# Chapter 1

# A Learnable Class of CCGs from Typed Examples

DANIELA DUDAU-SOFRONIE, ISABELLE TELLIER, MARC TOMMASI

## 1.1 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. The simplest instance of this large family is known as AB-Categorial Grammars, or Classical Categorial Grammars (CCG in the following). Although too rudimentary 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 (Kanazawa, 1998). But these results do not provide tractable learning algorithms, as most of the problems to be solved are NP-hard (Florêncio, 2001; Florêncio, 2002). The only favorable case is when rigid CCGs are to be learned 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 are thus more "naturally available" than Structural Examples.

The availability of the Typed Examples can be argued firstly from a theoretical point of view, exploiting the compositionality and secondly by the efforts that are done to build up a natural language resource (corpus) from texts enriched with semantic information. As a matter of fact, the connection of Categorial Grammars with semantics relies on a formal statement of the well known Principle of Compositionality that states : *"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"* (Partee, 1990). If the *"parts"* 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. This paper presents a new way of considering learning Categorial Grammars from semantic knowledge. We make the hypothesis that semantic types, in the usual sense, are general information making a distinction between facts,



*Proceedings of Formal Grammar 2003*
G. Jäger, P. Monachesi, G. Penn & S. Wintner (editors).

.

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 we need corpora of typed texts. Such corpora are not available and they have to be built. As 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, for example some post-treatment needs to be done, as well as for conjunctions, etc.). 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, canonical 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. The fifth section concludes.

## 1.2 A New Subclass of Classical Categorial Grammars

### 1.2.1 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 $\Sigma$ be a fixed alphabet called vocabulary. A **categorial grammar over** $\Sigma$ is any finite relation between $\Sigma$ 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[1]:

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

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

[1]These rules justify the fractional notations of the category-building operators $/$ and $\backslash$ 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 $B$. In this paper, we do not use this notation because of some constructions.

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 $G$.

### 1.2.2 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 $\backslash$ 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.

### 1.2.3 The Class $\mathcal{G}_{Type}$

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

$$\forall \langle a, A\rangle \in G,\ \langle a, A'\rangle \in G \quad h(A) = h(A') \Rightarrow A = A'. \tag{1.1}$$

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.* 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 1.1.

The set of languages generated (or recognized) by CCGs is the set of $\epsilon$-free context-free languages, where $\epsilon$ is the empty word (Gaifman's theorem in Hillel et al. (1960)). Now that we have defined a subclass of $\mathcal{G}$, it is natural to wonder if its members allow the generation of the same set of languages or only a subset of it.

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

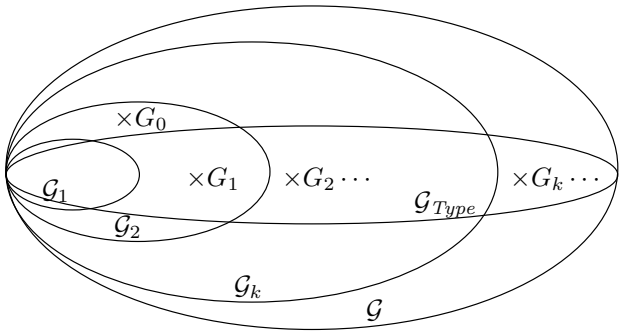



Figure 1.1: Comparisons between classes of $k$-valued categorial grammars and the $\mathcal{G}_{Type}$ class.

*Proof.* It has long been recognized that Gaifman's proof is equivalent with the existence of the strong Greibach normal form for $\epsilon$-free context-free languages. That is, for every $\epsilon$-free context-free language $L \subset \Sigma^*$ there exists a phrase structure grammar $G' = \langle \Sigma, NT, P, S \rangle$ (where $\Sigma$ 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 $\Sigma$. 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.

## 1.3 Structure Languages of $\mathcal{G}_{Type}$

A functor-argument structure over an alphabet $\Sigma$ is a binary-branching tree whose leaf nodes are labeled by elements of $\Sigma$ and whose internal nodes are labeled either by $BA$ or $FA$. The set of functor-argument structures over $\Sigma$ is denoted $\Sigma^F$. Let $G$ be a CCG. A *Structural Example* for $G$ is an element of $\Sigma^F$ 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 (Buszkowski and Penn, 1990; Kanazawa, 1998). Furthermore, it allows the connection between syntax and semantics (Tellier, 1999). 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 $BA(FA(a, b), a)$.

### 1.3.1 The Main Result

**Theorem 1.** *For any vocabulary $\Sigma$ 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. (1.1) 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. (1.1). To transform assignments that do not fulfill Eq. (1.1), 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 number of assignments than the grammar $G$. We illustrate the construction by a small example.

*Example 3.*

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

For this grammar $G_0$ we have 2 pairs of assignments (denoted by (1) and (2)) that contradict Eq. (1.1). 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 and their corresponding category in the initial grammar we define a mapping $\Phi$ 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:

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

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. (1.1), but the transformation has introduced four assignments for the symbol $b$ that also contradicts Eq. (1.1). Fortunately, pairs 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 pair (2) of $G_0$ and this can be checked using $\Phi$. This trick is essential in our proof to obtain the termination of the replacement process.

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

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

We now enlighten the properties satisfied by pairs 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 $\Phi, G$ and maximal in size.

**Definition 2.** *Let $G$ and $G'$ be two CCGs and let $\Phi$ be a mapping from $\mathcal{B}(G')$ into $Cat_{\mathcal{B}}(G)$ such that $\Phi(G') = G$. Let $\Gamma$ be a set of pairs 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 $\Gamma$ is* type-indistinguishable *w.r.t $\Phi, G$ 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* $\langle a, C[\backslash(A', B')]\rangle \in G$
$\forall i \in \{1, \ldots, n\},\ \alpha_i = \langle a, C_i[/(A_i, B_i)]\rangle,$
$\beta_i = \langle a, C'_i[\backslash(A'_i, B'_i)]\rangle$

$$\begin{array}{lll} h(C_i) = h(C'_i), & & \Phi(C_i) = C,\ \Phi(C'_i) = C', \\ h(A_i) = h(A'_i), & \textit{and} & \Phi(A_i) = A,\ \Phi(A'_i) = A', \\ h(B_i) = h(B'_i) & & \Phi(B_i) = B,\ \Phi(B'_i) = B'. \end{array}$$

**Lemma 1.** *Let $G_0, \ldots, G_m$ be the sequence of grammars built by the algorithm 1 G to GType, where $G_0 = G_i$ and $G_m = G_f$ and $n_k$ the number of disjoint sets $\Gamma$ maximal in size in $G_k$ that are* type-indistinguishable *set w.r.t $\Phi, G_i$. If at step $k$ we transform $G_k$ in $G_{k+1}$ by applying the algorithm Transform then $n_{k+1} = n_k - 1$.*

*Proof.* In fact, we have to prove that the Transform algorithm 2 doesn't introduce new sets $\Gamma$ but only new assignments such that, if they contradict Eq. (1.1), then they increase the size of some already existent type-indistinguishable set w.r.t $\Phi, G_i$. We do that by verifying at each step of the algorithm the changes. The step (1) eliminates all the assignments of the set $\Gamma$ (suppressing it), so the number $n_{k+1}$ of sets type-indistinguishable w.r.t.$\Phi', G_i$ for $G_{k+1}$ will decrease with one compared to $n_k$. All other steps will only allow the introduction of some new assignments as follows:

- at step (3) the assignments introduced cannot be in a type-indistinguishable set because for each $i$, each pair of types are not equal anymore as $A_f$ and $A_b$ are new categories just introduced and there will be no pairs in $G_{k+1}$ as to form a new type-indistinguishable set.

- steps (6) and (9) are similar. In fact here, if there exist in $G_k$ two assignments $\langle b, D[/(W', A_i)]\rangle$ and $\langle b, D[\backslash(W', A_i)]\rangle$ that are part of some type-indistinguishable set w.r.t.$\Phi, G_i$ then, by the step (6) we'll add two new assignments in this set with $A_i$ replaced by $A_f$; if not, we'll add some new assignments that are not part of any set type-indistinguishable. The same we induce for the step (9).

**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 $\Gamma$ in $G'$, a maximal type-indistinguishable set w.r.t $\Phi, G_i$ **do**
    ($G'$,$\Phi$) = Transform($G'$,$G_i$,$\Phi$,$\Gamma$)
**end while**

**Output :** $G'$

**Algorithm 2** Transform.

**Input :** Two grammars $G$ and $G_i$, a mapping $\Phi$ and $\Gamma$ a *type-indistinguishable* set w.r.t $\Phi, G_i$.
{With $\Gamma$ 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 $\mathcal{B}(G)$;
$G' = G' \cup \bigcup_i \{\langle a, C_i[/(A_f, B_i)]\rangle, \langle a, C'_i[\backslash(A_b, B'_i)]\rangle\}$;
*REM* $= G'$; {*REM* is the set of assignments that remain to be processed}
**while** *REM* $\neq \emptyset$ **do**
    {Result position}
    Pick and remove $\langle b, D[W]\rangle$ from *REM*;
    **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 *REM*;
    **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 *REM*;
    **end if**
    **if** $D[W] = A_i$ **then**
        Add $\langle b, A_f\rangle$ to $G'$ and to *REM*;
    **end if**
    **if** $D[W] = A'_i$ **then**
        Add $\langle b, A_b\rangle$ to $G'$ and to *REM*;
    **end if**
    {Any position }
    **if** $W = /(A_i, B_i)$ **then**
        Add $\langle b, D[/(A_f, B_i)]\rangle$ to $G'$ and to *REM*;
    **end if**
    **if** $W = \backslash(A'_i, B'_i)$ **then**
        Add $\langle b, D[\backslash(A_b, B'_i)]\rangle$ to $G'$ and to *REM*;
    **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 $\Phi'$.

- for the steps (13) and (16) the assignments are not part of any set type-indistinguishable and don't create a new such set.
- for the steps (19) and (22), if $\langle b, D[/(A_i, B_i)]\rangle \in G$ and $\langle b, D[\backslash(A'_i, B'_i)]\rangle \in G$ belong to some set $\Gamma'$ type-indistinguishable for $G_k$, then the new added assignments don't belong to $\Gamma'$ because the types associated to the new introduced categories are different. If $/(A_i, B_i)$ is contained in an expression $\langle b, D[/(/(A_i, B_i), Y]\rangle$ and respectively $\langle b, D[\backslash(/(A_i, B_i), Y)]\rangle$ that already belong to a type-indistinguishable set, then the new assignments added belong also to this type-indistinguishable set. Idem for $\backslash(A'_i, B'_i)$.

□

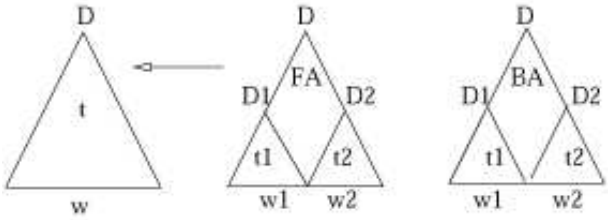
**Corollary 1.** *If* $n_k = 0$ *then* $G_k = G_f \in \mathcal{G}_{Type}$.

**Lemma 2.** *Let us consider the transformation algorithm 2 with the input* $\Phi$, $G$,$G_i$, $\Gamma$ *and the output* $G'$ *and* $\Phi'$. *If* $w \in \Sigma^+$ *such that* $w$ *has a partial parse tree* $t$ *in the grammar* $G$ *and* $t$ *has as root the category* $D \in Cat_{\mathcal{B}}(G)$, *then the following is true:* $\forall D' \in Cat_{\mathcal{B}}(G')$ *such that* $\Phi'(D') = D$, $\exists t'$ *parse tree in* $G'$, $t'$ *having a structure*[2] *identical to* $t$, *such that* $w$ *has the partial parse* $t'$ *in* $G'$ *and* $t'$ *has the root* $D'$.

*Proof.* For the proof, we proceed by induction on the size $|w|$ of $w \in \Sigma^+$, $|w|$ representing the number of symbols in the word $w$.

If $|w| = 1$ then the partial parse tree $t$ has a single node (root) $D$. If $D$ doesn't contain any of $A_i$ and $A'_i$, $\forall i$ then $D$ isn't affected by the transformation $\Phi'$ and $t'$ is $t$ q.e.d. If $D$ contains $A_i$ or $A'_i$ then $\forall D'$ with the propriety in the hypothesis there exists a partial parse tree $t'$ with a single node, in fact the old partial parse tree $t$ that has now the node (root) $D'$ q.e.d.

If $|w| > 1$ we suppose the propriety true for all $|w| < k$ and we prove it for $|w| = k$.




The hypothesis is true for all $|w| < k$ implies that for a $w_1 \in \Sigma^+$, $|w_1| < k$ that has a partial parse tree $t_1$ in $G$, with the root $D_1$ we have the following propriety: $\forall D'_1$ such that $\Phi'(D'_1) = D_1$ there exists $t'_1$, partial parse tree in $G'$, having the same structure as $t_1$ and rooted in $D'_1$. The same thing we can induce for a $w_2$, $|w_2| < k$. If in $G$ we apply a rule on $D_1$ and $D_2$ to obtain $D$, where $w = w_1.w_2$ (by concatenation) ($|w| = k$) has a partial parse $t$, with the root $D$, then we prove that $\forall D'$ such that $\Phi'(D') = D$ it exists a partial parse tree $t'$ in $G'$ with the root $D'$ and where $D'$ is obtained by applying the same rule on $D'_1$ and $D'_2$. This last fact expresses that $t'$ will have the same structure as $t$, knowing by hypothesis that the partial parse trees $t_1$ and $t_2$ in $G$ have the same structure as $t'_1$ and $t'_2$ in $G'$. We distinguish here all the possible cases:

1. $D_1$ and $D_2$ don't contain $A_i$ and $A'_i$, $\forall i$;

[2]Here, structure means functor-argument structure

2. $D_1 = /(A_i, Y)$ and $D_2 = A_i$ or $D_1 = /(A'_i, Y)$, $D_2 = A'_i$ for some $i$ (when $FA$ was applied) or $D_1 = A_i$, $D_2 = \backslash(A_i, Y)$ or $D_1 = A'_i$, $D_2 = \backslash(A'_i, Y)$ for some $i$ (when $BA$ was applied), where $Y$ doesn't contain $A_i$ and $A'_i$;

3. $D_1 = /(Z, A_i)$, $D_2 = Z$ or $D_1 = /(Z, A'_i)$, $D_2 = Z$ (for $FA$) or $D_1 = Z$,$D_2 = \backslash(Z, A_i)$ or $D_1 = Z$, $D_2 = \backslash(Z, A'_i)$ (for $BA$), where $Z$ doesn't contain $A_i$ and $A'_i$;

4. $D_1 = /(Z, Y)$, $D_2 = Z$ (for $FA$) or $D_1 = Z$, $D_2 = \backslash(Z, Y)$ (for $BA$) where $Y$ doesn't contain $A_i$ and $A'_i$, $\forall i$ but $Z$ does;

5. $D_1 = /(Z, Y)$, $D_2 = Z$ (for $FA$) or $D_1 = Z$, $D_2 = \backslash(Z, Y)$ (for $BA$) where $Z$ doesn't contain $A_i$ and $A'_i$, $\forall i$ but $Y$ does;

6. $D_1 = /(Z, Y)$, $D_2 = Z$ (for $FA$) or $D_1 = Z$, $D_2 = \backslash(Z, Y)$ (for $BA$), where both $Y$ and $Z$ contain $A_i$ and $A'_i$.

In the case (1) the conclusion is evident. In all the other cases we convey to treat only the first variant, because all the other are similar. We only treat here the cases (2) and (6). In the case (2) $D_1 = /(A_i, Y)$, $D_2 = A_i$ and $Y$ doesn't contain occurrences of $A_i$ or $A'_i$, $\forall i$. If $D_1 = /(A_i, Y)$ belong to the set $\Gamma$ that was solved, then $D'_1 = /(A_f, Y)$, where $\Phi'(A_f) = A_i$, because $Y$ doesn't contain $A_i$ and $A'_i$. Because $D'_2$ received besides $A_i$ the category $A_f$ (by the step (13) of the algorithm 2) we can apply over $D'_1$ and $D'_2$ the same rule as over $D_1$ and $D_2$ to obtain $D'$ and $t'$, the partial parse tree obtained, has the same structure as $t$, q.e.d. If $D_1 = /(A_i, Y)$ doesn't belong to $\Gamma$ we add the category $/(A_f, Y)$ for $D_1 = /(A_i, Y)$ if $Y = B_i$ (by the step (19) of the algorithm 2), or we add nothing if $Y \neq B_i$ and at the same time we add for $D_2 = A_i$ the category $A_f$, where $\Phi'(A_f) = A_i$ (by the step (13) of the same algorithm). So, $D'_1 \in \{/(A_i, Y), /(A_f, Y)\}$ or $D'_1 = \{/(A_i, Y)\}$ and $D'_2 \in \{A_i, A_f\}$ and $D' = Y$ because there is always a manner to combine $D'_1$ and $D'_2$ by applying the same rule as for $D_1$ and $D_2$. q.e.d.

In the case (6) $D_1 = /(Z, Y)$ and $D_2 = Z$, where $Z$ and $Y$ contain both $A_i$ or $A'_i$, for some $i$ and this means that the transformation $\Phi'$ affects both $Z$ and $Y$. $D'_1 \in \{/(Z', Y') | \Phi'(Y') = Y, \Phi'(Z') = Z\}$ and $D'_2 \in \{Z'' | \Phi'(Z'') = Z\}$. It is evident that we can apply the same rule as for $D_1$ and $D_2$ for every $D'_1$ and $D'_2$ ($\{Z' | \Phi'(Z') = Z\} = \{Z'' | \Phi'(Z'') = Z\}$, the modifications over the same category relative to the same transformation are identical) and we obtain for every result $D' \in \{Y' | \Phi'(Y') = Y\}$ a partial parse tree $t'$, rooted with $D'$, with a structure identical to $t$ and with $\Phi'(D') = D$ q.e.d. □

*Sketch of proof of Theorem 1* 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 $\Gamma$ maximal in size in $G_k$ that are *type-indistinguishable* w.r.t $\Phi, G_i$. $n_0$ 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 (see lemma 1). The Transform algorithm only introduces new assignments such that, if they contradict Eq. (1.1), then they increase the size of some type-indistinguishable sets w.r.t $\Phi, G_i$. 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 Buszkowski and Penn (1990)), 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 (see lemma 2), 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 $\tau$ in $G$ where $A_i$ is not useless. Then there is two sibling nodes defining subtrees $\tau_1$ and $\tau_2$ labelled by $A_i$ and $/(A_i, B_i)$ in $\tau$. 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 $\tau_1$ 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 $\tau_1$ 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 $\Phi$. Hence structural languages $FL(G_i)$ and $FL(G_f)$ are identical.

□

## 1.4 Learnability of $\mathcal{G}_{Type}$ 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 $\mathcal{G}_{Type}$ in terms of formal learning theory and argue for its plausibility to model natural language learning.

### 1.4.1 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 $TL : \mathcal{G} \rightarrow pow((\Sigma \times Types(\mathcal{B}))^*)$ the function that maps every CCG $G$ with the set $TL(G)$ of the Canonical Typed Examples it can generate, also called 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 1998 introduces the notion of grammar system. This allows a reformulation of the classical Gold's model of *identification in the limit from positive examples* (Gold, 1967). We recall this notion here.

A grammar system is a triple $\langle \Omega, \Lambda, L\rangle$ where $\Omega$ is the hypothesis space (in our context, $\Omega$ will be a set of formal grammars), the sample space $\Lambda$ is a recursive subset of $A^*$, for some fixed alphabet $A$ (elements of $\Lambda$ are sentences and subsets of $\Lambda$ are languages) and $L$ is a naming function that maps elements of $\Omega$ 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 $\phi$ converges to $G \in \Omega$ on a sequence $\langle s_i\rangle_{i \in \mathbb{N}}$ of elements of $\Lambda$ 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 $\phi$ is said to learn $\mathcal{G} \subseteq \Omega$ and $\mathcal{G}$ is said learnable 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 $\phi$ 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 (Florêncio, 2002).

Another candidate is the learning strategy proposed in Dudau-Sofronie, Tellier, and Tommasi (2001), taking advantage of the functional nature of types and of their closeness with categories. The only remaining problem is that we still do not know if the inclusion of Typed Languages is decidable for CCGs. The answer to that open problem can have a great influence on the computational complexity of the strategy.

### 1.4.2 Semantic Interpretation of Types

Learning subclasses of CCGs from text (i.e. strings of words) is intractable. More input data need to be provided to help the learning process. The strategy investigated so far consisted in providing indications about the analysis structure underlying a given string, under the form of a Structural Example. On the contrary, we focus here on providing additional *lexical information*. The argument of learnability used in the last proof also applies for any kind of lexical item associated with a word (i.e. a member of the vocabulary) and given as input provided that two categories assigned to a same word in a grammar always give rise to two different items. This property defines the class of CCGs learnable from non-ambiguous lexical information (in the sense of ambiguity used here). A special case of lexical information, especially for natural languages, is lexical semantics. Furthermore, the types used here are inspired by those used in Montague's typed lambda calculus to represent natural language semantics. This strategy is then both more "natural to justify" and hopefully more efficient than the one making use of Structural Examples.

## 1.5 Conclusion

The main contribution of this paper is the definition of a precise subclass of $\mathcal{G}$ which is both learnable from typed examples and has good properties from a language-theoretic point of view, as it has the same expressive power than $\mathcal{G}$ in a strong sense. So, $\mathcal{G}_{Type}$ is *representative* of $\mathcal{G}$ and grammars in this class can be considered as CCGs of a special normal form. The learnability of $\mathcal{G}_{Type}$ is guaranteed

because unambiguous types are provided, which is a strong condition. But this result can be compared for example, with Sakakibara's 1992 result which states that every context-free language can 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 in a natural language context.

## Bibliography

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

Dudau-Sofronie, D., I. Tellier, and M. Tommasi (2001). Learning categorial grammars from semantic types. In *13th Amsterdam Colloquium*, pp. 79–84.

Florêncio, C. C. (2002). Consistent identification in the limit of rigid grammars from strings is np-hard. In M. V. Z. P. Adriaans, H. Fernau, ed., *Grammatical Inference: Algorithms and Applications*, volume 2484 of *Lecture Notes in Artificial Intelligence*, pp. 49–62. Springer Verlag.

Florêncio, C. C. (2001). 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*, pp. 125–134. Springer Verlag.

Gold, E. (1967). Language identification in the limit. *Inform. Control*, **10**:447–474.

Hillel, Y. B., C. Gaifman, and E. Shamir (1960). On categorial and phrase structure grammars. *Bulletin of the Research Council of Israel*, **9F**.

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

Partee, B. (1990). *Mathematical methods in Linguistics*. Number 30 in Linguistics and Philosophy. Kluwer.

Sakakibara, Y. (1992). Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, **97**(1):23–60.

Tellier, I. (1999). Towards a semantic-based theory of language learning. In *12th Amsterdam Colloquium*, pp. 217–222.