How to Split Recursive Automata

Isabelle Tellier

How to Split Recursive Automata

Isabelle Tellier

LIFL - Inria Lille Nord Europe
university of Lille
isabelle.tellier@univ-lille3.fr**

In this paper, we interpret in terms of operations applying on extended finite state automata some algorithms that have been specified on categorial grammars to learn subclasses of context-free languages. The algorithms considered implement *specialization strategies*. This new perspective also helps to understand how it is possible to control the combinatorial explosion that specialization techniques have to face, thanks to a typing approach.

1 Introduction

There are often several ways to represent a language: it is well known that every regular language can be specified either by a regular grammar or by a deterministic finite state automaton. Context-free languages can also be specified by different kinds of devices. In recent previous papers [17, 18], we have shown that some classes of categorial grammars (CGs in the following), generating context-free languages, could easily be represented by a family of extended automata called recursive automata (RA). This translation allowed to exhibit connexions between two previously distinct approaches of grammatical inference from positive examples: the one used in [3, 13, 14] to learn CGs, and the one used to learn regular grammars represented by finite state automata [1, 10]. This was possible because both employ a *generalization strategy*. In particular, the generalization operators used in both contexts were shown to be similar.

Now, we want to apply the same process for *specialization strategies from positive examples*. In such strategies, the initial hypothesis is too general a grammar (or set of grammars) and each example is considered as a *constraint* which restricts the search space, until it is reduced to the target grammar. We show here that the translation of CGs into RA, which has helped to better understand the family of generalization strategies, can also help to better understand the family of specialization strategies. As a matter of fact, although barely used, specialization approaches have been proposed independently in both backgrounds: to learn subclasses of CGs in the one hand [16], and to learn regular grammars represented by finite state automata in the other hand [11]. A first move towards that direction has been briefly proposed in [19], but limited to unidirectional CGs. In this paper, we generalize the approach to its full generality.

** This work was partly supported by the ANR MDCO "CroTal"

To reach this aim, we first need to recall in section 2 how to transform a CG into a RA preserving the structures produced, in both unidirectional and bidirectional cases. In section 3, we first briefly present the specialization strategy described by Moreau in [16], allowing to learn rigid CGs from positive examples. We then explain how it relates to the specialization strategy proposed by Fredouille and Miclet in [11], which targets regular languages represented by finite state automata. We show that Moreau's algorithm can be interpreted as some kind of well founded "state splitting" strategy applying on RA. Finaly, the whole picture is completed in section 4, by a new interpretation of yet another already known algorithm allowing to learn CGs from sentences enriched by lexical types [8, 7]. It appears to be an efficiently controled specialization approach.

This paper thus proposes neither any new algorithm or result, nor any experiment, but it suggests a new stimulating look on already known strategies.

2 From categorial grammars to recursive automata

2.1 Basic definitions of categorial grammars

Definition 1 (Categories, Categorial Grammars and their Language). *Let $\mathcal{B}$ be a set (at most countable) of basic categories containing a distinguished category $S \in \mathcal{B}$, called the axiom. $Cat(\mathcal{B})$ is the smallest set such that $\mathcal{B} \subset Cat(\mathcal{B})$ and for any $A, B \in Cat(\mathcal{B})$: $A/B \in Cat(\mathcal{B})$ and $B\backslash A \in Cat(\mathcal{B})$. Unidirectional variants allow only one of these operators (either / or \) but not both. For every finite vocabulary Σ and for every set $\mathcal{B}$ containing S, a* ***categorial grammar*** *(or CG) is a finite relation G over $\Sigma \times Cat(\mathcal{B})$. We note $\langle v, C\rangle \in G$ the assignment of the category $C \in Cat(\mathcal{B})$ to the element of the vocabulary $v \in \Sigma$. The syntactic rules of a CG take the form of two rewriting schemes: $\forall A, B \in Cat(\mathcal{B})$*

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

Unidirectional CGs make use of only one of these rules (either FA or BA) but not of both. The language generated (or recognized) by a CG G is:
$L(G)$=$\{w = v_1 \dots v_n \in \Sigma^+ \mid \forall i \in \{1, \dots, n\}, \exists C_i \in Cat(\mathcal{B})$ such that $\langle v_i, C_i\rangle \in G$ and $C_1 \ \dots \ C_n \rightarrow^* S\}$,
where $\rightarrow^*$ is the reflexive and transitive closure of the relation $\rightarrow$, defined by FA and BA schemes. For every $w \in L(G)$, a syntactic analysis structure can be produced, taking the form of a binary-branching tree whose leaf nodes are assignments of G and whose internal nodes are labelled either by FA or BA and by a category (see Figure 1).

Example 1 (a simple CG). CGs have mainly been used to represent natural language syntax, as illustrated by this example. Let $\mathcal{B} = \{S, T, CN\}$ where T stands for "term" and CN for "common noun", $\Sigma = \{John, runs, a, man, fast\}$ and $G = \{\langle John, T\rangle, \langle runs, T\backslash S\rangle, \langle man, CN\rangle, \langle a, (S/(T\backslash S))/CN\rangle, \langle fast, ((T\backslash S)\backslash(T\backslash S))\rangle\}$. This over-simple CG recognizes sentences like "John runs" or "a man runs fast" with the syntactic analysis structures of Figure 1.

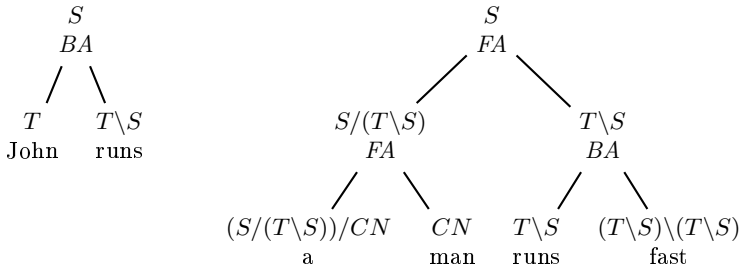

Fig. 1. Syntactic analysis structures produced by a CG

2.2 Recursive Automata and their Language

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

We restrict ourselves here to recursive automata (RA in the following) with unique initial and final states, but it is not a crucial choice. The only important difference between this definition and the usual definition of finite state automata is that, in a RA, it is possible to *label a transition either by an element of Σ or by an element of Q*. To use a transition labelled by a state, you have to produce a string belonging to the language of this state. RA can thus be considered as special cases of "recursive transition networks" or RTRs [20]. But, depending on the notion of "state language" used, there exist in fact two distinct notions of RA which will be called, for reasons that will become clear soon, RA_{FA} and RA_{BA}. In a RA_{FA}, the language $L_{FA}(q)$ associated with the state $q \in Q$ 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.

Definition 3 (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 $q \in Q$ we define the language $L_{FA}(q)$ (resp. $L_{BA}(q)$) associated with q as the smallest set satisfying:*

- *$\epsilon \in L_{FA}(F)$ (resp. $\epsilon \in L_{BA}(q_0)$);*
- *if there exists a transition labelled by $a \in \Sigma$ between q and $q' \in Q$, 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 labelled by $r \in Q$ between q and $q' \in Q$, 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 $q \in Q$ such that $q \neq F$ (resp. $q \neq q_0$), the definition of $L_{FA}(q)$ (resp. of $L_{BA}(q)$) may be recursive: when it exists, it is a smallest fix-point. A

real recursion occurs when, in a RA_{FA}, there exists a path starting from a state q, using a transition labelled by q and reaching F (resp., in a RA_{BA}, when there exists a path starting from q_0, using a transition labelled by q and reaching the state q). Unlike finite state automata, RA are not limited to producing flat trees, because recursive transitions allow a real branching. We have shown in [19] that RA_{FA} and RA_{BA} are respectively linked with the two possible unidirectional CGs. This property, which justifies their name, is detailed in the following.

2.3 From Unidirectional CGs to RA

Each one of the two families of unidirectional CGs can produce any ϵ-free context-free language [2]. Here, we show that every FA-unidirectional (resp. BA-unidirectional) CG can be easily transformed into a strongly equivalent RA_{FA} (resp. RA_{BA}), i.e. generating the same structural descriptions [19]. The process, for a given FA-unidirectional (resp. BA-unidirectional) CG G, is the following :

- the vocabulary Σ of the RA is the same as the one of G.
- 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} (resp. RA_{BA}) to be built is $N \cup \{F\}$ with $F \notin N$ (resp. $N \cup \{I\}$ with $I \notin N$). The initial state is S (resp. I), the final one is F (resp. S).
- for every $C \in N$, define a transition labelled by C between the states C and F (resp. between I ans C), i.e. $F \in \gamma(C, C)$ (resp. $C \in \gamma(I, C)$).
- for every $A/B \in N$ (resp. $A\backslash B \in N$), define a transition labelled by A/B (resp. $A\backslash B$) between the states A and B, that is: $B \in \gamma(A, A/B)$ (resp. $B \in \gamma(A, A\backslash B)$).
- for every $\langle v, , \rangle C \in G$, add a transition labelled by v between the state C and F, i.e. $F \in \gamma(C, v)$ (resp. add a transition between I and C labelled by v, i.e. $C \in \gamma(I, v)$).

2.4 Mutually Recursive Automata

Both families of unidirectional CGs have the expressivity of ϵ-free context-free languages at the string level, but bidirectional CGs are useful for linguistic purposes, because of the *structures* they produce, and particularly the labels FA or BA assigned to each internal node. It is thus natural to try to extend our notion of RA to the general case of bidirectional 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* (MRA in the following): one element of the pair 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 the two RA will be necessary. After an introducing example, we provide a general definition of MRA and give some of their properties.

Example 2 (Example of a MRA). Let us translate the CG G given in Example 1 into a MRA (cf. Figure 2). The states of each of these RA correspond to every possible subcategory of a category assigned by G to a element of the vocabulary, plus a final state F in the RA_{FA} (above), and an initial state I in the RA_{BA} (under). The transitions have been designed exactly as explained before. Then, each RA has been simplified for readability (some un-necessary states and transitions are deleted), but not as much as possible: here, we have chosen to preserve the representation of all the final vocabulary Σ in both automata.

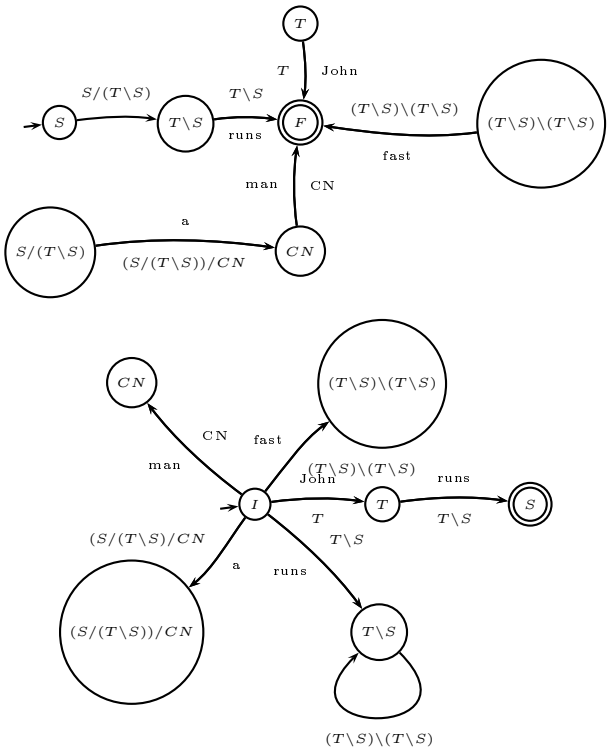

Fig. 2. A pair of mutually recursive automata: the RA_{FA} and the RA_{BA}

Definition 4 (MRA and their Language). *A pair of* ***mutually recursive automata*** *(or MRA) is a pair $M = (R_{FA}, R_{BA})$ where $R_{FA} = \langle Q \cup \{F\}, \Sigma, \gamma_{FA}, S_{FA}, F\rangle$ is a RA_{FA} and $R_{BA} = \langle Q \cup \{I\}, \Sigma, \gamma_{BA}, I, S_{BA}\rangle$ is a RA_{BA} sharing the same vocabulary Σ and the same set of state names Q except for the final state of the RA_{FA} ($F \notin Q$) and for the initial state of the RA_{BA} ($I \notin Q$). We consider $\epsilon \in L_{FA}(I)$ and $\epsilon \in L_{BA}(F)$ 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 labelled by* $r \in Q$ *between* q *and* $q' \in Q$ *in* R_{FA} *(resp. in* R_{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: $L(M) = L_M(S_{FA}) \cup L_M(S_{BA})$.

For every CG G, there exists a MRA $M = (R_{FA}, R_{BA})$ strongly equivalent with G, i.e. generating the same structures.

3 Learning by specialization

3.1 Learning rigid CG from positive examples

A rigid CG is a CG in which every $v \in \Sigma$ is assigned at most one category. Kanazawa has proved [13, 14] that the set of every (bidirectional) rigid CG is learnable *in the limit* (i.e. in the sense of [12]) from positive examples, i.e. from sentences. Two distinct learning algorithms are now available for this purpose. The best known is Kanazawa's, derived from "BP" (proposed earlier by Buszkowski and Penn [3]) and is a classical *generalization strategy*. The other one, called RGPL (Rigid Grammar Partial Learning) is described by Moreau in [16]. It is this second algorithm that we will concentrate on here. Although its author did not present it this way, we show that it is in fact a *specialization strategy*.

Let us first illustrate how it works on a simple example. We suppose that the available set of positive examples is {"John runs", "a man runs fast"}. At its first step, the algorithm assigns to each member of the vocabulary used at least once in the examples a distinct variable. This initial assignment is thus here:
$\mathcal{A} = \{\langle John, x_1\rangle, \langle runs, x_2\rangle, \langle a, x_3\rangle, \langle man, x_4\rangle, \langle fast, x_5\rangle\}$.
Even if a word is used several times in the examples, only one variable is introduced because the target grammar is rigid. In fact, $\mathcal{A}$ implicitely specifies *a set of grammars*: the set of rigid CGs built on the used vocabulary. As a matter of fact, every such rigid CG G can be obtained by applying a substitution σ from the set of variables to a set of categories to $\mathcal{A}$ such that:
$G = \sigma(\mathcal{A}) = \{\langle v, \sigma(C)\rangle | \langle v, C\rangle \in \mathcal{A}\}$
The substitution σ has only the effect of renaming the variables into categories.

Of course, $\mathcal{A}$ can also be represented by a MRA $M = (R_{FA}, R_{BA})$. In this MRA, R_{FA} (resp.R_{BA}) has $\{x_1, x_2, x_3, x_4, x_5, F\}$ (resp. $\{I, x_1, x_2, x_3, x_4, x_5\}$) as set of states, and each state x_i for $1 \leq i \leq 5$ is connected to F (resp. I is connected to x_i) by a transition labelled by the corresponding word (another transition labelled by x_i should be added but it is useless at this point). As S appears nowhere in this MRA, the language it recognizes is empty. But it is a compact way to represent *the whole class of rigid CGs built on* Σ.

Then, each sentence is syntactically parsed with the assigments in $\mathcal{A}$, by a CYK-like algorithm. The only two possible ways to parse "John runs" are :

– either to replace x_1 by S/x_2: then a FA rule can be applied
– either to replace x_2 by $x_1\backslash S$: then a BA rule can be applied

These kinds of substitutions express a *constraint* that the variables (x_1 or x_2) must satisfy: $\mathcal{A}$ must thus be updated to a disjunction of sets of assignments, each subset corresponding to a subclass of rigid CGs. A simpler way to store the current subsets of possible solutions is to store the set of possible substitutions that can be applied to $\mathcal{A}$. In our case, this set is made of $\{\sigma_1, \sigma_2\}$, with $\sigma_1(x_1) = S/x_2$ and is equal to the identical function elsewhere, and $\sigma_2(x_2) = x_1\backslash S$ and is equal to the identical function elsewhere. $\sigma_1(\mathcal{A})$, as well as $\sigma_2(\mathcal{A})$, can be represented by a MRA derived from the previous one. This time, both MRA recognize exactly the sentence "John runs".

To parse "a man runs fast", many more solutions are possible. The maximum theoretical number is $5 * 2^3 = 40$ because there are 5 possible binary branching trees with 4 leaves (this can be computed in the general case by the Catalan number), and each of them has 3 internal nodes which can receive either a FA or a BA label. This makes $40 * 2 = 80$ theoretical possible substitutions by combining the constraints obtained from both sentences (the combinaison is a classical composition of functions). But, among them, some are contradictory: as the target grammar is rigid, the unique category assigned to the word "runs" cannot be of the form x_i/x_j and $x_k\backslash x_l$ at the same time. We thus see where the initial class plays a role in the learning strategy.

It is easy to see that the main problem with this algorithm is the combinatorial explosion it has to face, especially when examples do not share any common word. This is not surprising, since the problem of learning rigid CGs from sentences is known to be NP-hard [4].To limit this explosion, Moreau proposes to exploit as much initial knowledge as possible, in the form of an *initial grammar*, that is, initially known categories for some usual words (for example the lexical ones) which cannot be renamed, as it is the case for the variable categories.

Furthermore, there is no guarantee at all that this strategy always converges to a unique solution. In theory, to fulfill the requirements of learnabiblity *in the limit*, when several possible compatible grammars are available, inclusion tests should be performed to select the one generating the "smallest" language. This problem also occurred with Kanazawa's algorithm, when applied to sentences.

3.2 State merges and state splits

The previous strategy can now be interpreted in terms of operations applying on MRA. As we have seen, at every step of this algorithm, the search space is a disjunction of sets of assignments of the form $\sigma(\mathcal{A})$ for some substitution σ, and each of them can be represented by a MRA. The MRA corresponding with $\mathcal{A}$ recognizes no sentence. But, as soon as at least one example has been treated (and, thus, the category S been introduced), $\sigma(\mathcal{A})$ specifies a set of CGs recognizing at least this example. What is the effect of a constraint on a MRA ?

The constraints always take the form: $x_k = x_l$, where x_k and x_l are already introduced variables or equal to S, or $x_k = X_m/X_n$ or $x_k = X_m\backslash X_n$, with X_m and X_n any category built on the set of every variable union S.

- the effect of a constraint of the form $x_k = x_l$ on a MRA is a *state merge* in both the RA_{FA} and the RA_{BA} of the MRA. As, in MRA, x_k can also be used as transition labels, corresponding *transition merges* can also occur.
- the effect of a constraint of the form $x_k = X_m/X_n$ (resp. $X_m\backslash X_n$) can be decomposed into four steps:
  1. X_m/X_n (resp. $X_m\backslash X_n$) replaces x_k everywhere in the MRA;
  2. every subcategory of X_m and X_n (including themselves) not already identified (i.e. not already a sub-category of the previous set of assignments) becomes a new state in both RA: in the RA_{FA}, it is linked to the state F (resp. in the RA_{BA} from the state I) by a transition labelled by its name;
  3. in the RA_{FA} (resp. in the RA_{BA}), a new transition labelled by X_m/X_n (resp. $X_m\backslash X_n$) links the states X_m and X_n, and the same occurs for every newly identified subcategory;
  4. the states and transitions of the same name are merged in each RA.

This operation can now be compared to the "state splitting strategy" proposed by Fredouille and Miclet in [11] to learn regular languages represented by finite state automata by specialization. For example, the constraint $x_1 = S/x_2$ has the effect of splitting the state x_1 into two new states: S and x_2. Then, as a state named x_2 already exists, the new one is merged with the previous one. But our specialization operation is more general than Fredouille and Miclet's, because of the recursive nature of the automata on which it applies. Furthermore, their algorithm was a specialization strategy *at the language level*: their initial hypothesis was the most general regular language Σ^* and constraints were used to *specialize the language*. Moreau's algorithm is a specialization strategy *at the set of grammars level*: its initial hypothesis is the set of possible grammars, and the examples are used to introduce constraints that reduce this set to subsets. The corresponding MRA represents a *set of grammars* and not only a specific language. For these reasons, our approach cannot be easily adapted to usual finite state automata. But we believe that our state splitting operator is better founded than the previous one, because it is the formal counterpart of well-defined substitutions.

4 Learning from typed examples revisited

We now show that the algorithm proposed in [8, 7] to learn CGs from typed examples can be considered as a specialization strategy where state splits and state merges are controlled by a typing approach.

4.1 Learning from semantically typed examples

The idea of learning CGs from typed examples was first introduced in [8]. The types considered in this work are borrowed from Montague's theory [6]: they are lexicalized terms derived from syntactic categories by a morphism, and coincide

with the type of the logical formula that translates the associated word. Learning from typed examples is cognitively relevant because types can be interpreted as semantic information available in the environment or previously learned. In this section, we briefly recall this notion in a general fashion and give the conditions under which they can help learning.

The notion of types useful for learning CGs is based on:

- a finite set τ of basic types among which is a distinguished type $t \in \tau$ standing for "truth values": usually, this set is $\tau = \{e, t\}$ where $e \in \tau$ is the type of "entities". Montague also used a type s for "intensions" that will not be used in the following;
- the set $\mathit{Types}(\tau)$ of every type is the smallest set such that $\tau \subset \mathit{Types}(\tau)$ and for every type $\alpha, \beta \in \mathit{Types}(\tau)$, $\langle \alpha, \beta \rangle \in \mathit{Types}(\tau)$. $\langle \alpha, \beta \rangle$ is the type of functions that require an argument of type α and provide a result of type β.

Types in $\mathit{Types}(\tau)$ are useful for learning a CG only if they are connected with its syntactic categories in $\mathit{Cat}(\mathcal{B})$. More precisely, the necessary condition to be fulfilled is that there exists a homomorphism h such that:

- for every basic category $C \in \mathcal{B}$, $h(C)$ is defined and belongs to $\mathit{Types}(\tau)$. The distinguished category $S \in \mathcal{B}$ is associated with the distinguished type $t \in \tau$: $h(S) = t$.
- for every other category in $\mathit{Cat}(\mathcal{B})$ of the form A/B or $A\backslash B$, we have: $h(A/B) = h(B\backslash A) = \langle h(B), h(A) \rangle$.

Example 3 (classical semantic types for natural languages). Let $\tau = \{e, t\}$. The words of the grammar defined in Example 1 receive the following semantic types:

- "John" can be considered as an entity of type e;
- "runs" and "man" are one-place predicates; their type is: $\langle e, t \rangle$;
- "fast" is a "one-place-predicate modifier", i.e. it transforms a predicate of arity one into another one of the same arity: it thus has the type $\langle \langle e, t \rangle, \langle e, t \rangle \rangle$;
- finaly, if we follow Montague's intuition about the "proper treatment of quantification" [6], the determiner "a" has the most complex type: $\langle \langle e, t \rangle, \langle \langle e, t \rangle, t \rangle \rangle$.

The corresponding homomorphism h is defined by: $h(S) = t$, $h(T) = e$, $h(CN) = \langle e, t \rangle$. As required, if $\langle v, C \rangle \in G$, the semantic type of v is $h(C)$.

4.2 How types help to control state splits and state merges

Learning from typed examples means learning from sentences where each element of the vocabulary $v \in \Sigma$, which should be assigned $C \in \mathit{Cat}(\mathcal{B})$ by the target grammar G to analyse this sentence, is provided with the corresponding type $h(C) \in \mathit{Types}(\tau)$. As we will see, the learning strategy proposed in [8, 7] can also be interpreted in terms of operations applying on MRA. We illustrate this algorithm on our example. The input data are now of the form:

John	runs
e	$\langle e, t \rangle$
e	$x_1 \langle e, t \rangle$

a	man	runs	fast
$\langle\langle e,t\rangle,\langle\langle e,t\rangle,t\rangle\rangle$	$\langle e,t\rangle$	$\langle e,t\rangle$	$\langle\langle e,t\rangle,\langle e,t\rangle\rangle$
$x_2\langle x_3\langle e,t\rangle, x_4\langle x_5\langle e,t\rangle,t\rangle\rangle$	$x_6\langle e,t\rangle$	$x_1\langle e,t\rangle$	$x_7\langle x_8\langle e,t\rangle, x_9\langle e,t\rangle\rangle$

In these typed examples, the third line is the result of a simple pre-treatment which consists in introducing variables in front of every "functional type". The variables are all distinct, except when the same couple "word, type" occurs (as it is the case here for the couple "runs, $\langle e,t\rangle$"). These variables will eventually take the value "/" or "\" during the learning process. The initial set of assignements is, this time:
$\mathcal{A} = \{\langle John, e\rangle, \langle runs, x_1\langle e,t\rangle\rangle, \langle a, x_2\langle x_3\langle e,t\rangle, x_4\langle x_5\langle e,t\rangle,t\rangle\rangle\rangle, \langle man, x_6\langle e,t\rangle\rangle,$
$\langle fast, x_7\langle x_8\langle e,t\rangle, x_9\langle e,t\rangle\rangle\rangle\}$.
As previously, $\mathcal{A}$ implicitely specifies a *set of grammars*. This set is much larger than the one of rigid CGs: it is the set of CGs which can assign an arbitrary number of distinct categories to each word (so, it intersects every class of k-valued CGs), but for which there exists a homomorphism such that every distinct category assigned to the same word gives rise to a distinct type. In formal terms, it is such that there exists a homomorphism h satisfying:
$\forall\langle v, C_1\rangle, \langle v, C_2\rangle \in G,\ h(C_1) = h(C_2) \Longrightarrow C_1 = C_2$.
We have shown [9] that for every ϵ-free context-free language, it is possible to define a CG generating this language, a set of types and a homomorphism such that this property is satisfied. This new target class is learnable in the limit from typed examples [8, 7].

As previously, $\mathcal{A}$ can also be represented by a MRA. But the information carried by the types is much richer than the one carried by the basic variables Moreau used: types can be interpreted as some kind of maximal bound on the possible categories they replace; they display all their potential renaming.

The learning algorithm applies as in section 3.1: it consists in trying to parse each sentence with the rules *FA* and *BA* adapted to types so as to reach the type t at the root, by defining constraints on the variables (see [7] for details). The only possible type-compatible way to parse the first typed example "John runs" is to have: $x_1 = \backslash$, meaning that only a *BA* rule is compatible with the type assignments. "runs" should thus finaly receive the category $e\backslash t$. This time, there is only one type-compatible way to parse "a man runs fast": this parse (isomorphic to the one in Figure 1) is shown on Figure 3. Both typed examples lead to the following (unique) set of constraints: $x_2 = /$, $x_3 = x_6$, $x_7 = \backslash$, $x_8 = x_1 = \backslash$, $x_4 = /$, $x_5 = x_9$.

The set of assignments is thus updated to:
$\mathcal{A} = \{\langle John, e\rangle, \langle runs, \backslash\langle e,t\rangle\rangle, \langle a, /\langle x_3\langle e,t\rangle, /\langle x_5\langle e,t\rangle,t\rangle\rangle\rangle, \langle man, x_3\langle e,t\rangle\rangle,$
$\langle fast, \backslash\langle\backslash\langle e,t\rangle, x_5\langle e,t\rangle\rangle\rangle\}$.
If we apply the process of section 2.4 to this set (after re-ordering the types to make them similar to syntactic categories and t playing the role of S), we obtain the MRA of Figure 4. In this example, with only two typed examples, we obtain a unique MRA which is nearly isomorphic to the target one.

In this context, the constraints take the form $x_i = x_j$, $x_i = /$ or $x_i = \backslash$ and give rise to the same transformations as the one detailed in section 3.2. It could

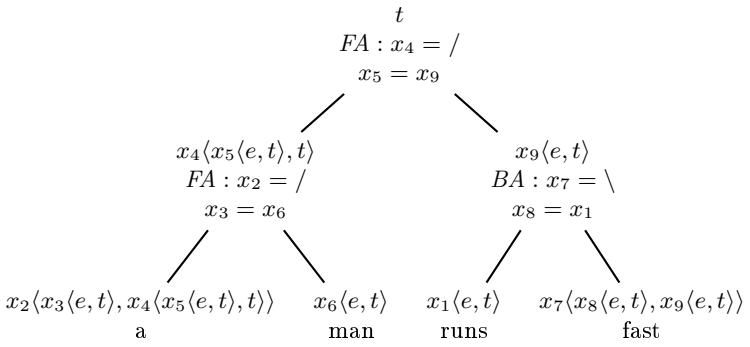

Fig. 3. parse tree for a typed example

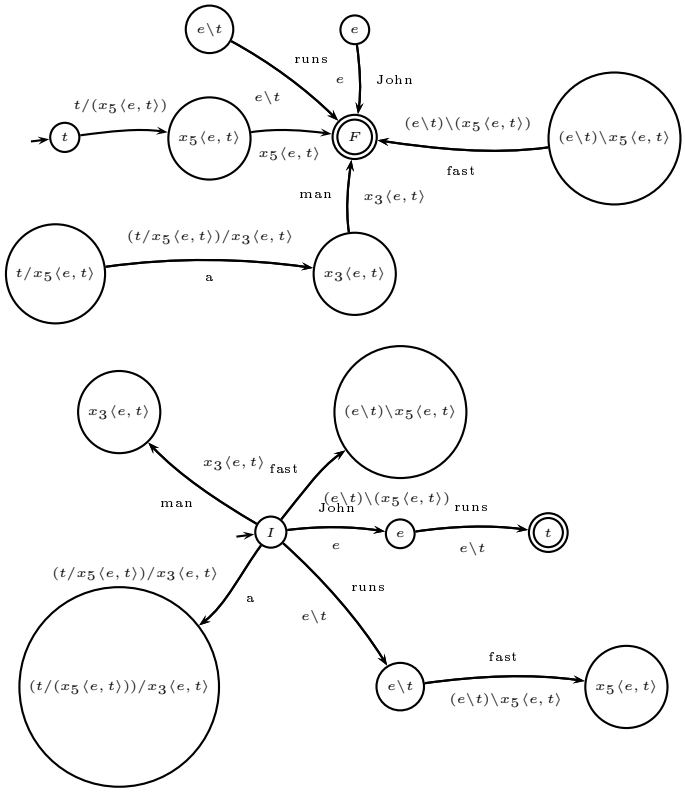

Fig. 4. MRA for type assignments

seem that the first kind corresponds to a state merge and the other two to a state split, but the situation is a bit more complex. In our example, to reach the target, only one constraint is missing: $x_5 = \backslash$. The typed example corresponding to the sentence "John runs fast" would provide this constraint. Its first effect on the MRA would be to rename the state $x_5\langle e, t\rangle$ both in the RA_{FA} and in the RA_{BA} by $\backslash\langle e, t\rangle$, that is $e\backslash t$. But, doing so, this state becomes identical to an already existing one and then must be merged to it.

The table of Figure 5 explains why types help the algorithm to avoid a combinatorial explosion and to converge quicker. We have seen that there always exists a homomorphism σ between column 2 and column 3, which is the target of the learning process. Hypotheses about types ensure that there also exists a homomorphism h between column 3 and column 4. This situation is similar to the one described in [15], and analyzed in [5]. The two learning algorithms presented here are both specialization strategies at the *set of grammars level*, but their initial hypothesis is either a lower bound or an upper bound of the set of categories of the target grammar. Types are efficient because they allow to control the possible renamings.

vocabulary	*Moreau's initial assigment*	*target category*	*pre-treated initial assignment types*
John	x_1	T	e
a	x_2	$(S/(T\backslash S)/CN$	$x_2\langle x_3\langle e, t\rangle, x_4\langle x_5\langle e, t\rangle, t\rangle\rangle$
man	x_3	CN	$x_6\langle e, t\rangle$
runs	x_4	$T\backslash S$	$x_1\langle e, t\rangle$
fast	x_5	$(T\backslash S)\backslash(T\backslash S)$	$x_7\langle x_8\langle e, t\rangle, x_9\langle e, t\rangle\rangle$

Fig. 5. tabular showing the starting points and target of the two algorithms

5 Conclusion

In grammatical inference from positive examples, two sources of information are usually available: the target class and the set of examples. Generalization techniques use the examples to generate a "most specific grammar" compatible with them (the prefix tree automaton in the case of regular languages), and then use the target class to generalize it. Specialization techniques do the contrary: the class is the starting point and the examples help to specialize it.

In this paper, we propose a new perspective on these techniques. First, we see that disjunctions of MRA are able to represent the *search space* of such learning algorithms. Second, we show that the algorithm to learn CGs from typed examples proposed in [8, 7] introduces type control into the process. The initial semantic types associated with the elements of the vocabulary specify some kind of maximal bound on the possible renamings, allowing to limit the combinatorial explosion of solutions.

References

1. D. Angluin. Inference of Reversible Languages *Journal of the ACM* 3: p.741–765, 1982.
2. Y. Bar Hillel and C. Gaifman and E. Shamir. On Categorial and Phrase Structure Grammars. *Bulletin of the Research Council of Israel*, 9F, 1960.
3. W. Buszkowki and G. Penn. Categorial grammars determined from linguistic data by unification, newblock *Studia Logica*, p.431–454, 1990.
4. C. Costa-Florencio. Consistent Identification in the Limit of Rigid Grammars from Strings Is NP-hard. proceedings of the *IGGI: Algorithms and Applications*, p.49–62, LNAI 2484, 2002.
5. F. Coste, D. Fredouille, C. Kermovant, C. de la Higuera. Introducing domain and typing bias in automata inference, in *proceedings of the 7th ICGI*, Vol. 3264 of *LNAI*, p.115–126, 2004.
6. D. Dowty and R.E. Wall and S. Peters. *Introduction to Montague Semantics*, Linguistics and Philosophy, Reidel, 1981.
7. D. Dudau-Sofronie. *Apprentissage de grammaires catégorielles pour simuler l'acquisition du langage naturel à l'aide d'informations sémantiques*, thèse d'informatique, université Lille3, 2004.
8. D. Dudau-Sofronie, I. Tellier, and M. Tommasi. Learning categorial grammars from semantic types, in *proceedings of the 13th Amsterdam Colloquium*, p. 79–84, 2001.
9. D. Dudau-Sofronie, I. Tellier, and M. Tommasi. A Learnable Class of CCGs from Typed Examples, in *proceedings of the 8th conference on Formal Grammars*, p. 77–88, 2003.
10. P. Dupont and L. Miclet and E. Vidal. What is the search space of the regular inference. proceedings of *ICGI*, p.25–37, LNCS 862, 1994.
11. D. Fredouille, and L. Miclet. Experiences sur l'inference de langage par specialisation, in *proceedings of CAP'2000*, p.117–130, 2000.
12. E.M. Gold. Language identification in the limit. *Information and Control*, 10: p.447–474, 1967.
13. M. Kanazawa. Identification in the limit of categorial grammars, *Journal of Logic, Language and Information* 5(2), p.115–155, 1996.
14. M. Kanazawa. *Learnable Classes of Categorial Grammars*, CSLI Publications. 1998.
15. C. Kermovant, C. de la Higuera Learning language with help proceedings of *ICGI*, p.161–173, LNAI, 2002.
16. E. Moreau. Apprentissage partiel de grammaires lexicalisées, *TAL* 45(3), p.71–102, 2004.
17. I. Tellier. When categorial grammars meet regular grammatical inference, in *proceedings of the 5th LACL*, Vol. 4492 of *LNAI*, p.317–332, 2005.
18. I. Tellier. Learning recursive automata from positive examples, *Revue d'Intelligence Artificielle* New Methods in Machine Learning(20/2006), p.775–804, 2006
19. I. Tellier. Grammatical inference by specialization as a state splitting strategy, in *proceedings of the 16th Amsterdam Colloquium*, p.223-228,2007.
20. W.A. Woods Transition Network Grammars of Natural Language Analysis *Communication of the ACM* vol.13,p.591–606, 1970.