed that one of the main interests of Categorial Grammars is their connection with semantics. This connection relies on a formal statement of the well known Principle of Compositionality [Jan97]. Its contemporary version states that : *"the meaning of a compound expression is a function of the meaning of its parts and of the syntactic rules by which they are combined"* [Par90]. If the *"parts"* mentioned in this definition are assimilated with words and the *"compound expressions"* with phrases, this formulation implies that words have individual meanings and the semantics of a phrase (and thus of a sentence) only depends of the meaning of its words and of its syntactic structure. We believe that this Principle is still under-exploited in formal models of grammatical inference. Links between Structural Examples and semantic information have been shown in [Tel99]. This first approach was still not satisfactory as it did not avoid combinatorial explosion. This paper presents a new way of considering learning Categorial Grammars from semantic knowledge. But, here, we will not need to suppose that word meanings are fully known before the grammar learning process can start. Instead, we make the smoother hypothesis that the crucial information to be extracted from the environment is the *semantic type* of words. Semantic types, in the usual sense, are general information making a distinction between facts, entities and properties satisfied by entities. Most knowledge representation formalisms use this notion, so types can be supposed to be directly extracted from the environment. Types can also be considered as *lexicalized structural information*.

For practical reasons, to achieve experiments, we need corpora of typed texts. But such corpora are not available and they have to be built. For this construction, tree banks (necessary to obtain Structural Examples) are of no interest. On the contrary, because semantic types are lexicalized, simpler resources like

lexical taggers are of great help. A tagger is able to recognize proper nouns, common nouns and other lexical items whose lexical tag is easily transformable in a lexical type. For verbs, a tagger does not necessarily make the distinction between transitive and intransitive ones, and a post-treatment needs to be done. We are working to produce a clean version of a typed corpus in French of almost 100000 words that will be used for experiments.

This paper is organized in five sections. The second section introduces the preliminary notions : CCGs, semantic types and the definition of the new introduced sub-class of CCGs . The third section presents the main result of strong equivalence and the fourth section is about the learnability from typed examples. We conclude with some comparative remarks.

2 A New Subclass of Classical Categorial Grammars

2.1 Classical Categorial Grammars

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 $/$, $\backslash$ and the set $\mathcal{B}$: $Cat_{\mathcal{B}}$ is the smallest set such that $\mathcal{B} \subset Cat_{\mathcal{B}}$ and for any $A \in Cat_{\mathcal{B}}$ and $B \in Cat_{\mathcal{B}}$ we have: $/(A, B) \in Cat_{\mathcal{B}}$ and $\backslash(A, B) \in Cat_{\mathcal{B}}$.

Let Σ be a fixed alphabet called vocabulary. A **categorial grammar over** Σ is any finite relation between Σ and $Cat_{\mathcal{B}}$, i.e. $G \subset \Sigma \times Cat_{\mathcal{B}}$ and G is finite. For a symbol $a \in \Sigma$ and a category $A \in Cat_{\mathcal{B}}$ if $\langle a, A\rangle \in G$, we say that the category A is assigned to a.

In the general framework of categorial grammars, the language $L(G)$ of a grammar G is the set of finite concatenations of elements of the vocabulary for which there exists an assignment of categories that can be *reduced* to the axiom S. For **Classical Categorial Grammars (or CCGs)**, the admitted reduction rules for any categories A and B in $Cat_{\mathcal{B}}$ are[2]:

- forward application $FA : /(A, B).A \to B$;
- backward application $BA : A.\backslash(A, B) \to B$

We denote by $\mathcal{G}$ the set of every CCG and for any integer $k \geq 1$, $\mathcal{G}_k$ is the set of k-valued CCGs, i.e. the set of CCGs assigning at most k different categories to each member of its vocabulary.

As usual in term algebras, a context is a category with exactly one occurrence of a distinguished constant (not in $\mathcal{B}$). We denote $C[]$ a context and $C[A]$ is the category obtained by replacing the distinguished constant by the category A. Forward and backward rules justify that for any category $X = C[/(A, B)]$ (or $X = C[\backslash(A, B)]$) we say that A occurs in X at an argument position and B occurs in X at a result position.

Any mapping Φ defined from $\mathcal{B}$ to $Cat_{\mathcal{B}}$ can be extended to contexts and elements of $Cat_{\mathcal{B}}$ in the following way: for every $A \in Cat_{\mathcal{B}}$ and $B \in Cat_{\mathcal{B}}$,

[2]These rules justify the fractional notations of the category-building operators / and \ usually used in the literature. Categories, often written B/A (resp. $A\backslash B$), can be considered as functors expecting as argument the category A and providing as result the category B.

$\Phi(\backslash(A,B)) = \backslash(\Phi(A),\Phi(B))$ and $\Phi(/(A,B)) = /(\Phi(A),\Phi(B))$. For any CCG G, we can also define $\Phi(G) = \{\langle a, \Phi(A)\rangle \mid \langle a, A\rangle \in G\}$.

We denote by $\mathcal{B}(G)$ the (finite) set of basic categories that occur in G and $Cat_{\mathcal{B}}(G) \subset Cat_{\mathcal{B}}$ is the set of categories that occur in G.

2.2 Canonical Types

For any countable set of basic categories $\mathcal{B}$, we define the set of Canonical Types $Types(\mathcal{B})$ as the smallest set such that $\mathcal{B} \subset Types(\mathcal{B})$ and for any $U \in Types(\mathcal{B})$ and $V \in Types(\mathcal{B})$ we have $(U,V) \in Types(\mathcal{B})$. The type (U,V) is to be read as a functor taking as argument the type U and providing as result the type V. Canonical Types can thus be seen as non oriented categories, *i.e.* categories where both operators / and \ are erased. We call the Canonical Typing Function h the unique function from $Cat_{\mathcal{B}}$ to $Types(\mathcal{B})$ recursively defined by: (1) $h_{|\mathcal{B}} = Id_{\mathcal{B}}$ where $Id_{\mathcal{B}}$ denotes the identity function on the set $\mathcal{B}$; (2) for any $A \in Cat_{\mathcal{B}}$ and $B \in Cat_{\mathcal{B}}$ we have: $h(/(A,B)) = h(\backslash(A,B)) = (h(A), h(B))$. The Canonical Typing Function simply transforms categories into the corresponding Canonical Types by deleting the operators. h can also be naturally extended to contexts of categories.

2.3 The Class $\mathcal{G}_{Type}$

Definition 1 *For any vocabulary Σ and any set of basic categories $\mathcal{B}$, we note $\mathcal{G}_{Type}$ the set of CCGs G on Σ satisfying the property:*

$$\forall \langle a, A\rangle \in G,\ \langle a, A'\rangle \in G \quad h(A) = h(A') \Rightarrow A = A'. \tag{1}$$

In other words, in the CCGs of $\mathcal{G}_{Type}$, the types corresponding with the categories of a single member of the vocabulary are all distinct. Different categories assigned to the same symbol of the vocabulary can be distinguished looking at their type. We can compare the class $\mathcal{G}_{Type}$ with the classes $\mathcal{G}_k$ of k-valued CCGs. Interestingly, the two notions do not coincide.

Example 1 *Let $\Sigma = \{a, b\}$ and let us consider a CCG grammar G_0 recognizing the language a^*ba^*. G_0 is defined by the assignments $\{\langle b, S\rangle, \langle a, /(S,S)\rangle, \langle a, \backslash(S,S)\rangle\}$. It is worth noting that G_0 belongs to the class $\mathcal{G}_2$ of 2-valued CCGs and therefore belongs to any $\mathcal{G}_k$, $k \geq 2$ but is not in $\mathcal{G}_{Type}$.*

Let $(X_i)_{i\geq 0}$ be a sequence of categories defined by $X_0 = A$, $X_1 = /(A,S)$ and $X_i = /(A, X_{i-1})$ for $i > 1$. Let us consider the grammar G_k defined by: $G_k = \{\langle a, X_i\rangle | 0 \leq i \leq k\}$. For $k \geq 1$, the language recognized by G_k is $L(G_k) = \{a^i \mid 2 \leq i \leq k+1\}$. For every $k \geq 1$, G_k is in $\mathcal{G}_{Type}$ and G_k is $(k+1)$-valued but not k-valued.

Proposition 1 *$\mathcal{G}_1 \subset \mathcal{G}_{Type}$ and $\forall k > 1$, $\mathcal{G}_{Type} \cap \mathcal{G}_k \neq \emptyset$, $\mathcal{G}_k \not\subseteq \mathcal{G}_{Type}$ and $\mathcal{G}_{Type} \not\subseteq \mathcal{G}_k$.*

Proof 1 *The fact that $\mathcal{G}_1 \subset \mathcal{G}_{Type}$ is obvious and other properties are easily proved using the grammars given in Example 1.*

Therefore, the situation is the one displayed in Figure 1.

The set of languages generated (or recognized) by CCGs is the set of ϵ-free context-free languages, where ϵ is the empty word (Gaifman's theorem in

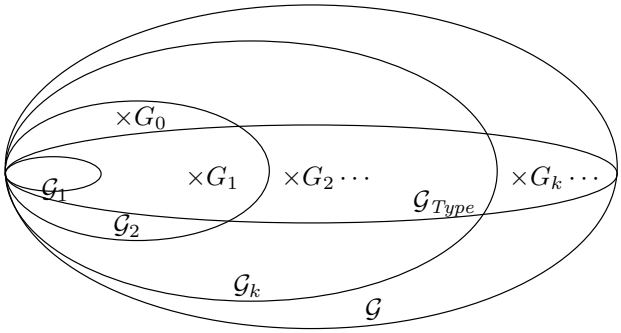

Figure 1: Comparisons between classes of k-valued categorial grammars and the $\mathcal{G}_{Type}$ class.

[BHGS60]). Now that we have defined a subclass of $\mathcal{G}$, it is natural to wonder if its members allow to generate the same set of languages or only a subset of it.

Proposition 2 *For every fixed sets Σ and $\mathcal{B}$ and every ϵ-free context-free language $L \subset \Sigma^*$ there exists a CCG $G \in \mathcal{G}_{Type}$ so that $L(G) = L$.*

Proof 2 *It has long been recognized that Gaifman's proof is equivalent with the existence of the strong Greibach normal form for ϵ-free context-free languages. That is, for every ϵ-free context-free language $L \subset \Sigma^*$ there exists a phrase structure grammar $G' = \langle \Sigma, NT, P, S \rangle$ (where Σ is the set of terminal symbols, NT the set of nonterminal symbols, P the set of production rules and $S \in NT$ the axiom) in strong Greibach normal form satisfying $L(G') = L$. This normal form specifies that every production rule in P is of the form $X \longrightarrow a$ or $X \longrightarrow aX_1$ or $X \longrightarrow aX_1X_2$ with X, X_1 and X_2 elements of NT and a element of Σ. To build a CCG G that is strongly equivalent with G', proceed as follows: (1) define a bijection f between NT and a part of $\mathcal{B}$ satisfying $f(S) = S$; (2) for every rule of the form $X \longrightarrow a$ in P, let $\langle a, f(X) \rangle \in G$; (3) for every rule of the form $X \longrightarrow aX_1$ in P, let $\langle a, /(f(X_1), f(X)) \rangle \in G$; (4) for every rule of the form $X \longrightarrow aX_1X_2$ in P, let $\langle a, /(f(X_1), /(f(X_2), f(X))) \rangle \in G$. We have $L(G) = L(G') = L$. Because f is a bijection and, by construction, types are pairwise distinct in G, it is trivial to check that $G \in \mathcal{G}_{Type}$.*

Though constructive, this proof is not satisfying, as the CCG obtained produces syntactic analysis trees of a very peculiar form, where only the rule *FA* is used. The next section will take into account the structures of the analysis trees produced by CCGs and provide a much more interesting (because structure-preserving) result of the same kind.

3 Structure Languages of $\mathcal{G}_{Type}$

A functor-argument structure over an alphabet Σ is a binary-branching tree whose leaf nodes are labeled by elements of Σ and whose internal nodes are labeled either by *BA* or *FA*. The set of functor-argument structures over Σ is denoted Σ^F. Let G be a CCG. A *Structural Example* for G is an element of Σ^F

obtained from the parse tree of a sentence w in $L(G)$ by deleting categories. The *Structure Language* of G, denoted by $FL(G)$, is the set of Structural Examples of G.

The notion of Structural Example is of crucial importance as it is the basis of every learning result about CCGs [BP90, Kan98]. Furthermore, it allows the connection between syntax and semantics [Tel99]. In the domain of CCGs, two grammars can be said strongly equivalent if they share the same Structure Language.

Example 2 *Let us consider the grammar G_0 of Example 1. The sentence aba has two parses in this grammar and therefore there are two Structural Examples for G and aba: $FA(a, BA(b, a))$ and $BA(FA(a, b), a)$.*

3.1 The Main Result

Theorem 1 *For any vocabulary Σ and any set of basic categories $\mathcal{B}$, for every CCG $G \in \mathcal{G}$, there exists a CCG $G' \in \mathcal{G}_{Type}$ so that $FL(G) = FL(G')$.*

The theorem states that the class $\mathcal{G}_{Type}$ has the same expressive power in a strong sense as the entire class $\mathcal{G}$ of Classical Categorial Grammar. That is to say that the restriction imposed by Eq. (1) has no incidence on the expressive power of the grammars in $\mathcal{G}_{Type}$.

The proof of this theorem is constructive, having as support the two algorithms respectively named: algorithm 1 G to Gtype and algorithm 2 Transform. We construct a finite chain of grammars $G_0, G_1, \ldots G_m \in \mathcal{G}$ such that $G_0 = G$, $G_m = G'$ and $\forall k < m$, $FL(G_k) = FL(G_{k+1})$ every time performing some transformations over some pairs of categories that contradict Eq. (1). To transform assignments that do not fulfill Eq. (1), we need new basic categories and we must also introduce new assignments. Therefore, the counterpart of this result is that the grammar G' can be much larger w.r.t. to the number of assignments than the grammar G. In fact the explosion can be enormous w.r.t. the number of assignments (we are not yet able to give a satisfying evaluation), but linear w.r.t. the number of new basic categories introduced [3]. This transformation is important from a theoretic point of view, to prove the expressivity of a class of grammars, but it is not to be applied when practical algorithms working with real data are executed.

We illustrate the construction by a small example.

$$\text{Let } G_0 = \left\{ \begin{array}{lll} \langle a, /(B,S)\rangle, & \langle a, \backslash(B,S)\rangle, & (1) \\ \langle b, /(A,B)\rangle, & \langle b, \backslash(A,B)\rangle, & (2) \\ \langle e, /(/(B,S),S)\rangle, & & \\ \langle d, /(B,S)\rangle, & \langle c, A\rangle & \end{array} \right\}$$

For this grammar G_0 we have 2 pairs of assignments (denoted by (1) and (2)) that contradict Eq. (1). We begin by applying a transformation on the first pair: we introduce two basic categories $B_f, B_b \in \mathcal{B}$ in order to replace occurrences of B in the argument position of the two categories. The two basic categories B_f, B_b are new categories for G_0, that is to say that they do not occur in any assignment of G_0. In order to memorize which categories have been introduced

[3] In fact, the number of new basic categories introduced is equal to the number of assignments in the initial grammar that, pairwise contradict the Eq. (1)

and their corresponding category in the initial grammar we define a mapping Φ that maps B_f and B_b to B.

But this replacement has two consequences. First, if a sequence $w \in \Sigma^*$ is reduced by G_0 into B then wa aw are in $L(G_0)$. Therefore, to preserve the set of correct sentences, for every category in the grammar where B occurs at a result position, we add new assignments where these occurrences are replaced respectively by B_f and B_b. Second, if a sequence $w \in \Sigma^*$ is reduced by G_0 into $/(B, S)$, then ew is also in $L(G_0)$. Again, to preserve the set of correct sentences, for every assignment $\langle l, C\rangle$ in G_0 where $/(B, S)$ occurs in C, we need to add a new assignment $\langle l, C\rangle'$ where $/(B, S)$ has been replaced by $/(B_f, S)$ (the same stands for $\backslash(B, S)$). This leads to the grammar:

$$G_1 = \left\{ \begin{array}{lll} \langle a, /(B_f, S)\rangle, \quad \langle a, \backslash(B_b, S)\rangle, & & \\ \langle b, /(A, B)\rangle, \quad \langle b, \backslash(A, B)\rangle, & & (2a) \\ \langle b, /(A, B_f)\rangle, \quad \langle b, \backslash(A, B_f)\rangle, & & (2b) \\ \langle b, /(A, B_b)\rangle, \quad \langle b, \backslash(A, B_b)\rangle, & & (2c) \\ \langle e, /(/(B, S), S)\rangle, \quad \langle e, /(/(B_f, S), S)\rangle & & \\ \langle d, /(B, S)\rangle, \quad \langle d, /(B_f, S)\rangle, \quad \langle c, A\rangle & & \end{array} \right\}$$

Now, the replacement process repeats with the grammar G_1. In G_1 there are no more assignments for the symbol a that contradicts Eq. (1), but the transformation has introduced four assignments for the symbol b that also contradicts Eq. (1). Fortunately, couples of assignments (2a), (2b) and (2c) can be processed simultaneously, *i.e.* only two new basic categories A_f and A_b are necessary. This is true because the three pairs in G_1 have been obtained from the unique couple (2) of G_0 and this can be checked using Φ. This trick is essential in our proof to obtain the termination of the replacement process.

$$G_2 = \left\{ \begin{array}{llll} \langle a, /(B_f, S)\rangle, \quad \langle a, \backslash(B_b, S)\rangle, & & & \\ \langle b, /(A_f, B)\rangle, \quad \langle b, \backslash(A_b, B)\rangle, & & & \\ \langle b, /(A_f, B_f)\rangle, \quad \langle b, \backslash(A_b, B_f)\rangle, & & & \\ \langle b, /(A_f, B_b)\rangle, \quad \langle b, \backslash(A_b, B_b)\rangle, & & & \\ \langle e, /(/(B, S), S)\rangle, \quad \langle e, /(/(B_f, S), S)\rangle & & & \\ \langle d, /(B, S)\rangle, \quad \langle d, /(B_f, S)\rangle, \quad \langle c, A\rangle, & \langle c, A_f\rangle, & \langle c, A_b\rangle & \end{array} \right\}$$

and we obtain that G_2 is in $\mathcal{G}_{Type}$.

We now enlighten the properties satisfied by couples of assignments that must be processed simultaneously. In the algorithm we will apply a transformation step for sets that are *type-indistinguishable* w.r.t Φ, G and maximal in size.

Definition 2 *Let G and G' be two CCGs and let Φ be a mapping from $\mathcal{B}(G')$ into $Cat_{\mathcal{B}}(G)$ such that $\Phi(G') = G$. Let Γ be a set of couples of assignments $\{(\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n)\}$ such that $\alpha_i, \beta_i \in G'$ for every $1 \leq i \leq n$. We say that Γ is* type-indistinguishable *w.r.t Φ, G if*
there exists:

$a \in \Sigma$, *two contexts* C, C' *and* $A, B, A', B' \in Cat_{\mathcal{B}}(G)$, *and*

$\forall i \in \{1, \ldots, n\}$, $A_i, B_i, A'_i, B'_i \in Cat_{\mathcal{B}}(G')$, *and* $2 \times n$ *contexts* C_i, C'_i
such that

$h(C) = h(C'), h(A) = h(A'), h(B) = h(B')$, *and*
$\langle a, C[/(A, B)]\rangle \in G$ *and* $\langle a, C[\backslash(A', B')]\rangle \in G$

$$\forall i \in \{1,\ldots,n\},\ \alpha_i = \langle a, C_i[/(A_i,B_i)]\rangle,\ \beta_i = \langle a, C'_i[\backslash(A'_i,B'_i)]\rangle$$
$$\begin{array}{lll} h(C_i) = h(C'_i), & & \Phi(C_i) = C,\ \Phi(C'_i) = C', \\ h(A_i) = h(A'_i), & \textit{and} & \Phi(A_i) = A,\ \Phi(A'_i) = A', \\ h(B_i) = h(B'_i) & & \Phi(B_i) = B,\ \Phi(B'_i) = B'. \end{array}$$

A sketch of proof for the theorem 1 can be now given. We need to prove three things: given as input an initial grammar $G_i \in \mathcal{G}$ the algorithm terminates, it ultimately outputs a final grammar $G_f \in \mathcal{G}_{Type}$ and $FL(G_i) = FL(G_f)$.

The algorithm iteratively builds a sequence of grammars $G_0, \ldots, G_m$, where $G_0 = G_i$ and $G_m = G_f$. Let n_k be the number of disjoint sets Γ maximal in size in G_k that are *type-indistinguishable* w.r.t Φ, G_i. n_0 can be easily calculated w.r.t. the initial grammar. Termination proof relies on the fact that n_k decreases at each step of the iteration. The Transform algorithm only introduces new assignments such that, if they contradict Eq. (1), then they increase the size of some type-indistinguishable sets w.r.t Φ, G_i. That is to say that no such new sets are introduced by the transformation algorithm. Moreover, when Transform applies, it suppresses such a type-indistinguishable set from the input grammar.

To prove that G_f is in $\mathcal{G}_{Type}$, we use the fact that if $n_k = 0$ for some grammar G_k in the sequence above, then G_k is in $\mathcal{G}_{Type}$.

For the last point, we first prove that $FL(G_i) \supseteq FL(G_f)$ using the fact that $\Phi(G_f) = G_i$. Indeed, the equality is invariant with the Transform algorithm and trivially true at the beginning. Then, using properties of substitutions in CCGs, to which our mappings can be assimilated (see [BP90]), we obtain the inclusion. The proof for the reverse inclusion is more technical and relies on case analysis. We develop the proof ideas in one case in this sketch of proof, other cases being similar. Since some assignments have been removed we must check that the same derivations are still possible using new assignments. Consider that G' is obtained from G by the Transform algorithm. Following notations in the Transform algorithm, $\langle a, C_i[/(A_i,B_i)]\rangle$ has been removed and replaced by $\langle a, C_i[/(A_f,B_i)]\rangle$. Consider a parse tree τ in G where A_i is not useless. Then there is two sibling nodes defining subtrees τ_1 and τ_2 labelled by A_i and $/(A_i,B_i)$ in τ. Because we have added assignments that put the basic category A_f in every category where A_i occurs in a result position, we can do the same derivation tree as τ_1 and label it by A_f. Every time $/(A_i,B_i)$ occurs in a category, a new assignment with $/(A_f,B_i)$ is introduced, therefore we can do the same derivation tree as τ_1 and label it by $/(A_f,B_i)$. Using these properties, we are able to prove that a word w is reduced in a category C in G' if and only if it is reduced in category $\Phi(C)$ in the initial grammar G_i and both derivation trees are identical up to Φ. Hence structural languages $FL(G_i)$ and $FL(G_f)$ are identical.

4 Learnability of $\mathcal{G}_{Type}$ from Typed Examples

We have already mention in the introduction the motivation of learning from Typed Examples. The previous section proved that the class $\mathcal{G}_{Type}$ has good properties relatively to the entire class $\mathcal{G}$ as it allows to produce every possible Structure Language generated by a CCG. But the initial motivation for introducing this class was that of learnability. We now justify the interest of $\mathcal{G}_{Type}$

Algorithm 1 G to GType.

Input : a CCG G_i

$\Phi = Id_{Cat_{\mathcal{B}}}$ is the identity on $Cat_{\mathcal{B}}$.
$G' = G_i$
while There exists Γ in G', a maximal type-indistinguishable set w.r.t Φ, G_i **do**
  (G', Φ) = Transform(G',G_i,Φ,Γ)
end while

Output : G'

Algorithm 2 Transform.

Input : Two grammars G and G_i, a mapping Φ and Γ a *type-indistinguishable* set w.r.t Φ, G_i.
{With Γ we assume as indicated in Def. 2 $a \in \Sigma$, two contexts C, C' and $A, B, A', B' \in Cat_{\mathcal{B}}(G_i)$, and $\forall i \in \{1, \ldots, n\}$, $A_i, B_i, A'_i, B'_i \in Cat_{\mathcal{B}}(G)$, and $2 \times n$ contexts C_i, C'_i.}

$G' = G - \bigcup_i \{\langle a, C_i[/(A_i, B_i)]\rangle, \langle a, C'_i[\backslash(A'_i, B'_i)]\rangle\}$;
Let A_f and A_b be two basic categories not in $\mathcal{B}(G)$;
$G' = G' \cup \bigcup_i \{\langle a, C_i[/(A_f, B_i)]\rangle, \langle a, C'_i[\backslash(A_b, B'_i)]\rangle\}$;
Remaining $= G'$; {*Remaining* is the set of assignments that remains to be processed}
while *Remaining* $\neq \emptyset$ **do**
  {Result position}
  Pick and remove $\langle b, D[W]\rangle$ from *Remaining*;
  if $W = f(W', A_i)$ for some $i \in \{1, \ldots, n\}$ and $f \in \{\backslash, /\}$ **then**
    Add $\langle b, D[f(W', A_f)]\rangle$ to G' and to *Remaining*;
  end if
  if $W = f(W', A'_i)$ for some $i \in \{1, \ldots, n\}$ and $f \in \{\backslash, /\}$ **then**
    Add $\langle b, D[f(W', A_b)]\rangle$ to G' and to *Remaining*;
  end if
  if $D[W] = A_i$ **then**
    Add $\langle b, A_f\rangle$ to G' and to *Remaining*;
  end if
  if $D[W] = A'_i$ **then**
    Add $\langle b, A_b\rangle$ to G' and to *Remaining*;
  end if
  {Any position }
  if $W = /(A_i, B_i)$ **then**
    Add $\langle b, D[/(A_f, B_i)]\rangle$ to G' and to *Remaining*;
  end if
  if $W = \backslash(A'_i, B'_i)$ **then**
    Add $\langle b, D[\backslash(A_b, B'_i)]\rangle$ to G' and to *Remaining*;
  end if
end while
$\Phi'(A_f) = A$, $\Phi'(A_b) = A'$ and $\Phi'(X) = \Phi(X)$ for any $X \notin \{A_f, A_b\}$.

Output : G' and Φ'.

in terms of formal learning theory, especially from the learning from Typed Examples point of view and argue for its plausibility to model natural language learning. We also provide a discovery procedure adapted to the class $\mathcal{G}_{Type}$ having Typed Examples as input and providing CCGs as output.

4.1 Grammar Systems and Learnability

For any CCG G, a Canonical Typed Example for G is a sequence of couples $\langle word, type \rangle$ where the first items of the couples build a sentence of G and the second items are the types corresponding with the categories assigned to each word and allowing the syntactic analysis in G. We define the Canonical Typed Language of G:

$TL(G) = \{\langle u_1, \tau_1 \rangle ... \langle u_n, \tau_n \rangle | \forall i \in \{1, ..., n\} \exists c_i$ so that $\langle u_i, c_i \rangle \in G$, $\tau_i = h(c_i)$ and $c_1 ... c_n \rightarrow^* S\}$.

To deal with questions of learnability, Kanazawa [Kan96, Kan98] introduces the notion of grammar system. This allows a reformulation of the classical Gold's model of *identification in the limit from positive examples* [Gol67]. We recall this notion here and its connexion with the domain of learnability theory.

A grammar system is a triple $\langle \Omega, \Lambda, L \rangle$ where Ω is the hypothesis space (in our context, Ω will be a set of formal 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) and L is a naming function that maps elements of Ω into languages i.e. $L : \Omega \rightarrow pow(\Lambda)$. The universal membership problem, i.e. the question of whether $s \in L(G)$ holds between $s \in \Lambda$ and $G \in \Omega$, is supposed computable.

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 $\langle s_i \rangle_{i \in \mathbb{N}}$.

Kanazawa has proved that $\forall k \geq 1, \mathcal{G}_k$ is learnable both in the grammar system $\langle \mathcal{G}, \Sigma^F, FL \rangle$ (i.e from Structural Examples) and in the grammar system $\langle \mathcal{G}, \Sigma^*, L \rangle$ (i.e. from string examples).

In the formal learning model of Gold, learnability results for CCGs become trivial when typed examples are given in input. Indeed, there are a bounded number of compatible grammars with any finite presentation as soon as all elements of the lexicon have been presented. Membership is decidable and therefore any simple enumerative algorithm of compatible grammars can be easily transformed into a learner. But this is not satisfactory and more interesting remarks can be done for $\mathcal{G}_{Type}$ grammars. Indeed, we notice that to learn $\mathcal{G}_{Type}$ in the grammar system $\langle \mathcal{G}, (\Sigma \times Types(\mathcal{B}))^*, TL \rangle$, it is enough to be able to learn $\mathcal{G}_1$ in the grammar system $\langle \mathcal{G}, (\Sigma \times Types(\mathcal{B}))^*, L \rangle$. As a matter of fact, grammars G in $\mathcal{G}_{Type}$ are such that for all pairs $\langle u, \tau \rangle \in \Sigma \times Types(\mathcal{B})$ belonging to a member of $TL(G)$, there exists only one c so that $\langle u, c \rangle \in G$ and $h(c) = \tau$ and are thus one to one distinct. On the vocabulary $\Sigma \times Types(\mathcal{B})$, these grammars are thus rigid.

This suggests a learning algorithm which would be an adaptation of the one that learns $\mathcal{G}_1$ from strings. Unfortunately, this strategy would not be efficient, since learning $\mathcal{G}_1$ from strings is NP-hard [Flo02]. Another possible solution is to notice the finite relation that connects Typed Examples and Structural Examples (from a given Typed Example, there is a finite number of compatible Structural Examples). But the combinatorial explosion between them is still exponential. Furthermore, none of these strategies takes advantage of the functional nature of types and of their closeness with categories. In the following, we propose a learning algorithm adapted to Typed Examples inputs.

4.2 A learning algorithm from types

This algorithm is described in full details in [**?**]. We simply introduce here its key idea. Types can combine following rewriting rules similar with the ones defined for the categories of a CCG. For every type Y and X in $\mathit{Types}(\mathcal{B})$ we have:

- Type Forward **TF**: $(Y, X)\ Y \rightarrow X$;
- Type Backward **TB**: $Y\ (Y, X) \rightarrow X$.

Lemma 1 *For any CCG G, we have the following property for any categories X and Y in $Cat_{\mathcal{B}}(G)$:*

- *if FA applies between $/(Y, X)$ and Y to give X then TF applies between $h(/(Y, X)) = (h(Y), h(X))$ and $h(Y)$ to give $h(X)$;*
- *if BA applies between Y and $\backslash(Y, X)$ to give X then TB applies between $h(Y)$ and $h(\backslash(Y, X)) = (h(Y), h(X))$ to give $h(X)$.*

The main syntactic difference between categories of a Categorial Grammar and their types in $\mathit{Types}(\mathcal{B})$ related by h is that the *direction* of a functor-argument application in categories is indicated by the operator used ($/$ or $\backslash$), whereas this direction is lost in types. The functor-argument application of types is commutative, whereas it is not for categories. When input data are Typed Examples, what remains to be learned is thus the direction of every implicit operators ($/$ or $\backslash$) in types.

We assume that every sentence given as input is syntactically correct and thus (with lemma 1) that the associated sequence of types can be reduced using **TF** and **TB** to the axiom S (Fig. 2).

As the whole learning process is the search for introducing operators in type expressions to get categories, we introduce distinct variables into types expressions to receive the operator values. Let $\mathcal{X}$ be a countable set of variables denoted by $x_1, \ldots, x_n, \ldots$ Possible final values for these variables are $\backslash$ or $/$. Let a substitution σ be a mapping from $\mathcal{X}$ to $\{/, \backslash\}$.

We define $\mathit{VarType}(\mathcal{B})$, the set of Variable Canonical Types as the smallest set such that $\mathcal{B} \subset \mathit{VarType}(\mathcal{B})$ and for any $U \in \mathit{VarType}(\mathcal{B})$, $V \in \mathit{VarType}(\mathcal{B})$ and $x_i \in \mathcal{X} \cup \{/, \backslash\}$ we have $x_i(U, V) \in \mathit{VarType}(\mathcal{B})$. Any substitution σ can be naturally extended to elements of $\mathit{VarType}(\mathcal{B})$ in the following way: for every X in $Cat_{\mathcal{B}} \cup \{/, \backslash\}$, $\sigma(X) = X$ and for any $U \in \mathit{VarType}(\mathcal{B})$, $V \in \mathit{VarType}(\mathcal{B})$, $\sigma(x_i(U, V)) = \sigma(x_i)(\sigma(U), \sigma(V))$. Two elements U and U' in $\mathit{VarType}(\mathcal{B})$ are unifiable if there exists a substitution σ such that $\sigma(U) = \sigma(U')$. Each reduction

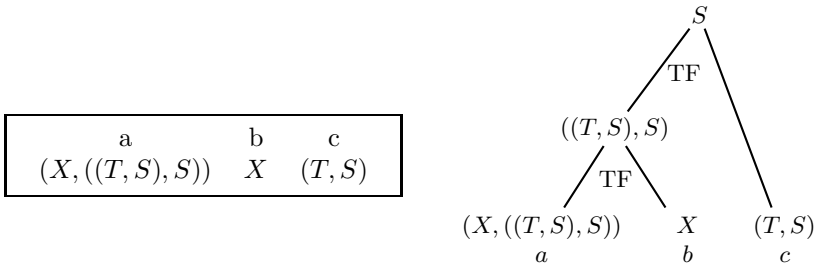

Figure 2: Example of reduction for types using **TF** and **TB**.

via **TF** or **TB** allows to identify the value of a variable (the one of the type at a functor position) and implies some identities, up to some substitution, between others. Hence, type reductions allow to identify substitutions and a parse leads to a set of substitutions. In Figure 3 the unifiable subterms are underlined for each reduction and there exists a substitution σ such that $\sigma(x_1) = /$ for the first reduction and $\sigma(x_4) = \sigma(x_2), \sigma(x_3) = \backslash$ for the second reduction.

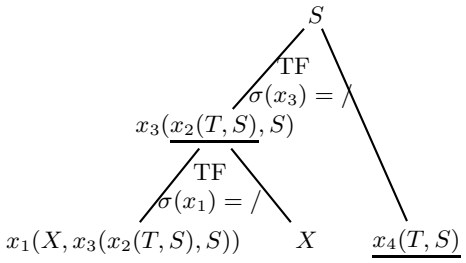

Figure 3: A parse tree underling identity up to a substitution between sub-types and identification of variable values

Note that there could be more than one parse for a given input. The outcome of the parser is thus a set of parses. Similarly, the output of learning process for a single input is a set of sets of substitutions, each of which defines a grammar.

From the algorithm described, we can derive an incremental process. Indeed, the parse algorithm outputs a set of substitutions that can be applied to the next sentence given in input. The fact that that the target grammar(s) is(are) in $\mathcal{G}_{Type}$ allows to associate the same variables to a given type associated with a given word and thus renders the process incremental. For an input sample S of Typed Example, our algorithm parses types and computes a set of grammars $G_1, \ldots, G_n$ in $\mathcal{G}_{Type}$ consistent with S. However, it is not shown that every output grammar G_i is minimal with respect with string language inclusion among the set of grammars in $\mathcal{G}_{Type}$ consistent with S. Indeed, we have not proved that there is no $G \in \mathcal{G}_{Type}$ such that $L(G) \subsetneq L(G_i)$ and $TL(G)$ equals $TL(G_i)$ when restricted to S. Even though this algorithm could be followed by an inclusion test to select minimal grammars among $G_1, \ldots, G_n$, this is not satisfactory from a complexity point of view. Worth, we do not know whether

inclusion is decidable for typed languages. Hence, there is no guarantee that no over-generalization occurs.

The way proposed by Kanazawa to learn k-valued (and rigid) categorial grammars from strings implies to build all possible structural examples for input sentences and then to use his learning algorithm from structural examples. There is therefore a combinatorial explosion because the number of all structural examples compatible with a sentence of m words is exponential in m.

For our algorithm, the situation is slightly different. A parse in a context free grammar in Chomsky Normal Form can be computed in polynomial time in the size of the input. This complexity result is stated for a given context free grammar. The type combination rules **TF** and **TB** instantiated by types in the input sequence are in Chomsky Normal Form, so, at first view our algorithm acts like a simple parse algorithm. But, in our setting, we do not have a context free grammar but a "set of possible ones" defined by schemes of rules **TF** and **TB**. This changes a lot the complexity issues as illustrated by the following example:

Let us consider the typed example associated with a string $a^n b a^n$ where a has type e and b has a type with $2n$ arguments of type e:

$$\langle a, e\rangle^n \langle b, (e, (e, ...(e, t)))\rangle \langle a, e\rangle^n.$$

One can parse such a string in a context free grammar in polynomial time with respect to the size of the input sequence. But there exist $\binom{2n}{n}$ different context free grammars compatible with this input. As a matter of fact the type associated with b expects $2n$ arguments among which n are on its right and n are on its left. Since our algorithm builds all possible grammars, it will run in exponential time with respect to n. Nonetheless, note that this n does not depend on the number of words of the input. It is related to the lexicon and to the size of types associated with words. If for instance b occurs m times in a sentence, the algorithm will run in $O(m^3 * 2^n)$ and not in $O(2^{km})$. Moreover, for practical situations several heuristics and remarks could be done in this framework. For instance, choosing only one grammar among the set of consistent ones could circumvent the problem detailed in the example above. Also, for natural language grammars, the size of types and categories is not so large and therefore the combinatorial explosion due to the size of types is not dramatic. Finally, the way of presenting examples to the learner could have important consequences from the complexity point of view. An idea is to consider helpful presentations — *e.g.* presentations that avoid a large number of new words in a unique sentence.

5 Conclusion

Learning a Categorial Grammar means associating categories with words. Categories are built from basic categories and operators. Learning categories from strings of words seems impossible in reasonable time. So, richer input data need to be provided. The approach developed so far by Buskowsky & Penn and Kanazawa consisted in providing Structural Examples, i.e. giving the nature of the operators / and \ and letting the basic categories to be learnt. Structural Examples give information about the global syntactic analysis of a sentence. But one of the most interesting property of Categorial Grammars is their lexicalized nature. This property is lost in the notion of Structural Example.

In contrast, our approach consists in providing types, i.e. indications about the categories assigned to each word, but where the operators / and \ are deleted. Types are lexicalized structural information and thus seem more relevant to learn lexicalized grammars than structural information given at the sentence level. Moreover, we have seen that types can also be considered as lexical semantic information and are thus arguably learned by children before the grammar of their mother tongue.

The main contribution of this paper is the definition of a precise subclass of $\mathcal{G}$ which is both learnable under conditions and has good properties from a language-theoretic point of view, as the members of this class allow to generate every possible language and every possible Structure Language that a CCG can generate. In this sense, $\mathcal{G}_{Type}$ is *representative* of $\mathcal{G}$ and grammars in this class can be considered as CCGs of a special normal form.

Of course, the learnability of grammars in this class is guaranteed only if unambiguous types are provided, which is a strong condition. But this result can be compared to other results of the same kind in grammatical inference. For example, Sakakibara proved that every context-free language could be generated by a reversible context-free grammar and that the set of reversible context-free grammars is learnable from skeletons, i.e. from syntactic analysis trees where non-terminal symbols are deleted. This result is interesting but limited because transforming a plain context-free grammar into a reversible context-free grammar recognizing the same string language does not preserve the corresponding set of skeletons. On the contrary, our transformation is structure-preserving, which is a crucial condition if one considers that structures are the basis for semantics.

Acknowledgements

This research was partially supported by: "CPER 2000-2006, Contrat de Plan état - région Nord/Pas-de-Calais: axe TACT, projet TIC"; fonds européens FEDER "TIC - Fouille Intelligente de données - Traitement Intelligent des Connaissance" OBJ 2-phasing out - 2001/3 - 4.1 - n 3; ARC INRIA Gracq et Maison des Sciences de l'Homme -Institut International Erasme.

References

[Adr92] P. W. Adriaans. *Language Learning from a Categorial Perspective*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, 1992.

[BHGS60] Y. Bar Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars. *Bulletin of the Research Council of Israel*, 9F, 1960.

[BM02] J. Besombes and J.Y. Marion. Apprentissage des langages réguliers d'arbres et applications. In *CAP'2002*, pages 55–70, 2002.

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

[BR01] R. Bonato and C. Rétoré. Learning rigid Lambek grammars and minimalist grammars from structured sentences. In *Proceedings of LLL01*, pages 23–34, 2001.

[FlN02] A. Foret and Y. le Nir. On limit points for some variants of rigid lambek grammars. In M. Van Zaanen P. Adriaans, H. Fernau, editor, *Grammatical Inference: Algorithms and Applications*, volume 2484 of *Lecture Notes in Artificial Intelligence*, pages 106–119. Springer Verlag, 2002.

[Flo01] C. Costa Florêncio. Consistent identification in the limit of any of the classes k-valued is NP-hard. In *Logical Aspects of Computational Linguistics*, volume 2099 of *Lecture Notes in Artificial Intelligence*, pages 125–134. Springer Verlag, 2001.

[Flo02] C. Costa Florêncio. Consistent identification in the limit of rigid grammars from strings is np-hard. In M. Van Zaanen P. Adriaans, H. Fernau, editor, *Grammatical Inference: Algorithms and Applications*, volume 2484 of *Lecture Notes in Artificial Intelligence*, pages 49–62. Springer Verlag, 2002.

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

[Jan97] T. M. V. Janssen. Compositionality. In J. V. Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, pages 417–473. MIT Press, 1997.

[Kan96] M. Kanazawa. Identification in the limit of categorial grammars. *Journal of Logic, Language and Information*, 5(2):115–155, 1996.

[Kan98] M. Kanazawa. *Learnable Classes of Categorial Grammars*. The European Association for Logic, Language and Information. CLSI Publications, 1998.

[Par90] B. Partee. *Mathematical methods in Linguistics*. Number 30 in Linguistics and Philosophy. Kluwer, 1990.

[Tel99] I. Tellier. Towards a semantic-based theory of language learning. In *proceedings of the 12th Amsterdam Colloquium*, pages 217–222, 1999.