Learning Recursive Automata from Positive Examples

Isabelle Tellier

Learning Recursive Automata from Positive Examples

Isabelle Tellier

LIFL-GRAppA & Inria Futurs (Mostrare project)
Université Charles-de-Gaulle Lille 3
BP 60149
F-59653 Villeneuve d'Ascq Cedex
isabelle.tellier@univ-lille3.fr

ABSTRACT. *In this theoretical paper, we compare the "classical" learning techniques used to infer regular grammars from positive examples with the ones used to infer categorial grammars. To this aim, we first study how to translate finite state automata into categorial grammars and back. We then show that the generalization operators employed in both domains can be compared, and that their result can always be represented by generalized automata, called "recursive automata". The relation between these generalized automata and categorial grammars is studied in detail. Finally, new learnable subclasses of categorial grammars are defined, for which learning from strings is nearly not more expensive than from structures.*

RÉSUMÉ. *Dans cet article théorique, nous proposons de comparer les techniques "classiques" employées en inférence grammaticale de langages réguliers par exemples positifs avec celles employées pour l'inférence de grammaires catégorielles. Pour cela, nous commençons par étudier les traductions réciproques entre automates finis et grammaires catégorielles. Nous montrons ensuite que les opérateurs de généralisation utilisés dans chacun des domaines sont comparables, et que le résultat de leur application peut toujours se représenter à l'aide d'automates généralisés appelés "récursifs". Les liens entre ces automates généralisés et les grammaires catégorielles sont étudiés en détail. Enfin, nous exhibons de nouvelles sous-classes apprenables de grammaires catégorielles pour lesquelles l'apprentissage à partir de textes n'est presque pas plus coûteux que l'apprentissage à partir de structures.*

KEYWORDS: *grammatical inference, positive examples, Gold's model, categorial grammars.*

MOTS-CLÉS : *inférence grammaticale, exemples positifs, modèle de Gold, grammaires catégorielles.*

RSTI - RIA – 20/2006. New Methods in Machine Learning, pages 775 to 804

1. Introduction

Grammatical inference deals with the problem of how to infer a grammar from examples it generates -and, sometimes, from examples it does not generate- (see (Cornuéjols *et al.*, 2002) for an introduction, and the proceedings of the ICGI conference, dedicated to this domain). We focus here on grammatical inference from positive examples only, and our approach is mainly theoretical. The theoretical framework is provided by Gold's model of *learnability in the limit from positive examples* (Gold, 1967) which requires precise criteria to be fulfilled.

Learning from positive examples is notoriously difficult, because it is difficult to avoid over-generalization. In Gold's model, it is now well known that no class of grammars able to generate every finite language and at least one infinite language is learnable (Gold, 1967). So, none of most classical classes of grammars (for example: those of the Chomsky hierarchy) is learnable. Nevertheless, some of their subclasses can be. As a matter of fact, subclasses of regular grammars, represented by finite state automata, have been proved learnable from strings (Angluin, 1982; Denis *et al.*, 2002).

The learnability of subclasses of context-free grammars has also been studied in this context. But a new problem arises for non-regular context-free grammars: their string language under-determines the syntactic structures they produce. So, the notion of "example generated by a grammar" is usually extended to integrate part of this syntactic structure. This is the approach adopted by Sakakibara (Sakakibara, 1990; Sakakibara, 1992) and, more recently, Kanazawa (Kanazawa, 1996; Kanazawa, 1998).

Kanazawa's main results concern the learnability from positive "structural examples" of large subclasses of AB-categorial grammars. AB-categorial grammars is a grammatical formalism, mostly known in the computational linguistic community (Oehrle *et al.*, 1988). The class of every AB-categorial grammar has the expressive power of ε-free context-free grammars (Bar Hillel *et al.*, 1960). But, surprisingly enough, nobody seems to have ever tried to represent regular languages by AB-categorial grammars, to see whether Kanazawa's learnability results coincide with other known results of regular grammatical inference from positive example. This is the starting point of this article.

So, in this paper, after having introduced the necessary definitions, we first study how finite state automata translate into AB-categorial grammars, and conversely. This is an easy application of classical language theory techniques. This translation then allows to compare learning results from positive examples known in both domains, and to deduce some new ones. But comparing learning algorithms themselves reveals to be even more interesting. In particular, we show that the generalization operator of "state fusion", used in traditional regular grammatical inference is a special case of the generalization operator of "unifying substitutions on variables" used in AB-categorial grammar learning. The latter, as we will see, is even too powerful, because it can transform an AB-categorial grammar generating a regular language into another one generating a context-free language. Yet, in this case, we show that the result can still be represented by a generalised automaton, which is a special case of Recursive Tran-

sition Network (Woods, 1970). This result leads us to study more in details the link between these generalized automata, called recursive automata, and AB-categorial grammars.

Finally, in the last part of the paper, we go back to learning considerations. We identify new learnable subclasses of AB-categorial grammars represented by recursive automata. The main interest of these new classes is that they are learnable from strings at a much smaller computational cost than in Kanazawa's approach.

This article is mainly theoretical. By linking AB-categorial grammars and finite state models, it contributes to language theory. This link is used to unify grammatical inference results coming from two different traditions, and to show that they can both benefit from ideas coming from the other one.

2. Finite State Automata and Categorial Grammars

We assume the basics of formal language theory, as stated for example in (Aho *et al.*, 1972; Hopcroft *et al.*, 1979; Gécseg *et al.*, 1996), are known by the reader. For any symbol a, we use the classical notation $a^* = \{\varepsilon, a, aa, aaa, ...\}$ where ε is the empty string, and $a^+ = aa^*$. In this section, we recall basic definitions concerning finite state automata, formal grammars, AB-categorial grammars and their connexions.

2.1. *Finite State Automata and Regular Languages*

Definition 1 (Finite State Automaton (FSA) and their Language) *A **finite state automaton** (FSA in the following) A is a 5-tuple $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ such that Q is the finite set of* states *of A, Σ its finite* vocabulary*, $q_0 \in Q$ is the* initial state *of A (in this paper, we only consider automata with a unique initial state) and $F \subseteq Q$ its set of* final states*. Finally, δ is the* transition function *of A, defined from $Q \times \Sigma$ to 2^Q.*

*The language $L(A)$ generated (or recognized) by the FSA A is defined as: $L(A) = \{w \in \Sigma^* | \delta^*(q_0, w) \cap F \neq \emptyset\}$, where δ^* is the natural extension of δ on $Q \times \Sigma^*$ such that for any $a \in \Sigma$, any $u \in \Sigma^*$ and any $q \in Q$, $\delta^*(q, au) = \{\delta^*(q', u) | q' \in \delta(q, a)\}$. The set of languages recognized by a FSA is called the set of **regular languages**.*

Example 1 *Figure 1 displays a simple FSA A such that $L(A) = a^+b^+$.*

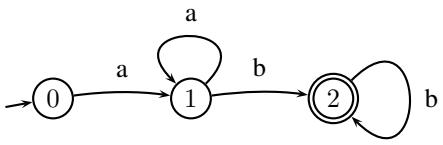

Figure 1. *a simple finite state automaton*

2.2. *Context-free Grammars and Categorial Grammars*

We now recall the classical definitions of a context-free generative grammar (useful further as an intermediary representation in proofs), and of categorial grammars. In this paper, we restrict ourselves to the most simple ones, called AB-categorial grammars (referring to their inventors, Adjukiewicz and Bar-Hillel).

Definition 2 (Context-free Grammars and their Language) *A **context-free grammar** (or simply a CF-grammar in the following) is a 4-tuple $G = \langle \Sigma, N, P, S \rangle$ with Σ the finite terminal vocabulary of G, N its finite non terminal vocabulary ($\Sigma \cap N = \emptyset$), $P \subset N \times (\Sigma \cup N)^*$ its finite set of rules and $S \in N$ its axiom.*

The language generated (or recognized) by a grammar G is $L(G) = \{w \in \Sigma^* | S \longrightarrow^* w\}$ where $\longrightarrow^*$ is the reflexive and transitive closure of the binary relation defined by P.

Definition 3 (Categories, AB-Categorial Grammars and their Language) *Let $\mathcal{B}$ be a set (at most countably infinite) of basic categories containing a distinguished category $S \in \mathcal{B}$, called the axiom. We note $Cat(\mathcal{B})$ 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 $B\backslash A \in Cat(\mathcal{B})$.*

*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 $\Sigma \times Cat(\mathcal{B})$. We note $\langle v, A \rangle \in G$ the assignment of the category $A \in Cat(\mathcal{B})$ to the element of the vocabulary $v \in \Sigma$. AB-categorial grammars (CGs in the following) are categorial grammars where the syntactic rules take the form of two rewriting schemes: $\forall A, B \in Cat(\mathcal{B})$*

- *FA (Forward Application) : $A/B \;\; B \rightarrow A$*
- *BA (Backward Application) : $B \;\; B\backslash A \rightarrow A$*

These schemes justify the "fractional" notation used to define categories. The language generated (or recognized) by a CG G is: $L(G)$=$\{w = v_1 \dots v_n \in \Sigma^+ \mid \forall i \in \{1, \dots, n\}, \exists A_i \in Cat(\mathcal{B})$ such that $\langle v_i, A_i \rangle \in G$ and $A_1 \;\dots\; A_n \rightarrow^* S\}$, where $\rightarrow^*$ is the reflexive and transitive closure of the relation $\rightarrow$, defined by FA and BA schemes. For every $w \in L(G)$, it is possible to produce a syntactic analysis structure detailing each step of the derivation of w in G. For CGs, these structures take the form of binary-branching trees whose leaf nodes are assignments of G and whose internal nodes are labeled either by FA or BA and by a category (see Example 2).

The main characteristic of CGs is that they are *lexicalized*, in the sense that the syntactic information are entirely carried by the assignments of categories to the vocabulary, and not by the rewriting rules which are general schemes defined once for all. They are mainly known in the domain of natural language processing, because they allow natural correspondences with logical semantics (Oehrle *et al.*, 1988).

Example 2 *CGs have mainly been used to represent natural language syntax, so this is the case in this example. For example, let* $\mathcal{B} = \{S, T, CN\}$ *(where* T *stands for "term" and* CN *for "common noun"),* $\Sigma = \{John, runs, loves, a, cat, man\}$ *and* G *be defined by:*
$G = \{\langle John, T\rangle, \langle runs, T\backslash S\rangle, \langle loves, (T\backslash S)/T\rangle, \langle loves, T\backslash(S/T)\rangle, \langle cat, CN\rangle, \langle man, CN\rangle, \langle a, (S/(T\backslash S))/CN\rangle, \langle a, ((S/T)\backslash S)/CN\rangle\}$.
This CG allows to recognize sentences like "John runs" or "John loves a cat" with the syntactic analysis structures of Figure 2.

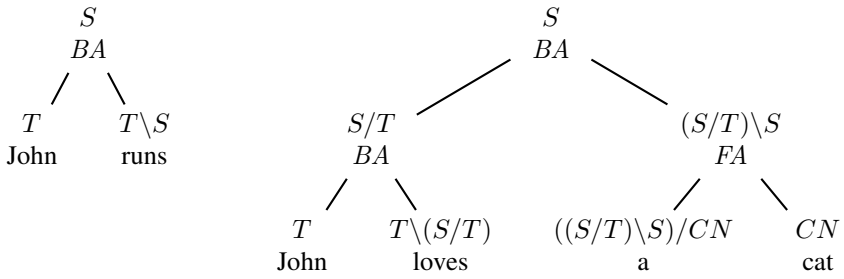

Figure 2. *syntactic analysis structures produced by an AB-categorial grammar*

Definition 4 (F-Structures, Structural Examples, Structured Language) *A* ***functor-argument (or F-) 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 F-structures over* Σ *is denoted* $F(\Sigma)$.
For any AB-categorial grammar $G \subset \Sigma \times Cat(\mathcal{B})$*, a structural example for* G *is an element of* $F(\Sigma)$ *which can be obtained from a syntactic analysis structure associated with a string* $w \in L(G)$ *by deleting each of the categories it contains. For any CG* $G \subset \Sigma \times Cat(\mathcal{B})$*, the structured language* $FL(G)$ *associated with* G *is the set of its structural examples (see Example 4).*

$\mathcal{G}$ denotes the class of every CGs. For every integer $k \geq 1$, the set of CGs assigning at most k distinct categories to each member of its vocabulary is the class of k-valued CGs denoted by $\mathcal{G}_k$. When $k = 1$ the grammars are also called *rigid*.

2.3. *From Automata to Regular Categorial Grammars*

The expressive power of CGs is the one of ε-free context-free grammars (Bar Hillel *et al.*, 1960). So, they can also generate every ε-free regular languages. Correspondences between FSA and CGs recognizing the same languages are easy to define.

Definition 5 (Regular CGs) *We call a* FA*-regular (resp.* BA*-regular) CG a CG* $G_{FA} \subset \Sigma \times Cat(\mathcal{B})$ *(resp.* $G_{BA} \subset \Sigma \times Cat(\mathcal{B})$*) that only contains assignments of*

the form $\langle a, A\rangle$ or $\langle a, A/B\rangle$ (resp. of the form $\langle a, A\rangle$ or $\langle a, B\backslash A\rangle$) with $a \in \Sigma$ and $A, B \in \mathcal{B}$ (so, A and B are basic categories*). We note $\mathcal{G}^r_{FA}$ (resp. $\mathcal{G}^r_{BA}$) the class of every FA-regular (resp. BA-regular) CG.*

Let $A = \langle Q, \Sigma, \delta, q_0, F\rangle$ be any FSA. Let $\mathcal{B} = (Q\backslash\{q_0\}) \cup \{S\}$ (where $S \notin Q$). There exist a FA-regular CG $G_{FA} \in \mathcal{G}^r_{FA}$ and a BA-regular CG $G_{BA} \in \mathcal{G}^r_{BA}$, both finite subsets of $\Sigma \times Cat(\mathcal{B})$ and both recognizing the language $L(A)\backslash\varepsilon$. To obtain these CGs, you just have to apply the classical transformations of a FSA into a right-linear regular grammar and a left-linear regular grammar and then to transform the former into a FA-regular CG and the latter into a BA-regular CG, in an obvious way.

Example 3 *Let us transform the FSA of Example 1 into a FA-regular (resp. BA-regular) CG. The rules of the right-linear regular grammar G_1 recognizing the same language as A are the following (where the state of number i is associated with a non terminal symbol noted q_i, with $q_0 = S$): $S \longrightarrow aq_1$, $q_1 \longrightarrow aq_1$, $q_1 \longrightarrow b$, $q_1 \longrightarrow bq_2$, $q_2 \longrightarrow b$, $q_2 \longrightarrow bq_2$. The corresponding FA-regular CG G_{FA} is:*
$G_{FA} = \{\langle a, S/q_1\rangle, \langle a, q_1/q_1\rangle, \langle b, q_1\rangle, \langle b, q_1/q_2\rangle, \langle b, q_2\rangle, \langle b, q_2/q_2\rangle\}$.
The rules of the left-linear regular grammar G_2 recognizing the same language as A are the following (where the state of number i is associated with a non terminal symbol noted q_i): $q_1 \longrightarrow a$, $q_1 \longrightarrow q_1a$, $q_2 \longrightarrow q_1b$, $q_2 \longrightarrow q_2b$, $S \longrightarrow q_1b$, $S \longrightarrow q_2b$. The corresponding BA-regular CG G_{BA} is:
$G_{BA} = \{\langle a, q_1\rangle, \langle a, q_1\backslash q_1\rangle, \langle b, q_1\backslash q_2\rangle, \langle b, q_2\backslash q_2\rangle, \langle b, q_1\backslash S\rangle, \langle b, q_2\backslash S\rangle\}$

So, FSA can easily be *lexicalized*. In fact, *the previous transformations preserve not only the string language, but also the structural descriptions associated by the intermediate grammars with the strings*. So, G_{FA} generates right-branching flat trees using only the FA scheme of rule (as displayed in Figure 3), whereas G_{BA} generates left-branching flat trees using only the BA scheme. A crucial consequence is that, in both subclasses of CGs, structural examples in the sense of Definition 4 are available for free, i.e. in linear time, from the corresponding strings.

Example 4 *Figure 3 displays (on the left) two syntactic structures produced by the CG G_{FA} obtained in Example 3, and (on the right) the corresponding structural examples.*

2.4. *From Regular Categorial Grammars to Finite State Automata*

The way back, from regular CGs to FSA, is not more difficult. Let $G_{FA} \in \mathcal{G}^r_{FA}$ (resp. $G_{BA} \in \mathcal{G}^r_{BA}$) be a FA-regular (resp. BA-regular) CG. To transform it into a FSA A_{FA} (resp. A_{BA}) recognizing the same (ε-free) language, it is enough to proceed as follows: if $\mathcal{B}$ is the set of basic categories of G_{FA} (resp. of G_{BA}), then the set of states of A_{FA} (resp. A_{BA}) contains B and an additional final (resp. initial) state. Each lexical assignment in the initial CG corresponds to a transition in the FSA:

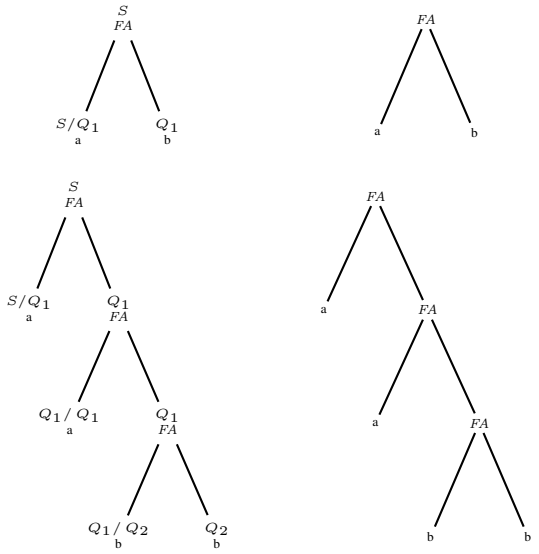

Figure 3. *syntactic structures and the corresponding structural examples*

1) Starting from G_{FA}, let the initial state of A_{FA} be $q^0_{FA} = S$. Let $Q_{FA} = \mathcal{B} \cup \{F\}$ with $F \notin \mathcal{B}$ and $F_{FA} = \{F\}$. Each assignment $\langle a, U/V\rangle \in G_{FA}$ corresponds to a transition labeled by a between the states U and V in A (i.e. $\delta_{FA}(U, a) = V$) and each assignment $\langle a, U\rangle \in G_{FA}$ to a transition labeled by a between U and F_A (i.e. $\delta_{FA}(U, a) = F_A$).

2) Starting from G_{BA}, let $Q_{BA} = \mathcal{B} \cup \{I\}$ with $I \notin \mathcal{B}$. Let $q^0_{BA} = \{I\}$ and $F_{BA} = \{S\}$. Each assignment $\langle a, U\backslash V\rangle \in G_{BA}$ corresponds to a transition labeled by a between the states U and V in A (i.e. $\delta_{BA}(U, a) = V$) and each assignment $\langle a, U\rangle \in G_{BA}$ to a transition labeled by a between I_A and U (i.e. $\delta_{BA}(I_A, a) = U$).

Note that these transformations do not mean that only the CGs that are FA or BA-regular generate a regular language. The CG defined in Example 2 is neither FA-regular nor BA-regular in the sense of Definition 5 but it generates a finite (and thus regular) language. But the structures it produces are not flat.

Example 5 *The FSA built by applying these operations to the CGs obtained in Example 3 are given in Figure 4. They are not exactly the same as the initial one (in Figure 1), because of the state (F in the first one, I in the second one) added to the ones coming from basic categories. They are non deterministic.*

These transformations have an easy but interesting consequence, which will be useful further: the notion of *language associated with a state* can now receive two different characterizations.

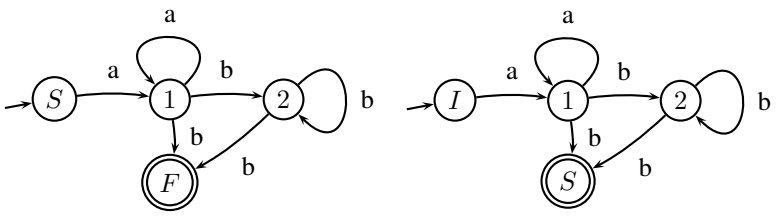

Figure 4. *FSA obtained from a FA-regular CG (left) and a BA-regular CG (right)*

Definition 6 (State Languages) *In a CG $G \subset \Sigma \times Cat(\mathcal{B})$, the language $L(C)$ associated with a category $C \in Cat(\mathcal{B})$ is the set of strings which can be assigned categories that can be reduced to C (in other words, they are sub-constituants of category C): $L(C) = \{w = u_1 \dots u_n \in \Sigma^+ \mid \forall i \in \{1, \dots, n\}\ \exists A_i \in Cat(\mathcal{B})$ such that $\langle u_i, A_i \rangle \in G$ and $A_1\ \dots\ A_n \rightarrow^* C\}$. Now, if this CG is FA-regular (resp. BA-regular), it can be transformed into a FSA A_{FA} (resp. A_{BA}) as previously explained, where C becomes a non-final state (resp. a non initial state). In both cases, $L(C)$ can receive a new characterization:*

1) in A_{FA}: $L(C) = L_{FA}(C) = \{w \in \Sigma^+ | F \in \delta^+_{FA}(C, w)\}$ where F is the unique final state of A_{FA} and δ_{FA} is its transition function. This definition excludes ε (we know that $C \neq F$ so $\varepsilon \notin L_{FA}(C)$) and uses the fact that there is only one terminal state F in A_{FA}. This means that strings in $L_{FA}(C)$ correspond to paths starting from the state C in A_{FA} ($C \neq F$), and reaching the final state F. It is sometimes also called the suffix language of the state C. *Of course, if $C = S$, we have: $L_{FA}(S) = L(A_{FA}) = L(G)$.*

2) in G_{BA}: $L(C) = L_{BA}(C) = \{w \in \Sigma^+ | C \in \delta^+_{BA}(I, w)\}$ where I is the unique initial state of A_{BA} and δ_{BA} is its transition function. So, this time, strings in $L_{BA}(C)$ correspond to paths starting from the initial state I ($I \neq C$) in A_{BA} and reaching the state C. It is sometimes also called the prefix language of the state C. *Of course, if $C = S$, we also have: $L_{BA}(S) = L(A_{BA}) = L(G)$.*

Example 6 *In the left automaton of Figure 4, $L_{FA}(q_1) = a^*b^+$ and $L_{FA}(q_2) = b^+$; in the right automaton $L_{BA}(q_1) = a^+$ and $L_{BA}(q_2) = a^+b^+$;*

3. Inference of Categorial Grammars from Positive Examples

The learnability of CGs in Gold's model from positive examples has received great attention in the recent years. Now that we have an easy translation of subclasses of such grammars (the subclasses $\mathcal{G}^r_{FA}$ and $\mathcal{G}^r_{BA}$) into automata, it is natural to see how these results translate into regular language learning, to compare them with known results in this domain. In this section, we focus on FA-regular CGs, but the same could be done with BA-regular CGs.

3.1. *Gold's model*

The notion of learnability in Gold's model from positive examples (Gold, 1967) is also known as *learnability in the limit from positive examples*. We recall its definition and Kanazawa's main results here.

Definition 7 (Gold's Learnability Criterion) *Let $\mathcal{G}$ be a set of grammars (in the following, $\mathcal{G}$ will always be a subclass of CGs) over an alphabet Σ and let a function associate a language to each grammar. This function will either be the string language generated by the grammar $L : \mathcal{G} \longrightarrow pow(\Sigma^*)$ (cf. Definition 3) or its structured language $FL : \mathcal{G} \longrightarrow pow(F(\Sigma))$ (cf. Definition 4), where for any set Δ, $pow(\Delta)$ refers to the set of subsets of Δ.*

*ϕ is a mapping which associates to some finite samples of strings in Σ^* (resp. of structural examples in $F(\Sigma)$) a grammar in $\mathcal{G}$. ϕ is said to **converge** to $G \in \mathcal{G}$ on a sample $\langle s_i \rangle_{i \in \mathbb{N}}$ of elements of Σ^* (resp. of $F(\Sigma)$) if $\exists n_0 \in \mathbb{N}$ such that $\forall i \geq n_0$, $G_i = \phi(\langle s_0, ..., s_i \rangle)$ is defined and equal to G.*

*ϕ is said to **learn** the class $\mathcal{G}$ from positive examples (resp. from positive structural examples) if for every language $\mathcal{L}$ in $L(\mathcal{G}) = \{L(G) | G \in \mathcal{G}\}$ (resp. in $FL(\mathcal{G}) = \{FL(G) | G \in \mathcal{G}\}$) and every sequence $\langle s_i \rangle_{i \in \mathbb{N}}$ which enumerates $\mathcal{L}$, i.e. for which $\{s_i | i \in \mathbb{N}\} = \mathcal{L}$, there exists $G \in \mathcal{G}$ such that $\mathcal{L} = L(G)$ (resp. $\mathcal{L} = FL(G)$) and ϕ converges to G on $\langle s_i \rangle_{i \in \mathbb{N}}$.*

*$\mathcal{G}$ is **learnable** if there exists a computable function ϕ which learns $\mathcal{G}$.*

This well known criterion of learnability applies to *classes of grammars*. A mapping ϕ learns a class if it is able to identify *any member of this class* when a sample enumerating the corresponding language is given as input. Children, too, are able to learn *any natural language*, provided that they are raised in a environment where this language is spoken. Note also that ϕ is not required to converge on the exact grammar which generated the sample (the target grammar), but on any grammar *generating the same language*. The notion of language which serves as a success criterion is, of course, the same as the one which was used to generate the sample. For example, if structural examples are provided to the learning function, the grammar on which it converges must generate the same structured language as the target grammar.

It is known since (Gold, 1967) that no class able to produce every finite language and at least one infinite language is learnable. So, the class of every CG $\mathcal{G}$ is not learnable. Nevertheless, Kanazawa (Kanazawa, 1998) has proved that interesting subclasses of $\mathcal{G}$ are learnable.

Theorem 1 (Learnability of $\mathcal{G}_k$ (Kanazawa, 1998)) *For any $k \geq 1$, the class $\mathcal{G}_k$ of k-valued CGs is learnable from strings and from structural examples.*

And, of course, $\mathcal{G} = \cup_{k \in \mathbb{N}} \mathcal{G}_k$. Nevertheless, a learnability result does not necessary lead to a tractable learning algorithm ϕ. As a matter of fact, it has been proved that

the problem of identifying an element of $\mathcal{G}_1$ from strings is NP-hard (Florêncio, 2002), and identifying an element of $\mathcal{G}_k$ with $k > 1$ from structural examples is also NP-hard (Florêncio, 2001).

3.2. *Categorial Grammars and Regular Grammatical Inference*

In this section, we first translate Kanazawa's results into regular grammatical inference. We then show the equivalence between two other known results.

Theorem 2 (Learnability of k-valued FA-Regular CGs) *For every $k \geq 1$, the class of k-valued FA-regular CGs $\mathcal{G}_k \cap \mathcal{G}^r_{FA}$ is learnable from structural examples and from strings.*

Proof 1 (Proof of Theorem 2) *This theorem is a direct consequence of Theorem 1, restricted to the class $\mathcal{G}_k \cap \mathcal{G}^r_{FA}$. To prove it directly, it is enough to notice that for any given integer $k \geq 1$, $\mathcal{G}_k \cap \mathcal{G}^r_{FA}$ contains a finite number of distinct classes of isomorph CGs (up to a renaming of their basic categories). We know from (Angluin, 1980) that this property implies learnability.* □

Note also that, according to section 2.4, elements of $\mathcal{G}_k \cap \mathcal{G}^r_{FA}$ can be transformed into FSA containing at most k distinct transitions *labeled by the same element of* Σ. So, Theorem 2 can also be seen as a generalization of a result stated in (Yokomori, 1995). The class of "Strictly Deterministic Automata" defined in this paper coincides with the class of automata which translate CGs belonging to $\mathcal{G}_1 \cap \mathcal{G}^r_{FA}$. Similarly, we can introduce the notion of k-valued automata, i.e. FSA which are the result of applying the process of the section 2.4 to k-valued FA-regular CGs. This class seems more adapted to represent languages with large alphabets Σ than many usual classes. And, of course, $\cup_{k\geq 1}\{L(G)|G \in \mathcal{G}_k \cap \mathcal{G}^r_{FA}\}$ contains every regular language.

Furthermore, the computable function proposed by Kanazawa to learn $\mathcal{G}_k$ from structures (and which is given in section 3.3) can easily be adapted to learn $\mathcal{G}_k \cap \mathcal{G}^r_{FA}$ from strings. To do this, first associate a right-branching flat structure with FA internal nodes to every string of the sample and apply the learning function to this input. The result provided by this function is conserved only if the grammar belongs to $\mathcal{G}_k \cap \mathcal{G}^r_{FA}$ (which is decidable). It is interesting to notice that, in the case of FA-regular CGs, unlike in the case of general CGs, strings and structures are equivalent - that is, structures are available for free from strings. This is the idea we will try to generalize for larger classes of CGs in the last section.

Other interesting results worth being compared: the one concerning the class of 0-reversible FSA (Angluin, 1982) and the one concerning the class of reversible CGs (Besombes *et al.*, 2004).

Definition 8 (0-Reversibility of a FSA (Angluin, 1982)) *A FSA is said to be 0-reversible if and only if it is deterministic and the automaton obtained by reversing the transitions backwards is also deterministic.*

Definition 9 (Reversibility of a CG (Besombes *et al.*, 2004)) *A CG is said to be reversible if it does not contain two assignments of categories for the same element of vocabulary, which are distinct by only one basic category.*

The class of 0-reversible FSA is learnable in the limit from positive examples (Angluin, 1982). The class of reversible GCs is learnable in the limit from structural examples and from strings (Besombes *et al.*, 2004). We show in the following the connections between these two results.

Theorem 3 (Equivalence of these Reversibility Notions) *Let G be a FA-regular CG and let A be the FSA obtained from it, as described in section 2.4. A is 0-reversible in the sense of Definition 8 if and only if G is reversible in the sense of Definition 9.*

Proof 2 (Sketch of Proof of Theorem 3) *This theorem is a direct consequence of the way A is built, with a unique initial state, a unique final state and no ε transition. So, the conditions for A to be deterministic only concern transitions starting from the same state and labeled by the same symbol. They are equivalent for G with the following ones: $\forall a \in \Sigma$*

– $\forall Q_1, Q_2, Q_3 \in \mathcal{B}$: $\langle a, Q_1/Q_2\rangle \in G$ and $\langle a, Q_1/Q_3\rangle \in G \Leftrightarrow Q_2 = Q_3$.

– $\forall Q_1, Q_2 \in \mathcal{B}$: $\langle a, Q_1\rangle \in G$ and $\langle a, Q_1/Q_2\rangle \in G \Leftrightarrow Q_2 = F_A$ (in fact, $\langle a, Q_1\rangle$ stands for $\langle a, Q_1/F_A\rangle$ and no transition starts from F_{FA});

Similarly, as A has only one final state, the conditions for the reversed automaton to be deterministic are equivalent with the following ones: $\forall a \in \Sigma$

– $\forall Q_1, Q_2 \in \mathcal{B}$: $\langle a, Q_1\rangle \in G$ and $\langle a, Q_2\rangle \in G \Leftrightarrow Q_1 = Q_2$

– $\forall Q_1, Q_2, Q_3 \in \mathcal{B}$: $\langle a, Q_1/Q_2\rangle \in G$ and $\langle a, Q_3/Q_2\rangle \in G \Leftrightarrow Q_1 = Q_3$.

For FA-regular CGs, these conditions coincide with the one of Definition 9.□

So, the learnability of the class of 0-reversible FSA from strings (Angluin, 1982) can be seen as a consequence of the learnability of the class of FA-regular reversible CGs from structures, which itself is a consequence of (Besombes *et al.*, 2004)'s result. In fact, this is not very surprising, because (Besombes *et al.*, 2004) were inspired by a result on the learnability of reversible context-free grammars from structures (Sakakibara, 1992), which was already a generalization of (Angluin, 1982)'s result. The corresponding learning algorithms should be more carefully compared, but they also seem to be equivalent. Note that the class of reversible CGs, which includes the class of rigid CGs $\mathcal{G}_1$, is learnable from structural examples in polynomial time.

3.3. *Learning Algorithm*

The learnability results of Theorem 1 are not only theoretical ones. From a given sample of structural examples, it is possible to characterize the set of CGs of a certain normal form compatible with this sample, that is the set of CGs G whose structured language $FL(G)$ contains these structural examples. The original algorithm able to identify the set of k-valued CGs in reduced form (cf. Definition 15 in section 5.1) compatible with a sample of structural examples is due to Buszkowski & Penn (Buszkowski *et al.*, 1990). When the target is a *rigid grammar*, the algorithm is usually called RG. In the more general case, when the target grammar is simply known to belong to $\mathcal{G}_k$ for some known $k \in \mathbb{N}$, we will call it BP_k. We recall it here, examplifying it on a simple sample of flat structural examples, to see how it relates to traditional learning strategies employed in regular grammatical inference.

Let D be a sample of structural examples for an unknown CG. The first steps of $BP_k(D)$ are the following: for each element of D

1) introduce a label S at the root;

2) introduce a distinct variable $x_i \in \chi$ at each argument node (i.e. at the left daughter of each BA node and at the right daughter of each FA node);

3) introduce a fractional category at every other node, allowing the functional application identified by the label FA or BA of the father node to apply.

The result of collecting all the categories associated with each distinct member of the vocabulary (at the leaves of the trees) after these steps is a CG called the **general form** of D and noted $GF(D)$. The structured language of this grammar is exactly the initial sample D, i.e.: $FL(GF(D)) = D$.

Example 7 *Let D be the pair of structural examples of Example 4. The application of steps 1) to 3) to it gives the result of Figure 5, and $GF(D)$ is defined as follows:*

– *a:* $S/x_1,\ S/x_4,\ x_4/x_3$*;*

– *b:* $x_1,\ x_3/x_2,\ x_2$*.*

The corresponding FSA is given in Figure 6: it very much looks like what is known in regular grammatical inference as the ***maximal canonical automaton*** $MCA(I)$ *with respect to a set of strings $I \subset \Sigma^*$ (Dupont* et al.*, 1994; Cornuéjols* et al.*, 2002). By definition, $MCA(I)$ has a unique initial state Q_0 and for each $w = u_1...u_n \in I$ it has a sequence $Q_1, ..., Q_n$ of distinct states with $Q_n \in F$, and $\forall i \in \{1, ..., n\}$: $\delta(Q_{i-1}, u_i) = Q_i$. The only difference between our automaton and the MCA built with respect to the strings associated with the elements of D is that it has a unique final state. In regular grammatical inference from a set of positives examples I, the starting point is usually not $MCA(I)$ but $PTA(I)$: the* Prefix Tree Automaton*, which is obtained from $MCA(I)$ by merging the states sharing the same prefixes. Both automata exactly generate the language I but $PTA(I)$, contrarily to $MCA(I)$, is always deterministic. But the notion of determinism has no relevance in the context of CGs, so there is no reason to prefer $PTA(I)$ to our FSA here.*

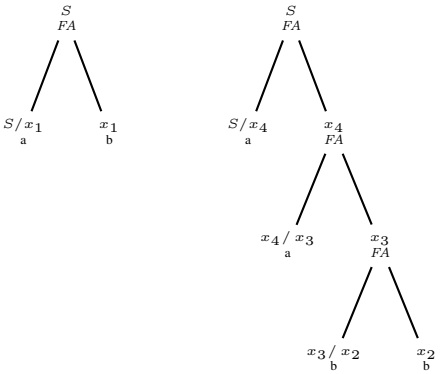

Figure 5. *result of applying the first steps of the learning algorithm*

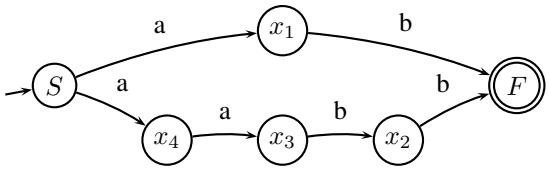

Figure 6. *the automaton corresponding to* $GF(D)$

The next step of $BP_k(D)$ consists in searching for substitutions that unify some category assignments of the general form $GF(D)$. Thus, we first need to introduce the notion of substitution.

Definition 10 (Variable Categories and Substitutions) *Let* χ *be an enumerably infinite set of variables and* $\mathcal{B} = \chi \cup \{S\}$. *A substitution is a function* $\sigma : \chi \longrightarrow Cat(\mathcal{B})$ *that maps variables to categories. It can be extended to a function from categories to categories as follows: (i)* $\sigma(S) = S$, *(ii)* $\sigma(A/B) = \sigma(A)/\sigma(B)$ *and (iii)* $\sigma(A\backslash B) = \sigma(A)\backslash\sigma(B)$ *for any* $A, B \in Cat(\mathcal{B})$. *Similarly, a substitution is naturally extended to apply to a CG:* $\forall G \in \mathcal{G}$, $\sigma(G) = \{\langle v, \sigma(A)\rangle | \langle v, A\rangle \in G\}$. *For any CG* G, *a* unifying substitution *for* G *is a substitution that unifies (i.e. gives the same result to) distinct categories assigned to the same element of the vocabulary of* G. *More formally,* σ *is a unifying substitution for* G *if* $\forall v \in \Sigma$ *we have:* $|\{\sigma(A) | \langle v, A\rangle \in G\}| \leq |\{A | \langle v, A\rangle \in G\}|$. *As a consequence, a unifying substitution can only* decrease *the maximal number of distinct categories assigned to the same member of the vocabulary (the value of the* k *such that the grammar is* k*-valued).*

Property 1 (Fundamental Property (Buszkowski *et al.*, 1990)) *For any CG G, the following two properties are equivalent: (i) $D \subseteq FL(G)$ and (ii) $\exists\sigma$ such that $\sigma(GF(D)) \subseteq G$.*

In other words, CGs G which are compatible with D, that is for which $D \subseteq FL(G)$, are those which contain a substitution of $GF(D)$. But, to avoid searching for every possible substitution, Kanazawa proves that it is enough to search for *unifying substitutions* only. In this case, the grammars obtained are in a certain normal form (cf. Definition 15 and Theorem 6 further). If the target grammar is k-valued, the next step of the computation of $BP_k(D)$ is the following:

4) find every possible unifying substitution σ for $GF(D)$ such that the grammar $\sigma(GF(D))$ is k-valued.

The result of the computation of $BP_k(D)$ is thus *a set of grammars of the form* $\sigma(GF(D))$. If $k = 1$, the set is at most a singleton because, if it is possible, there exists a unique way to unify every distinct category assigned to every distinct word. In this case, the 4 steps detailed above build a learning algorithm of $\mathcal{G}_1$ in the sense of Gold and can be efficiently implemented; this algorithm is quadratic in the size of the input D and is incremental (Kanazawa, 1998).

If $k > 1$, the situation may be more difficult because the output set $BP_k(D)$ of k-valued grammars may never be reduced to one grammar in the limit. Yet, Gold's criterion imposes to select only one grammar among this set. As only positive examples are available, there is a risk to select a grammar which overgeneralizes. To avoid this, Kanazawa proposes to perform inclusion tests on the structured languages generated by every grammar in the set (the inclusion of structured languages is decidable). Let μ be a computable function that maps any non-empty finite set of CGs to one of its minimal member with respect to the inclusion of its structured language. Then it is possible to define a learning function ϕ for $\mathcal{G}_k$ as follow: $\forall D = \langle T_i \rangle_{0 \leq i \leq n}$:

– $\phi(\langle T_0 \rangle) = \mu(BP_k(T_0))$
– $\phi(\langle T_0, ..., T_{i+1})$

$$= \begin{cases} \phi(\langle T_0, ..., T_i \rangle) & \text{if } T_{i+1} \in FL(\phi(\langle T_0, ..., T_i \rangle)), \\ \mu(BP_k(\{T_0, ..., T_{i+1}\}) & \text{otherwise} \end{cases}$$

This strategy is not tractable in practice, because of its high algorithmic cost, but it provides a learning algorithm in the sense of Gold.

Applying a substitution on a CG is a generalization operation, because we have the following property (Buszkowski *et al.*, 1990): $\sigma(G_1) \subseteq G_2 \implies FL(G_1) \subseteq FL(G_2)$. So we always have: $FL(G) \subseteq FL(\sigma(G))$ and, similarly, $L(G) \subseteq L(\sigma(G))$. As already stated, it has been proved that it is enough to search for *unifying substitutions*. What does it mean to unify two categories assigned to the same member of the vocabulary ? A good way to interpret this operation is to see its effect on the corresponding FSA. As we have seen in section 2.4, each assignment $\langle a, U \rangle$ of a category $U \in Cat(\mathcal{B})$ to a member of the vocabulary $a \in \Sigma$ in a FA-regular CG gives

a transition labeled by a in the corresponding FSA. *Unifying two categories assigned to the same $a \in \Sigma$ in a regular CG thus seems to mean "merging two transitions" labeled by a in the corresponding FSA*.

In usual regular grammatical inference, the most often used generalization operator is not this one: it is the operator of *state merging* (Angluin, 1982; Oncina *et al.*, 1992; Dupont *et al.*, 1994; Cornuéjols *et al.*, 2002). What is the link between "category unification" and "state merging" ? Is one of these operators more powerful than the other ? Let us look at this more carrefully. In the context of a set D containing only flat structures with FA internal nodes, $GF(D)$ is necessarily a FA-regular CG (see Example 7). Taking into account the general form of FA-regular CGs (as specified in Definition 5), only two cases can occur to unify two categories:

– conditions of the form $\sigma(x_i) = \sigma(x_j) = x_j$ for any $x_i \in \chi$ and any $x_j \in \chi \cup \{S\}$ exactly specify a state merging becauses the variables introduced coincide with states of the corresponding FSA: so, states x_i and x_j are merged.

– conditions of the form $\sigma(x_i) = x_j/x_k$, for any $x_i \in \chi$ and for any $x_j, x_k \in \chi \cup \{S\}$ are more difficult to understand. They seem to mean that *a state is merged with a transition*! The best way to understand them is to remember that variables correspond to basic categories, and can be used to label not only symbols of Σ but also *strings in* Σ^+. By unifying x_i and x_j/x_k, we thus allow every string that could be associated with the category x_i in the CG to be used as a transition between the states x_j and x_k in the FSA. We will see better how to interpret this in Example 8.

Example 8 *Let us illustrate both cases on our canonical example, with $k = 2$. Let us define step by step a unifying substitution σ for the $GF(D)$ of Example 7, such that $\sigma(GF(D))$ is 2-valued. To unify two categories assigned to $a \in \Sigma$, one solution is to set: $\sigma_1(x_4) = \sigma_1(x_1) = x_1$ (by convention, when variables are unified, the result is the variable with the smallest index), and σ_1 is the identity elsewhere. The resulting CG $\sigma_1(GF(D))$ is then the following:*

– *a:* $S/x_1,\ x_1/x_3$;

– *b:* $x_1, x_3/x_2,\ x_2$.

The corresponding FSA is displayed in Figure 7. In this case, the substitution σ_1

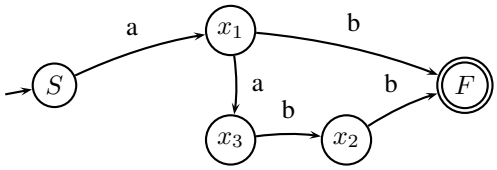

Figure 7. *the automaton corresponding to $\sigma_1(GF(D))$*

specifies to merge the states x_4 and x_1 in the FSA representing $GF(D)$. Here, state merging appears as a special case of unifying categories, and it does not allow to

extend the language recognized by the grammar (or by the FSA). But it is not always so simple. To unify categories assigned to $b \in \Sigma$*, let us now set:* $\sigma_2(x_1) = x_3/x_2$ *(and* σ_2 *is the identity elsewhere). The resulting CG* $\sigma_1 \circ \sigma_2(GF(D))$ *is the following:*

– *a:* $S/(x_3/x_2),\ (x_3/x_2)/x_3$*;*

– *b:* $x_3/x_2,\ x_2$*.*

We note that this CG is no longer FA*-regular. Nevertheless, we propose to represent it in a generalized automaton, as shown in Figure 8.*

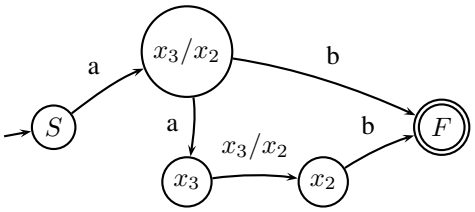

Figure 8. *the generalized automaton corresponding to* $\sigma_1 \circ \sigma_2(GF(D))$

In this automaton, the previous state x_1 *was renamed according to* σ_2 *as* x_3/x_2*. And, to express that every string that could previously be associated with the category* x_1 *can now be used as a transition between the states* x_3 *and* x_2 *a new transition labeled by* x_3/x_2 *replaces the one that was labeled by* b *between these states. To use this new transition, you need to produce a string of category* x_3/x_2*, that is a string belonging to the state language of* x_3/x_2*. According to Property 6, such a string is obtained by following a path starting from the state* x_3/x_2 *and reaching the final state* F*. The letter* b *labels such a direct path. But another possible choice is to first reach* x_3 *by* a *where the previous situation is posed again, and then to reach* F *by* b*. A stack is necessary to remember each time the transition labeled by* x_3/x_2 *is used. So, we call it a "recursive transition".*

This generalized automaton can be considered as a special case of Recursive Transition Network (Woods, 1970). The language it recognizes is the same as the one recognized by $\sigma_1 \circ \sigma_2(GF(D))$ *and is equal to* $a^n b^n$*, for* $n \geq 1$*. This language is not regular but is a correct generalization of the input* $\{ab,\ aabb\}$*.*

How could such a generalization occur ? Let us look at the analysis tree produced by $\sigma_1 \circ \sigma_2(GF(D))$ *for the string* $aaabbb$*, given in Figure 9.*

This tree is no longer flat. To understand how it was built, look back at the second tree of Figure 5. By combining substitutions σ_1 *and* σ_2*, we have specified the following condition:* $\sigma_1 \circ \sigma_2(x_4) = \sigma_1 \circ \sigma_2(x_1) = x_3/x_2$*. In the tree, this condition provides an equality between the label of an internal node (*x_4*) and the label of a leaf (*x_3/x_2*). This equality thus opens the possibility to copy the subtree rooted by* x_4 *in the place of the leaf labeled by* x_3/x_2*, as is done in Figure 9. This operation can be seen as what is called a* substitution*, explicitly used in the formalism of Tree Adjoining Grammars, for example (Joshi* et al.*, 1997).*

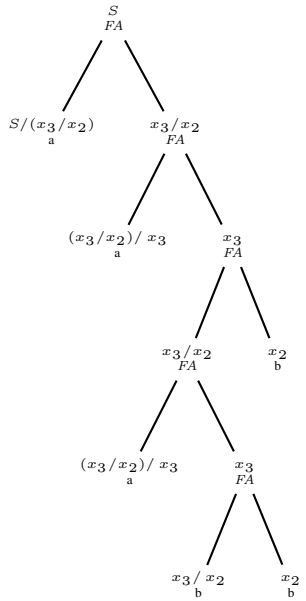

Figure 9. *syntactic analyses tree for* $aaabbb$

Example 8 shows that variable unification is more powerful than state merging, because it can transform a FSA into a generalized automaton, generating a language which outpasses the class of regular languages. To extend the connection between FA-regular (resp. BA-regular) CGs and FSA, we need to take into account the new notion of automata this example introduces. The next section is dedicated to the study of this connection, before going back to learning considerations.

4. Recursive Automata and Categorial Grammars

Section 2 showed that FSA and CGs go together well. But, of course, FSA are not powerful enough to represent every CG. In this section, we first study the connection, not only in terms of strings but also in terms of *structures*, between unidirectional CGs (that is, CGs making use of only one operator among $\{/, \backslash\}$) and our new class of generalized automata. This new class can be considered as a special case of recursive transition networks (Woods, 1970), but where the recursive transitions refer to paths inside the same automaton instead of refering to paths into another automaton (as is usually the case in RTNs). For this reason, we call the members of this class *recursive automata*. We then show that to produce the *structures* generated by any (not necessarily unidirectionnal) CG, it is always enough to consider *a pair of mutually recursive automata*.

4.1. *Recursive Automata and their Language*

Definition 11 (Recursive Automaton) *A **recursive automaton** R is a 5-tuple $R = \langle Q, \Sigma, \gamma, q_0, F\rangle$ such that Q is the finite set of* states *of R, Σ is its finite* vocabulary*, $q_0 \in Q$ its (unique)* initial state *and $F \in Q$ its (unique)* final state*. γ is the* transition function *of R, defined from $Q \times (\Sigma \cup Q)$ to 2^Q.*

The only important difference between this definition and Definition 1 is that in a recursive automaton (RA in the following), it is possible to *label a transition either by an element of Σ or by an element of Q*. We restrict ourselves to RA with a unique final state, but it is not a crucial choice. As we have seen in Property 6, there exist two different notions of state language. Similarly, there exist two different notions of RA which will be called, for obvious reasons, RA_{FA} and RA_{BA}. They differ in the way their language is defined.

Definition 12 (Language Recognized by a RA) *Let $R = \langle Q, \Sigma, \gamma, q_0, F\rangle$ be a RA_{FA} (resp. a RA_{BA}). For every $q \in Q$ (resp. every $q' \in Q$) we define the language $L_{FA}(q)$ (resp. $L_{BA}(q')$) associated with q (resp. with q') as the smallest set satisfying:*

– $L_{FA}(F) = \epsilon$ (resp. $L_{BA}(q_0) = \epsilon$);

– if there exists a transition labeled by $a \in \Sigma$ between q and q', i.e. $q' \in \gamma(q, a)$ then: $a.L_{FA}(q') \subseteq L_{FA}(q)$ (resp. $L_{BA}(q).a \subseteq L_{BA}(q')$);

– if there exists a transition labeled by $r \in Q$ between q and q', i.e. $q' \in \gamma(q, r)$ then: $L_{FA}(r).L_{FA}(q') \subseteq L_{FA}(q)$ (resp. $L_{BA}(q).L_{BA}(r) \subseteq L_{BA}(q')$).

The language $L_{FA}(R)$ of the RA_{FA} (resp. the language $L_{BA}(R)$ of the RA_{BA}) is defined by: $L_{FA}(R) = L_{FA}(q_0)$ (resp. $L_{BA}(R) = L_{BA}(F)$).

For a state $q \in Q$ (resp. $q' \in Q$) such that $q \neq F$ (resp. $q' \neq q_0$), the definition of $L_{FA}(q)$ (resp. of $L_{BA}(q')$) is recursive: when it exists, it is a smallest fix-point. As already seen, in a RA_{FA}, $L_{FA}(q)$ is the set of strings starting from q and reaching the final state F, whereas in a RA_{BA}, $L_{BA}(q')$ is the set of strings starting from the initial state q_0 and reaching q'. RA are not limited to producing flat trees, because recursive transitions allow a real branching, as shown in Figure 9. Real recursion occurs when, in the FA case, there exists a path starting from a state q, using a transition labeled by q and reaching F (and the corresponding situation in the BA case). We show in the following that RA_{FA} and RA_{BA} are respectively linked with the two families of CGs known as "unidirectionnal", because they make an exclusive use of either $/$ or $\backslash$.

4.2. *From Unidirectional CGs to RA*

Definition 13 (Unidirectional CGs) *A FA-unidirectional CG (resp. BA-unidirectional CG) is a CG that only assigns categories that are either basic or built from the operator $/$ (resp. $\backslash$).*

Unidirectional CGs are powerful: each one of the two families of unidirectional CGs can produce any ε-free context-free language (Bar Hillel *et al.*, 1960).

Theorem 4 (From Unidirectional CGs to RA) *Every FA-unidirectional (resp. BA-unidirectional) CG can be transformed into a strongly equivalent RA_{FA} (resp. RA_{BA}), i.e. generating the same structural descriptions.*

Proof 3 (Sketch of Proof of Theorem 4) *Let G be a FA-unidirectional CG (it is easy to adapt the proof for a BA-unidirectional CG). The construction proposed is a direct adaptation of the one described in section 2.4 for FA-regular CGs. It could also be formulated by using the CF-grammar in Chomsky Normal Form strongly equivalent with G (Kanazawa, 1998; Huet* et al.*, 2002), but we give here a shortcut. Let N be the set of every subcategory of a category assigned to a member of the vocabulary in G (a category is a subcategory of itself). The set of states for the RA_{FA} to be built is now: $N \cup \{F\}$ with $F \notin N$. The initial state is S, the final one is F. For every $A/B \in N$, define a transition labelled by A/B between the states A and B (that is: $B \in \gamma(A, A/B)$). And for every $\langle a, A\rangle \in G$, add a transition labelled by a between the states A and F ($F \in \gamma(A, a)$). The structures produced by both device are the same because every transition not leading to F in the RA_{FA} corresponds to an instantiation of the rewriting scheme FA by members of N: these rules are of the form $A \longrightarrow A/B\ B$, with $A, B \in N$ and mean that a subtree rooted by A can be decomposed into two subtrees: one rooted by A/B and one rooted by B. This is exactly what is also expressed in the RA_{FA}.*□

Corollary 1 *The set of RA_{FA} (resp. RA_{BA}) recognizes every ε-free CF language.*

Conversely, every recursive automaton can easily be transformed into a strongly equivalent CF-grammar in Chomsky Normal Form. So, their expressivity at the string and structure levels are exactly the same (except for languages with ε). Note also that it is easy to define a recursive automaton with infinite loops.

Example 9 *The classical FA-unidirectional CG recognizing the language $a^n b^n$, $n \geq 1$, is the following: $G_{FA} = \{\langle a, S/B\rangle, \langle a, (S/B)/S\rangle, \langle b, B\rangle\}$. Similarly, the classical BA-unidirectional CG recognizing the same language is: $G_{BA} = \{\langle a, A\rangle, \langle b, A\backslash S\rangle, \langle b, S\backslash(A\backslash S)\rangle\}$. The corresponding RA_{FA} and RA_{BA} are given in Figure 10. They are different from the RA_{FA} of Figure 8, even if they recognize the same language. These RA can be simplified: here, in the RA_{FA}, you can delete the state $(S/B)/S$ and replace the label of the recursive transition between S/B and S by a (and similarly for the RA_{BA}). But it is not possible for the state S/B, which is* really recursive*. Note finally that there is no hope that the RA_{FA} of Figure 10 belongs to the output set of the BP algorithm for some sample D made of only flat trees with FA internal nodes, because it contains a state S/B which is not accessible from the initial state (in the RA_{BA}, the problem comes from the state $A\backslash S$ from which the final state is not accessible). As a matter of fact, in any $MCA(D)$ built from such a sample*

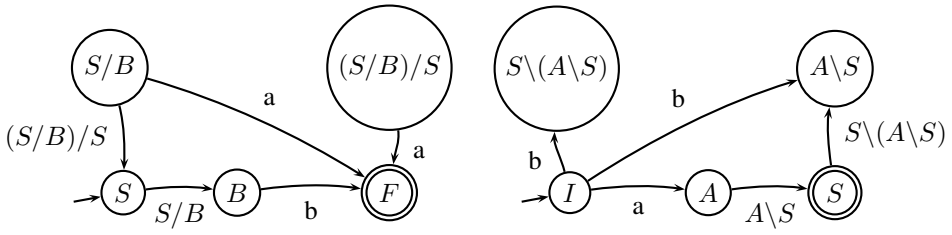

Figure 10. *a RA_{FA} and a RA_{BA} both recognizing $a^n b^n, n \geq 1$*

D, every state is accessible from the initial state and allows to reach the final state, and this property is preserved by merging transitions.

4.3. *Mutually Recursive Automata for AB-Categorial Grammars*

Unidirectional CGs and CF grammars have the same expressivity at the string level, but bidirectional CGs are useful for linguistic purposes, because of the *structures* they produce. It is thus natural to try to extend our finite state-like model to the general case of CGs, where both FA and BA rules are used. As we have seen, it is possible to represent the use of FA rules in a RA_{FA} and the use of BA rules in a RA_{BA}. So, we propose to represent a (bidirectional) CG by a *pair* of *mutually recursive automata*: one is a RA_{FA}, the other one is a RA_{BA}. For a syntactic analysis that uses both FA and BA rules, mutual calls between both recursive automata will be necessary. After an introducing example, we provide a definition of mutually recursive automata, and show that they have the same expressivity at the structure level as CGs.

Example 10 (Example of MRA) *Let us translate the CG given in Example 2 into a pair of mutually recursive automata (cf. Figure 11). The states of these automata correspond to every possible sub-category of a category assigned to a member of the vocabulary (duplicated in each RA), plus a final state F in the RA_{FA} (above), and an initial state I in the RA_{BA} (under). The transitions are designed in exactly the same way as in the proof of Theorem 4. Then, the RA have been simplified as explained in Example 9, but not as much as possible: here, we have chosen to preserve the representation of all the finite vocabulary in both automata. As the language recognized by the grammar is finite, a simple FSA recognizing the same string language could be defined. But a pair of MRA is necessary to represent the* structures *generated by the initial CG, especially to label the nodes by either FA or BA.*

Definition 14 (Mutually Recursive Automata and their Language) *A pair of **mutually recursive automata** (MRA in the following) is a pair $M = (A_{FA}, A_{BA})$ where $A_{FA} = \langle Q \cup \{S_{FA}, F\}, \Sigma, \gamma_{FA}, S_{FA}, F\rangle$ is RA_{FA} and $A_{BA} = \langle Q \cup \{I, S_{BA}\}, \Sigma, \gamma_{BA}, I, S_{BA}\rangle$ is a RA_{BA} sharing the same vocabulary Σ and the same*

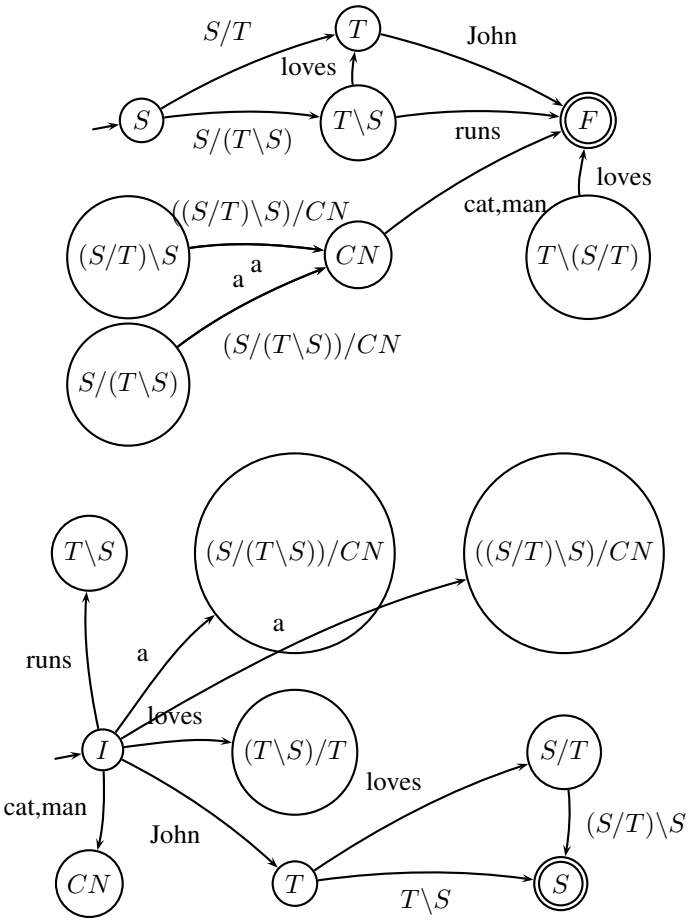

Figure 11. *a pair of mutually recursive automata: the RA_{FA} and the RA_{BA}*

set of state names Q except for their initial and final states ($S_{FA} \notin Q$, $F \notin Q$, $I \notin Q$ and $S_{BA} \notin Q$). We define $L_M(F) = \varepsilon = L_M(I)$ and for every state $q \in Q$, the language $L_M(q)$ of the state q in M is the smallest set such that:

– $L_{FA}(q) \cup L_{BA}(q) \subseteq L_M(q)$

– *if there exists a transition labeled by $r \in Q$ between q and q' in A_{FA} (resp. in A_{BA}), i.e. $q' \in \gamma_{FA}(q, r)$ (resp. $q' \in \gamma_{BA}(q, r)$) then: $L_M(r).L_{FA}(q') \subseteq L_{FA}(q)$ (resp. $L_{BA}(q).L_M(r) \subseteq L_{BA}(q')$).*

We define the language of the MRA as: $L(M) = L_M(S_{FA}) \cup L_M(S_{BA})$.

Theorem 5 *For every CG G there exists a pair of MRA $M = (A_{FA}, A_{BA})$ strongly equivalent with G.*

Proof 4 (Sketch of Proof of Theorem 5) *This theorem is a natural generalization of Theorem 4, the strongly equivalent grammar in Chomsky Normal Form playing the role of intermediary. Each member of the finite vocabulary must be present in at least one of the RA of the pair (or in both like in Figure 11). In both automata, a recursive transition labeled by S from its initial state and its final state can be added (it is not compulsary in most cases).*□

At the moment, we do not know if it is possible to produce the same structures as the ones produced by a CG with another unique finite-state-like device. The problem is not the structures themselves, but the labels FA, and BA associated to each node of each tree.

5. Learning Context-Free Languages Represented by CGs

Now that we have a more complete correspondence between CGs and automata, we try in this last section to improve or re-interpret learning results in terms of recursive automata. In the first subsection, we identify new subclasses of unidirectionnal CGs adapted to learning from flat structures. In the second one, we reinterpret the general BP algorithm in terms of operations applying on MRA.

5.1. *Learning Unidirectional Categorial Grammars from Flat Structures*

In this subsection, we go back to string languages. In this case, as we have seen, unidirectional CGs are enough because they can generate every ε-free context-free language. Learning context-free languages from strings is a difficult challenge. Kanazawa proved that for every k, $\mathcal{G}_k$ is learnable from strings but the learning strategy he suggests is not at all efficient. As a matter of fact, it consists in generating every possible functor-argument structure compatible with every string of the input data, then applying the BP algorithm on these structures, as if they were real strutural examples. It is very expensive a strategy, not tractable in practice. But Example 8 showed that at least some context-free languages ($a^n b^n$ was the canonical example) could be learned from strings, supposing that the underlying structural examples were flat. The purpose of this sub-section is to study this approach in more details.

Definition 15 (Reduced form (Kanazawa, 1998)) *A substitution σ is said to be faithful to a CG G if for all $\langle v, A\rangle \in G$ and $\langle v, B\rangle \in G$ such that $A \neq B$, then $\sigma(A) \neq \sigma(B)$. A faithful substitution is a substitution which is not a unifying substitution. Let $\sqsubseteq$ be the binary relation between CGs defined by : $G_1 \sqsubseteq G_2$ if and only if there exists a substitution σ faithful to G_1 such that $\sigma(G_1) \subseteq G_2$. A grammar G is*

said in reduced form *if there is no G' such that $G' \sqsubset G$ and $FL(G) = FL(G')$. It is decidable whether a grammar is in reduced form and, for any CG, one can find a grammar in reduced form generating the same structured language.*

This notion of "Reduced Form" is mainly to justify that only unifying substitutions can be used in a learning algorithm: as a matter of fact, by using faithful substitutions (i.e. substitutions which are not unifying substitutions) you do not extend the class of accessible structured languages, so you can afford to avoid them.

Theorem 6 (BP Algorithm and Grammars in Reduced Form (Kanazawa, 1998)) *For any CG $G \in \mathcal{G}_k$ in reduced form, for any sequence $\langle T_i \rangle_{i \in \mathbb{N}}$ enumerating $FL(G)$, we have: $\exists n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$, G belongs to the set $BP_k(\langle T_1, ..., T_n \rangle)$.*

Theorem 6 provides a sufficient condition to belong to the output of the BP algorithm. We will use this condition to define new classes of CGs adapted to learning from strings. Let:
$\mathcal{G}_{k,FA}^{\mathcal{S}ub} = \{\sigma(G) | G \in \mathcal{G}_k \cap \mathcal{G}_{FA}^r$ and G is in reduced form and σ is a unifying substitution for $G\}$.

Of course, it is also possible to similarly define $\mathcal{G}_{k,BA}^{\mathcal{S}ub}$. Elements of $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ are FA-unidirectional CGs, because unifying substitutions cannot introduce new directions of functional applications. Similarly, elements of $\mathcal{G}_{k,BA}^{\mathcal{S}ub}$ are BA-uniderectional CGs. And for every $k \geq 1$, we have: $\mathcal{G}_{k,FA}^{\mathcal{S}ub} \subset \mathcal{G}_k$ (resp. $\mathcal{G}_{k,BA}^{\mathcal{S}ub} \subset \mathcal{G}_k$) because unifying sustitutions can only decrease the maximal number of distinct categories assigned to the same member of the vocabulary.

What more can be said about the languages generated by the grammars of these classes, i.e. what are the extensions of $\cup_{k \geq 1}\{L(G) | G \in \mathcal{G}_{k,FA}^{\mathcal{S}ub}\}$ and of $\cup_{k \geq 1}\{L(G) | G \in \mathcal{G}_{k,BA}^{\mathcal{S}ub}\}$? First note that both sets include the set of every regular languages. As a matter of fact, $\cup_{k \geq 1}\{L(G) | G \in \mathcal{G}_k \cap \mathcal{G}_{FA}^r\}$ and $\cup_{k \geq 1}\{L(G) | G \in \mathcal{G}_k \cap \mathcal{G}_{FA}^r\}$ both contain every regular language (see commentaries following the proof of Theorem 2), for each regular language there exists a CG in reduced form in each set generating this language and in both cases $\sigma = Id$ can be considered as a special case of unifying substitution.

But Example 8 also showed that for some k, $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ contains a grammar which generates the non-regular language $a^n b^n$. As a matter of fact, in this example, $\sigma_1(GF(D)) \in \mathcal{G}_3 \cap \mathcal{G}_{FA}^r$ and is in reduced form and σ_2 is a unifying substitution for it, so $\sigma_2 \circ \sigma_1(GF(D)) \in \mathcal{G}_{3,FA}^{\mathcal{S}ub}$. But we will see in the following that elements of $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ cannot generate every ε-free context-free language. Before this, let us first characterize the elements of these classes.

For every k, the main interest of the class $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ is that it naturally extends the class $\mathcal{G}_k \cap \mathcal{G}_{FA}^r$, whose fundamental property is that its members produce structural examples that are only flat trees with FA internal nodes. This property can be used to adapt Kanazawa's standard learning algorithm.

Theorem 7 *For every $k \geq 1$, for every $G \in \mathcal{G}_{k,FA}^{\mathcal{S}ub}$ (resp. $G \in \mathcal{G}_{k,BA}^{\mathcal{S}ub}$), there exists a set $D \subset FL(G)$ only made of flat right-branching trees with FA internal nodes (resp. left-branching trees with BA internal nodes) and there exists τ, a unifying substitution for $GF(D)$ such that $G = \tau(GF(D))$. Such a set D will be called a* characteristic sample of G.

Proof 5 (proof of Theorem 7) *For every $G \in \mathcal{G}_{k,FA}^{\mathcal{S}ub}$, by definition there exist $G' \in \mathcal{G}_k \cap \mathcal{G}_{FA}^r$ in reduced form and σ a unifying substitution for G' such that $G = \sigma(G')$. We know by Theorem 6 that for any sequence $\langle T_i \rangle_{i \in \mathbb{N}}$ enumerating $FL(G')$, we have: $\exists n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$ G' belongs to the set output by the BP algorithm whose inputs are k and $\langle T_1, ..., T_n \rangle$. Let $D = \langle T_1, ..., T_{n_0} \rangle$. As G' is FA-regular, $FL(G')$ only contains right-branching flat trees with FA-internal nodes, and so does D. The fact that G' belongs to the output of BP from D means that there exists a unifying sustitution ρ for $GF(D)$ such that $G' = \rho(GF(D))$. So $G = \sigma(G') = \sigma(\rho(GF(D)))$. Let $\tau = \rho \circ \sigma$. τ is a composition of two unifying substitutions, so it is also a unifying substitution for $GF(D)$ and, as expected $G = \tau(GF(D))$. And $D \subset FL(G') \subset FL(\sigma(G'))$ (as already stated in the commentaries of Property 1, applying a substitution is a generalization operation), so $D \subset FL(G)$.□*

Theorem 7 means that the members of $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ can always be obtained by applying unifying substitutions on the general form $GF(D)$ obtained from a sample D of their structured language made of only right-branching flat trees with FA internal nodes. This suggests an adaptation of the standard Kanazawa's strategy to identify CGs belonging to the class $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ from strings (see Algorithm 1). The main difference between this algorithm and Kanazawa's is that it is no longer necessary to associate every possible functor-argument structures to each string: it is enough to associate right-branching flat structures with FA internal nodes which is, of course, much less expensive. But, as the set D is unknown, flat structures will be associated only to input strings not already recognized (associated with any structure) by the current hypotheses grammars (remember that a^3b^3 is recognized by the CG of Example 8 but not with a flat structure).

Let us suppose that Algorithm 1 is given an integer $k \geq 1$ and strings produced by a grammar $G \in \mathcal{G}_{k,FA}^{\mathcal{S}ub}$ whose characteristic sample is D. Theorem 7 ensures that as soon as the input set of strings includes the set of strings corresponding to the elements of the sample D (which will always occur in the limit), then for some j, the set $R_{j,k}$ contains G among its output set. Of course, the whole process can be adapted to handle $\mathcal{G}_{k,BA}^{\mathcal{S}ub}$, by supposing left-branching flat trees with BA internal nodes.

This algorithm to identify the set $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ compatible with a set of strings is exponentially less expensive than the one proposed by Kanazawa to identify the set of k-valued grammars compatible with the same set of strings.

Elements of $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ are at most k-valued. But it is possible that some language generated by a k_0-valued bidirectionnal CG can only be generated by a grammar in

Algorithm 1 algorithm to identify elements of $\mathcal{G}^{\mathcal{S}ub}_{k,FA}$ from strings

Require: $\langle s_0, ..., s_i \rangle$ where $\forall i$, $s_i \in \Sigma^+$ and k
1: $j \longleftarrow 0$
2: **repeat**
3: $C_j = \{s_0, ..., s_j\}$ \\ try C_j as a sample
4: associate a right-branching flat structure with FA internal nodes to every element of C_j to get the set of structural examples D_j
5: apply $BP_k(D_j)$ to find the set $R_{j,k} \subset \mathcal{G}_k$ of CGs compatible with this set
6: discard all elements of $R_{j,k}$ whose string language does not include $\{s_{j+1}, ..., s_i\}$
7: $j \longleftarrow j + 1$
8: **until** $j = i + 1$ OR $R_{j,k} \neq \emptyset$
Ensure: $R_{j,k}$: a set of CGs in $\mathcal{G}^{\mathcal{S}ub}_{k,FA}$ whose string language includes $\langle s_0, ..., s_i \rangle$

$\mathcal{G}^{\mathcal{S}ub}_{k_1,FA}$, where $k_1 > k_0$ because grammars in $\mathcal{G}^{\mathcal{S}ub}_{k_1,FA}$ are more constrained than k_0-valued CGs. So, the k needed by Algorithm 1 may be greater than the one needed by the BP algorithm to identify a grammar generating the same language. Grammars in $\mathcal{G}^{\mathcal{S}ub}_{k,FA}$ can be considered as having a special normal form: they produce only right-branching flat trees or right-branching flat trees of right-branching flat trees (cf Example 9).

For $k = 1$, BP is also called RG and is by itself an efficient learning algorithm. But note also that every rigid grammars is in reduced form and the only unifying substitution that can be applied to it is Id, so $\mathcal{G}^{\mathcal{S}ub}_{1,FA} = G_1 \cap \mathcal{G}^r_{FA}$, is a subclass of FA-regular CGs.

If $k > 1$, as usual, to make this algorithm a learning algorithm in the sense of Gold, inclusion tests may still be necessary to avoid over-generalization. In fact, inclusion between (string) context-free languages is undecidable. To overcome this problem, Kanazawa suggests to perform inclusion tests for the strings of bounded length l generated by every grammar present in the output set at each step l of the algorithm. This test is computable and, in the limit, allows to select the grammar generating the smallest string language. Such a strategy should be applied here too (but is not tractable).

Finally, Theorem 7 allows to identify some ε-free context language which cannot be generated by any grammar in any $G \in \mathcal{G}^{\mathcal{S}ub}_{k,FA}$. $a^n b^n c$ for $n \geq 1$ is such a language. Let us suppose that $\exists k_0 \geq 1$ and $\exists G_0 \in \mathcal{G}^{\mathcal{S}ub}_{k_0,FA}$ such that $L(G_0) = a^n b^n c$, $n \geq 1$. By Theorem 7, there exists a characteristic sample D_0 for G_0, made of right-branching flat trees with FA-internal nodes. So, D_0 only contains structural examples that take the form of Figure 12. As shown in Figure 9, to generalize context-free languages from examples of this kind, it is necessary to define substitutions that unify the label of an internal node and the label of a leaf in such trees, allowing to substituate a subtree to a leaf. But every subtree of every element of D_0 has a rightmost leaf labeled by c.

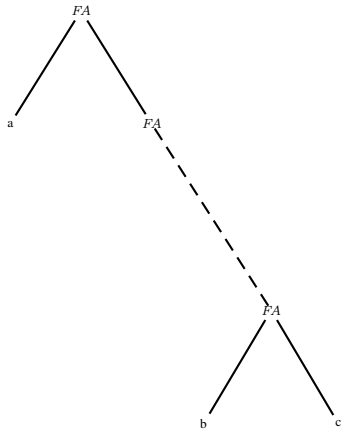

Figure 12. *a right-branching flat tree with FA internal nodes for an element of $a^n b^n c$*

So, it is impossible not to duplicate c by substituting a subtree to a leaf in such a tree. So, the language generated by $\tau(GF(D_0))$ can never be $a^n b^n c$, $n \geq 1$.

It is interesting to notice that $a^n b^n c$, $n \geq 1$ can easily be generated by an element $G \in \mathcal{G}_{k,BA}^{\mathcal{S}ub}$, for some $k \geq 1$. On the contrary, no $G \in \mathcal{G}_{k,BA}^{\mathcal{S}ub}$ for any $k \geq 1$ can generate $ca^n b^n$. So the set of languages generated by both classes are not the same. This shows that Algorithm 1 and the corresponding variant adapted for $\mathcal{G}_{k,BA}^{\mathcal{S}ub}$ do not have the same expressivity. It could thus be interesting to combine both algorithms by applying them alternatively, associating to strings the corresponding set of right-branching flat trees with FA-internal node on one hand, and the corresponding set of left-branching flat trees with BA-internal node on the other hand. Unfortunately, languages like $ca^n b^n c$, $n \geq 1$ can be generated neither by any element of any $\mathcal{G}_{k,FA}^{\mathcal{S}ub}$ nor by any element of any $\mathcal{G}_{k,BA}^{\mathcal{S}ub}$ and is thus still out of reach of such an algorithm.

5.2. *Another View on the BP Algorithm*

Finally, in this last subsection, we illustrate the BP algorithm in the general case (i.e. for bidirectional CGs) on an example. We do not provide any new result here, but just a new perspective on the same learning strategy, using the notion of mutually recursive automata introduced in the previous section. Example 11 thus generalises what has already been observed in Examples 7 and 8.

Example 11 (BP Algorithm Reinterpreted) *Let us apply the BP algorithm, as already stated in subsection 3.3, to a set of structural examples using both FA and BA labels. The set D of structural examples is made of the trees of Figure 12. After applying the steps 1 to 3 of the BP algorithm, we obtain the trees of Figure 13.*

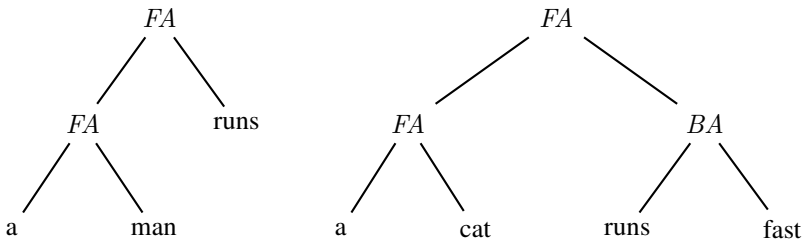

Figure 13. *a set of structural examples*

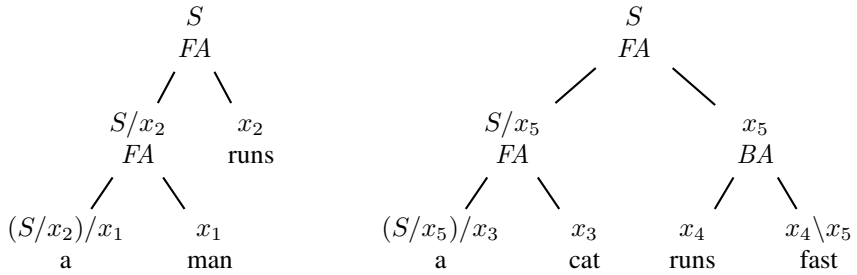

Figure 14. *a set of structural examples labeled by the BP algorithm*

The general form $GF(D)$ obtained has thus the following assignments:
$GF(D) = \{\langle a, (S/x_2)/x_1\rangle, \langle a, (S/x_5)/x_3\rangle, \langle man, x_1\rangle, \langle runs, x_2\rangle, \langle runs, x_4\rangle, \langle cat, x_3\rangle, \langle fast, x_4\backslash x_5\rangle\}$

The pair of mutually recursive automata representing $GF(D)$ is given in Figure 14: the FA-automaton is above, the BA-automaton is under (some transitions have several labels for readability). It is possible to define a substitution σ such that $\sigma(GF(D))$ is rigid: let $\sigma(x_1) = \sigma(x_3) = x_1$, $\sigma(x_2) = \sigma(x_5) = \sigma(x_4) = x_2$ and σ is the identity elsewhere. Then, $\sigma(GF(D))$ has the following assignments:
$GF(D) = \{\langle a, (S/x_2)/x_1\rangle, \langle man, x_1\rangle, \langle runs, x_2\rangle, \langle cat, x_3\rangle, \langle fast, x_4\backslash x_5\rangle\}$

As in Example 8, the substitution σ can be interpreted as opering transition merges (and thus state merges) on the automata of the pair, resulting in the new pair of recursive automata given in Figure 15.

This CG (and similarly, of course, the corresponding pair of MRA) recognizes new sentences like "a man runs fast fast fast": some generalization has occurred.

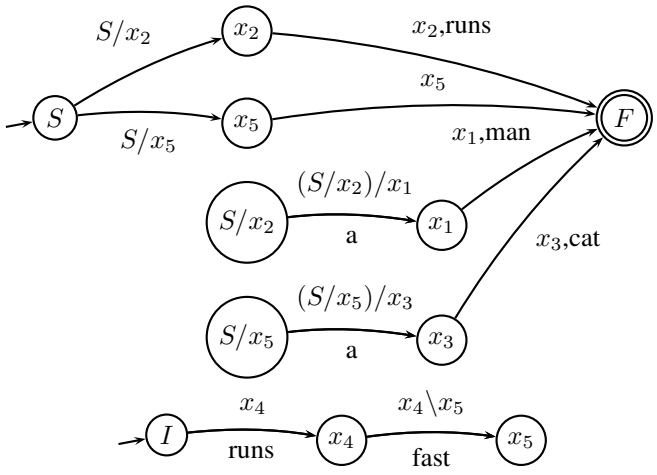

Figure 15. *the pair of mutually recursive automata representing* $GF(D)$

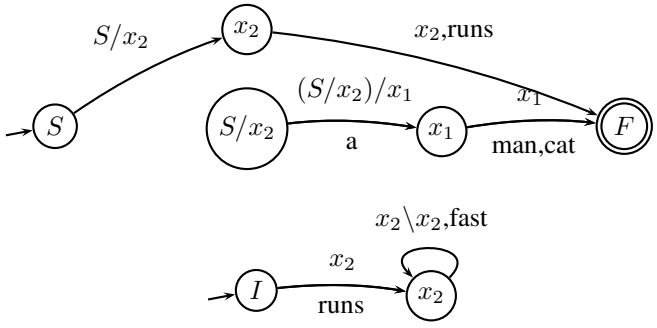

Figure 16. *the pair of mutually recursive automata representing* $\sigma(GF(D))$

6. Conclusion

To conclude, this study shows that the domain of regular grammatical inference and the domain of CGs learning can be integrated into a unified framework. The first benefits of this unification is the translation of results from one domain to the other one with very few efforts, and a better understanding of the nature of the generalization operator used in each domain.

By trying to translate subclasses of GCs into finite state automata, we finally allowed to extend the definition of finite state automata to give them the same expressive power at the structure level as general CGs. This is another (less expected) result of this work. It shows that the inference of context-free grammars may not be so different from the inference of regular grammars as it first seemed.

Our work also provides new ideas to learn context-free languages from strings. Learning context-free languages from strings is known difficult because strings underspecify the possible underlying structures. The solution we suggest is to *constrain as much as possible* these structures. Unidirectional CGs are better adapted to represent context-free-languages for inference from strings than general CGs, because they generate the same class of string languages while being more constrained at the structure level. Classes $\mathcal{G}_{k,BA}^{\mathcal{S}ub}$ and $\mathcal{G}_{k,BA}^{\mathcal{S}ub}$ are even more interesting because, for each grammar of these classes, there exists a characteristic sample of examples obtained in linear time from strings which is enough to ensure that the grammar is in the search space of an algorithm. The grammars in these classes produce flat trees of flat trees, so they are constrained to produce structures which are as close as possible to strings. Learning these classes from strings is not much more expansive than learning them from structural examples.

But a lot remains to be done. The exact expressivity of our new classes should be characterized more precisely. And a remaining problem with CGs is the fact that for a given language they can generate, no unique canonical representative CG generating this language has been identified. The existence of such a canonical representative exemplar, playing the role of a target grammar, would avoid the inclusion tests that make the full learning algorithms untractable. For regular languages, the minimal deterministic finite state automaton (Oncina *et al.*, 1992) or the canonical RFSA (Denis *et al.*, 2002) play this role. And, of course, experiments should be performed to test the real efficiency of our proposed algorithm.

Another possible perspective concerns the adaptation of this work to learning algorithms from positive and negative examples, or to the identification of other more powerful subclasses of categorial grammars like Lambek grammars (Lambek, 1958).

7. References

Aho A. V., Ullmann J. D., *The Theory of Parsing, Translation and Compiling, vol 1: Parsing*, Habile, 1972.

Angluin D., « Inductive Inference of Formal Languages from Positive Data », *Information and Control*, vol. 45, n° 2, p. 117-135, May, 1980.

Angluin D., « Inference of Reversible Languages », *Journal of the ACM*, vol. 29, n° 3, p. 741-765, July, 1982.

Bar Hillel Y., Gaifman C., Shamir E., « On Categorial and Phrase Structure Grammars », *Bulletin of the Research Council of Israel*, 1960.

Besombes J., Marion J.-Y., « Learning Reversible Categorial Grammars from Structures », *Categorial Gramars*, p. 148-163, 2004.

Buszkowski W., Penn G., « Categorial grammars determined from linguistic data by unification », *Studia Logica*, vol. 49, p. 431-454, 1990.

Cornuéjols A., Miclet L., *Apprentissage artificiel ; concepts et algorithmes*, Editions Eyrolles, 2002.

Denis F., Lemay A., Terlutte A., « Some language classes identifiable in the limit from positive data », *ICGI'02*, LNAI 2484, Springer Verlag, Amsterdam, p. 63-76, 2002.

Dupont P., Miclet L., Vidal E., « What is the search space of the regular inference », *in* , S. Verlag (ed.), *ICGI'94*, LNAI 862, Heidelberg, p. 25-37, 1994.

Florêncio C. C., « Consistent Identification in the Limit of Rigid Grammars from Strings Is NP-hard », *ICGI'02*, LNAI 2484, Springer Verlag, Amsterdam, p. 49-62, 2002.

Florêncio C. C., « Consistent Identification in the Limit of Any of the Classes k-valued Is NP-hard », *LACL'01*, LNAI 2099, Springer Verlag, p. 125-134, 2001.

Gold E., « Language Identification in the Limit », *Information and Control*, vol. 10, p. 447-474, 1967.

Gécseg F., Steinby M., « Tree Languages », *Handbook of Formal Languages*, vol. 3, Springer Verlag, p. 1-68, 1996.

Hopcroft J., Ullman J., *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

Huet G., Rétoré C., *Survey of a few fundamental representation structures for computational linguistics*, ESSLI lectures. 2002.

Joshi A., Schabes Y., *Handbook of Formal Languages, vol3*, Springer Verlag, chapter Tree-Adjoining Grammars, p. 69-120, 1997.

Kanazawa M., « Identification in the Limit of Categorial Grammars », *Journal of Logic, Language and Information*, vol. 5, n° 2, p. 115-155, 1996.

Kanazawa M., *Learnable Classes of Categorial Grammars*, FoLLI, CLSI Publication, 1998.

Lambek J., « The Mathematics of Sentence Structure », *American Mathematical Monthly*, vol. 65, p. 154-170, 1958.

Oehrle R. T., Bach E., Wheeler D. (eds), *Categorial Grammars and Natural Language Structures*, D. Reidel Publishing Company, Dordrecht, 1988.

Oncina J., Garcia P., « Inferring regular languages in polynomial update time », *Pattern Recognition and Image Analysis*, p. 49-61, 1992.

Sakakibara Y., « Learning context-free grammars from structural data in polynomial time », *Theoretical Computer Science*, vol. 76, p. 223 - 242, 1990.

Sakakibara Y., « Efficient Learning of Context-Free Grammars from Positive Structural Examples », *Information and Computation*, vol. 97, n° 1, p. 23-60, May , 1992.

Woods W. A., « Transition Network Grammars of Natural Language Analysis », *Communication of the ACM*, vol. 13, p. 591-606, 1970.

Yokomori, « On Polynomial-Time Learnability in the Limit of Strictly Deterministic Automata », *Machine Learning*, vol. 2, p. 153-179, 1995.