# Building factorized TAGs with meta-grammars

Éric Villemonte de La Clergerie

# Building factorized TAGs with meta-grammars

**Éric Villemonte de la Clergerie**
INRIA - Rocquencourt - B.P. 105
78153 Le Chesnay Cedex, FRANCE
Eric.De_La_Clergerie@inria.fr

## Abstract

Highly compacted TAGs may be built by allowing subtree factorization operators within the elementary trees. While hand-crafting such trees remains possible, a better option arises from a coupling with meta-grammar descriptions. The approach has been validated by the development of FRMG, a wide-coverage French TAG of only 207 trees.

## 1 Introduction

Tree Adjoining Grammars (TAG – (Joshi, 1987)), plus feature decorations, provide a powerful and elegant formalism to capture many syntactic phenomena, due to the adjoining mechanism, tree lexicalization, and extended domain of locality of trees. However, it is well-known that the two last properties easily lead to a combinatorial explosion in term of trees, with large coverage grammars of several thousand trees (or even more) (Crabbé, 2005; Abeillé, 2002). This explosion induces problems of development and maintenance of the grammars, but also of efficiency during parsing. Several approaches have been proposed to remedy to the situation, either on the maintenance side or on the efficiency side. On the maintenance side, besides the notion of families present in the XTAG architecture (Doran et al., 1994), one may cite the use of metarules (Prolo, 2002) to derive trees from the "canonical" versions, and meta-grammars (Candito, 1999; Duchier et al., 2004) where the grammars are derived from a constraint-based modular level of syntactic descriptions organized as an inheritance hierarchy of elementary classes. On the efficiency side, besides more or less clever lexicalization-based filtering techniques (such as *suppertagging*), one may cite the factorization of common sub-trees using automata (Carroll et al., 1998) or the possibility to attach, modulo regular expressions, several possible tree traversals to a tree (Harbusch and Woch, 2004). However, no approach cover both side of the problem, namely maintenance and efficiency. In particular, finding common sub-trees in a large TAG (with decorated nodes) is a difficult task (from an algorithmic point of view) and attaching tree traversals requires some efforts from the grammar writer. We propose a more modular approach based on the use of local subtree factorization operators that be expressed locally and easily in the elementary classes of a metagrammar. The generation of complete minimal trees by a meta-grammar compiler combines all these local factorizations to produce highly factorized trees that couldn't easily have been written by hand. These ideas have been validated during the development of FRMG, a large coverage French meta-grammar producing only 207 trees.

Some background about Meta-Grammars is provided in Section 2. Tree factoring through MG descriptions is illustrated by a few syntactic phenomena in Section 3. In Section 4, we present FRMG. Because a grammar is only useful with a parser, Section 5 precises some aspects of this parser, focusing on those that ensure its efficiency. Finally, Section 6 presents some results.

## 2 Meta-Grammars

Meta-Grammars favor the modular development of grammars by grouping small sets of elementary constraints in *classes* related to micro syntactic phenomena. Elementary constraints include node equal-

ity (A=B), node dominance (immediate with A >> B or not with A >>+ B) and node precedence (A < B). Decoration constraints as feature structures may also be attached to nodes (node v: [person: 1|3, mood :~imperative|gerundive] ) or to a whole class (desc :[ extraction : −] ), allowing, as values, constants, recursive feature structures, (possibly negated) finite set values, and variables. Path-based equations may also be used to unify decorations (node(Det). top.number = node(N).top.number or desc. diathesis = node(V).top. diathesis ).

Classes are organized in a multiple inheritance hierarchy (using the <: operator), allowing to progressively refine and enrich syntactic notions (for instance the notion of subject to get extracted subjects, impersonal subjects, French post-verbal subject, ...), a class inheriting its ancestors' constraints.

A crossing mechanism is used to combine the terminal classes (e.g. the classes with no children, wrt the inheritance hierarchy). The existing MG formalisms implement various flavors of crossing mechanisms, the general idea being to accumulate the constraints of the parent classes while checking that they remain satisfiable. Close from the original MGs (Candito, 1999) but constrained to be monotonic, we use a resource-based crossing mechanism: a class $C^-$ may require some resource $R$ (− subject) while another class $C^+$ may provide this resource $R$ (+ subject). In that case, we try to combine $C^-$ and $C^+$, neutralizing the resource $R$. The basic resource-based mechanism has been extended with the notion of namespace, allowing a class $C^-$ to require the same resource $R$ several times but in distinct namespaces $N_i$ (for instance, requiring two instances of agreement constraints on distinct nodes with − det :: agr and −root :: agr). The constraints from the $R$-provider class $C^+$ are renamed with namespace $N$ to avoid names clashes for nodes, variables, and resources, as shown by the following equation:

$$C^-[-N::R \cup \mathcal{K}^-] \oplus C^+[+R \cup \mathcal{K}^+] = (C^- \oplus N::C^+)[=N::R \cup \mathcal{K}^- \cup N::\mathcal{K}^+]$$

The surviving neutral classes (i.e. those requiring or providing no resources) are used to generate a minimal set of minimal trees, given their constraints. Again, the notion of tree minimality depends on the flavor of MGs and also on the target syntactic formalism (for instance MC-TAGs for (Kallmeyer et al., 2008)). In general, a minimal tree does not introduce nodes not mentioned in the constraints and replaces non immediate dominance constraints by parent relations. In our case, we also try also to preserve tree factoring as much as possible.

## 3 From MGs to factorized trees

Tree factoring relies on regexp-like operators working on nodes, or more precisely on the subtrees rooted by these nodes. The (informal) notation $T[(t_1 \text{op } t_2)]$ denotes the application of the operator op on subtrees $t_1$ and $t_2$ in the context of tree $T$.

### 3.1 Disjunction

The first operator concerns disjunction over nodes, with $T[(t_1; t_2)]$ straightforwardly equivalent to the set of trees $T[t_1]$ and $T[t_2]$. At the level of MGs, disjunction is explicitly introduced with special nodes carrying the information type: alternative . The alternatives nodes are largely used, for instance to represent all possible realizations for a subject as sketched in the following class:[1]

```
class collect_real_subject {
  node SAlt: [type: alternative];
  SAlt >> S_Cl;
  node S_Cl: [type: coanchor, cat: cl];
  SAlt >> S_NP;
  node S_NP: [type: subst, cat: np];
  SAlt >> S_Sent;
  node S_Sent: [type: subst, cat: S]; ...}
```

### 3.2 Optionality and guards

From disjunction immediately derives the optionality operator with $T[t?] \equiv T[(\epsilon; t)]$. At MG level, a node may be marked as optional using the `optional` feature. Note that even a special alternative node may be made optional, for instance the previous SAlt node.

However, most of the times, a node is made optional with conditions: positive (resp. negative) guards[2] may be used to control the presence (resp. absence) of a node. A guard $G$ is a Boolean disjunctive and conjunctive formula over path equa-

[1] The examples are simplified versions of classes in FRMG.

[2] The term *guard* comes from the Constraint Programming community.

tions. For instance, the presence of a subject may be (naively) controlled by the verb mood, with:

```
SAlt =>
  node(V).top.mood = ~imperative;
~ SAlt =>
  node(V).top.mood = imperative;
```

More formally, a guard $G$ is equivalent to a finite set $\Sigma_G$ of substitution, implying that $T[(G_+, t; G_-)]$ may be replaced by the finite set of trees $\{T[t]\sigma \| \sigma \in \Sigma_{G_+}\} \cup \{T[\epsilon]\sigma \| \sigma \in \Sigma_G\}$.

More than one guard may be progressively attached to a node while crossing classes. The positive guards from one part and the negative ones from the other part are separately combined using conjunction, which may finally lead to complex guards. However, for neutral classes, a set of rewriting rules is used to reduce the guards by removing the parts that are trivially true or false[3]. Obviously, if a guard is of the form $G = G_1 \wedge G_2$ and $G_1$ is shown to be trivially $\mathrm{true}$ (resp. $\mathrm{false}$) then $G$ may be reduced to $G_2$ (resp. to $\mathrm{false}$).

Guards are heavily used in FRMG. Actually, positive guards are also used on non optional nodes to attach disjunctive constraints to a node, or constraints to a node under a disjunctive node, for instance to state that a sentential subject should be in subjunctive mood:

```
S_Sent +
 node(V).top.mood=value(~subjunctive)
```

### 3.3 Repetition

The Kleene star operator provides subtree repetition, with $T[t*] \equiv \{T[\epsilon], T[t], T[(t,t)], \ldots\}$. More formally, the Kleene operator may be removed by introducing an extra node category $X_{t*}$ used as a substitution node in $T[X_{t*}]$, and two extra trees $X_{t*}(\epsilon)$ and $X_{t*}(t, X_{t*})$.[4]

At MG level, a (possibly special) node may be repeated using the `star` feature. Concretely, in FRMG, the Kleene star operator have only been used to represent repetition of coordinated components in coordination.

```
class coord {
node Seq: [type: sequence, star: *];
Seq < SeqLast;
Seq >> Punct_Comma;
Seq >> coord2
SeqLast >> coo;
SeqLast >> coord3;
coo < coord3; }
```

### 3.4 Shuffling

Less known but well motivated in (Nederhof et al., 2003) to handle free word ordering, the *shuffling* (or interleaving) of two sequences $(a_i)_{i=1\cdots n} \# \# (b_j)_{j=1\cdots m}$ returns all sequences containing all $a_i$ and $b_j$ in any order that preserves the original orderings (i.e. $a_i < a_{i+1}$ and $b_j < b_{j+1}$). For instance, the shuffling of $a, b$ with $c, d$ returns the sequences "$a, b, c, d$", "$a, c, b, d$", "$a, c, d, b$", "$c, a, b, d$", "$c, a, d, b$", and "$c, d, a, b$". In our case, the shuffle operator $\#\#$ is used on sequences of subtrees, with in particular $T[(t_1 \# \# t_2)] \equiv \{T[(t_1, t_2)], T[(t_2, t_1)]\}$.

At MG level, shuffling naturally arises from underspecification of the ordering between sibling nodes. For instance, the constraints "N >> N_1, N>> N_2, N >> N_3, and N_1 < N_2" produce the (minimal) tree fragment $N((N_1, N_2) \# \# N_3)$, stating that $N_3$ may occur anywhere (before, between, after) relatively to the sequence $N_1, N_2$. In FRMG, free node ordering is, in particular, present between verb arguments, including inverted subject, such as in

le livre que donne (à Paul) (son ami ...)
the book that gives (to Paul) (his friend ...)

To block free node ordering, one has to explicit the precedence constraints or use the special rank feature with the first or last values to force the position of a node wrt its siblings.[5] Finally, the shuffle operator is systematically expanded when covering a foot node, in order to ease the detection of TIG auxiliary trees (Section 5).

### 3.5 Some complements

The above-presented operators are first generic, being adaptable for many grammatical formalisms and not just for TAGs. Secondly, they do not change the expressive power of TAGs. As sketched, they may indeed be progressively removed to get a finite set of

[3] An equation is trivially true (resp. false) if true (resp. false) without further instantiation of the decorations.

[4] This scheme only applies if $t$ does not cover a foot node, and therefore Kleene stars are not allowed over such subtrees.

[5] of course, it is an error to have several sibling nodes carrying the first (or the last) value.

standard TAG trees, possibly by adding some extra new non-terminals. However, the number of extra trees may be exponential in the number of operators in a tree. Concretely, these operators provide a way to factorize a large number of trees into a single tree. Such a compact tree $S$ may be understood as representing a large set of potential traversals through the non-terminals occurring in $S$, similar in some aspects to (Harbusch and Woch, 2004).

Of course, it is important that these operators may be used at parsing time with none or low overhead (as shown in Section 6), in particular for the complex shuffle and Kleene star operators. In our case, very informally, the implementation of these two operators relies on the capacity to create and manage continuations. For the Kleene star operator, at some point one may choose between two continuations: "*exit the loop*" or "*reenter the loop*". For the shuffle operators, at each step, one may choose between the continuations "*advance in sequence 1*" and "*advance in sequence 2*".

## 4 A French Meta-Grammar

Based on the potentialities of the MG formalism and its coupling with factoring operators, FRMG was developed for French in 2004 and maintained since then. The generated grammar turned out to be a very compact grammar with only 207 trees (in May 2010), produced from 279 classes, 197 of them being terminal. This compactness does not hinder coverage or efficiency as we will see.

It may look surprising to get less trees than classes. There are two reasons for this situation, both of them resulting from the modularity of metagrammars. First, the trees are generated from the terminal classes, some of these classes inheriting from many ancestor simple classes. Secondly, some trees result from the crossing of many terminal classes.

Actually, the compactness is even more severe than it looks with only 21 trees used to cover all verbal constructions with up to 3 arguments (including subjects), covering "canonical" constructions, passive ones, extraction ones (for relatives, interrogatives, clefted, topicalizations), impersonal ones, causative ones (partially), subject inversions, support verbs (such as *faire attention à* / *take care of*), .... Two extra trees are available for auxiliary verbs. 20 trees are anchored by adjectives, providing elementary subcategorization for sentential arguments (*il est évident qu'il doit partir – it is obvious that he should leave*) and 40 for adverbs, a rather non-homogeneous syntactic category (Table 1(a)). It is difficult to describe the coverage of the grammar. Let us say that besides the verbal constructions, the grammar partially covers most punctuations, coordinations, superlatives, comparatives, floating incises (adverbs, time modifiers, ...), ...

Table 1(a) shows that 65 trees are not anchored, which does not mean they have no lexical component. It rather reflects the idea that their underlying semantic is not related to a lexical form. For instance, we use a non-anchored tree roughly equivalent to $NP(*NP, S)$ to attach relative sentences on nouns, this tree being used both when the relative pronoun in $S$ is an extracted argument or a modifier

Table 1(b) shows that compactness really arises from the factoring operators, and more specifically from guards. However, the use of these operators is not evenly distributed among all trees. Only a small set of complex trees (the verbal and adjectival ones) are concerned, as shown for tree #198 corresponding to the verbal canonical construction for most subcategorization frames. This tree results from the crossing of 36 terminal classes and is formed of 63 nodes, not including the special nodes listed in Table 1(b). It would be very difficult to craft and maintain this tree by hand. The Figure 1 shows a simplified representation of tree #198, not showing the content of the guards (for the nodes with a green background) and also not showing some nodes. The diamond-shaped nodes represent the alternative (|) and shuffle (##) nodes. In particular, we have a disjunction node (left side) over the possible realizations (nominative clitic, nominal group, sentences, prepositional group) for the subject in canonical position, and a shuffle operator (right side) over 2 possible arguments groups and a postverbal subject (which is also usable with a preposition for causative constructions).

Still, several questions arises in presence of such trees. The first one concerns the level of factorization squeezed in this tree. The unfolding of a previous version of the grammar (February 2007) produced almost 2 millions trees, 99.98% of them being generated by the verbal trees. The equivalent of

| anchored | v | coo | adv | adj | csu | prep | aux | prop. n. | com. n. | det | pro | Not anchored |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 142 | 21 | 26 | 40 | 20 | 6 | 5 | 2 | 3 | 1 | 1 | 5 | 65 |

(a) Distribution by anchors

| | Guards | Disjunctions | Interleaving | Kleene Stars |
|---|---|---|---|---|
| all trees | 2609 | 152 | 22 | 27 |
| tree #198 | 106 | 6 | 1 | 0 |

(b) Distribution by factorization operators

Table 1: Grammar anatomy

tree #198 was the most productive one with around 700000 unfolded trees.[6]

Given these figures, one may wonder about the potential overgenerativity of the grammar and its overhead at parsing time. Practically, overgeneration seems to be adequately controlled through the guards, with no obvious overhead. Another reason explaining this behaviour may come from the constraints provided by the forms of the input string. Indeed, tree #198 covers many subcategorization frames, much more than the number of frames usually attached to a given verb. The notion of family attached to a frame as defined in the XTAG model (Doran et al., 1994) is therefore no longer pertinent. Instead, we use a more flexible mechanism based on the notion of *hypertags* (Kinyon, 2000). A hypertag is a feature structure, issued from the class decorations, providing information on the linguistic phenomena covered by a tree or allowed by a word. The anchoring of a tree by a word is only possible if their hypertags do unify, as illustrated by Fig. 2. The verb "promettre/ `to promise`" may anchor tree #198, only selecting (after unification) the presence of an optional object (`arg1.kind`) and of an optional prepositional object (`arg2.kind`) introduced by "à/*to*" (`arg2.pcas`). The link between an hypertag $\mathcal{H}$ and the allowed syntactic constructions is done through the variables occurring in $\mathcal{H}$ and in the guards and node decorations. A partial anchoring is done at load time to select the potential trees, given the input sentence, and a full anchoring is then performed, on demand, during parsing. Anchoring through hypertags offers a powerful way to restrict the generative power of the factorized trees.

[6]It should be noted that while it is easy to naively unfold the operators, it is much more difficult and costly to get a minimal set of unfolded trees.

## 5 Building efficient parsers

The French TAG grammar is compiled offline into a chart parser, able to take profit of the factoring operators. Actually, several optimizations, developed for TAGs (and not related to MGs), are also applied to improve the efficiency of the parser.

The first optimisation is a static analysis of the grammar to identify the auxiliary trees that behave as left or right auxiliary trees as defined in Tree Insertion Grammars (TIG – (Schabes and Waters, 1995)), a variant of TAGs that may be parsed in cubic time rather than $O(n^6)$ for TAGs. Roughly speaking, TIG auxiliary trees adjoin material either on the left side or the right side of the adjoined node, which actually corresponds to the behavior of most auxiliary trees. TIG and TAG auxiliary trees may be used simultaneously leading to a hybrid TAG/TIG parser (Alonso and Díaz, 2003).

Another static analysis of the grammar is used to identify the node features that are left unmodified through adjoining, greatly reducing the amount of information to be stored in items, and potentially increasing computation sharing. A third static analysis is used to compute a left-corner relation to be used at parsing time.

When parsing starts, besides the non-lexicalized trees that are always loaded, the parser selects only the trees that may be anchored by a form of the input sentence, using the hypertag information of both forms and trees. Other lexical information are also used. During parsing, the parser uses a left-to-right top-down parsing strategy with bottom-up propagation of the node decorations. The left-corner relation controls the selection of syntactic categories and of trees at a given position in the input sentence.

Several experiments on test suites and corpora have shown the strong gains resulting from these op-

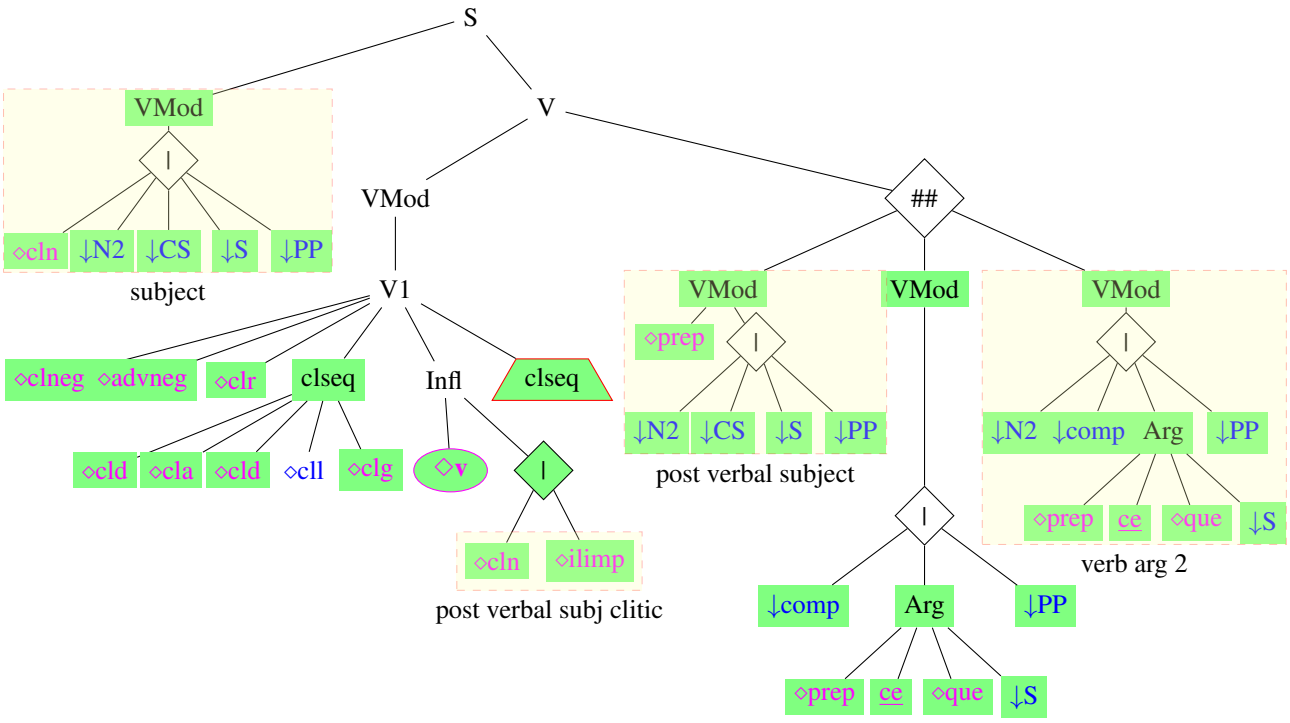


Figure 1: Tree #198 (simplified)

$$
\begin{bmatrix}
\texttt{arg0} & \boxed{a0} & \begin{bmatrix} \texttt{extracted} & - \\ \texttt{kind} & \text{subj} \\ \texttt{pcas} & - \\ \texttt{real} & \boxed{r0}\,\text{-} \mid \text{CS} \mid \text{N2} \mid \text{PP} \mid \text{S} \mid \text{cln} \mid \text{prel} \mid \text{pri} \end{bmatrix} \\
\texttt{arg1} & \boxed{a1} & \begin{bmatrix} \texttt{extracted} & - \\ \texttt{kind} & \boxed{k1}\,\text{-} \mid \text{acomp} \mid \text{obj} \mid \text{prepacomp} \mid \text{prepobj} \\ \texttt{pcas} & \boxed{p1}\,\text{+} \mid \text{-} \mid \text{apres} \mid \text{à} \mid \text{avec} \mid \text{de} \mid \ldots \\ \texttt{real} & \boxed{r1}\,\text{-} \mid \text{CS} \mid \text{N} \mid \text{N2} \mid \text{PP} \mid \text{S} \mid \text{V} \mid \text{adj} \mid \ldots \end{bmatrix} \\
\texttt{arg2} & \boxed{a2} & \begin{bmatrix} \texttt{extracted} & - \\ \texttt{kind} & \boxed{k2}\,\text{-} \mid \text{prepacomp} \mid \text{prepobj} \mid \text{prepscomp} \mid \text{prepv-comp} \mid \text{scomp} \mid \text{vcomp} \mid \text{whcomp} \\ \texttt{pcas} & \boxed{p2}\,\text{+} \mid \text{-} \mid \text{apres} \mid \text{à} \mid \ldots \\ \texttt{real} & \boxed{r2}\,\text{-} \mid \text{CS} \mid \text{N} \mid \text{N2} \mid \text{PP} \mid \text{S} \mid \ldots \end{bmatrix} \\
\texttt{cat} & \text{v} & & \\
\texttt{diathesis} & \text{active} & & \\
\texttt{refl} & \boxed{refl} & &
\end{bmatrix}
$$

(a) for tree #198

$$
\begin{bmatrix}
\texttt{arg0} & \begin{bmatrix} \texttt{kind} & \text{subj} \mid \text{-} \\ \texttt{pcas} & - \end{bmatrix} \\
\texttt{arg1} & \begin{bmatrix} \texttt{kind} & \text{obj} \mid \text{scomp} \mid \text{-} \\ \texttt{pcas} & - \end{bmatrix} \\
\texttt{arg2} & \begin{bmatrix} \texttt{kind} & \text{prepobj} \mid \text{-} \\ \texttt{pcas} & \text{à} \mid \text{-} \end{bmatrix} \\
\texttt{refl} & - &
\end{bmatrix}
$$

(b) for “to promise”

Figure 2: Grammar and lexicon hypertags

timisations, in particular for the left-corner relation (Table 1(b)).

While it would possible to parse tagged sentences, the parser rather takes word lattices as input and returns the shared forest of all possible derivations. When no full parses are found, the parser switches to a robust mode used to return a set of partial parses trying to cover to input sentence in some best way. In post-parsing phases, the derivation forests may be converted to dependency shared forests, which may then be disambiguated.

## 6 Some results

Many efforts have been devoted to improve the meta-grammar and the parser, in terms of accuracy, coverage (in terms of full parses), efficiency and level of ambiguity.

Accuracy has been tested by participating to 3 parsing evaluation campaigns for French, in the context of the French actions EASy and Passage (Paroubek et al., 2008). Table 2 shows the F-measures for 6 kinds of chunks (such as GN, GV, PP, ...) and 14 kinds of dependencies (such as SUBJ-V, OBJ-V, CPL-V, ...) for the two first campaigns[7]. The evolution between the 2 campaigns is clear, with a 3rd position on relations in 2007. The corpus **EasyDev** of the first campaign (almost 4000 sentences) has been used to steadily improve the grammar (and the disambiguation process).

| Campaign | f-measure Chunks | f-measure Relations |
|---|---|---|
| 2004 | 69% | 41% |
| 2007 | 89% | 63% |

Table 2: Results for the French evaluation campaigns

The other parameters (coverage, efficiency and ambiguity) are controlled by regularly parsing several corpora, including the EASy development corpus, as shown in Table 3. The time figures should be taken with some caution, having been computed over various kinds of environments, including laptops, desktops and grid computers[8]. For instance, recent figures computed after porting to 64bit architectures have shown a preliminary speedup by 2.

| Corpus | #sentence | Cov. | time (s) | amb. |
|---|---|---|---|---|
| EUROTRA | 334 | 100% | 0.15 | 0.63 |
| TSNLP | 1161 | 95.07% | 0.07 | 0.46 |
| EasyDev | 3879 | 64.73% | 0.93 | 1.04 |
| JRCacquis | 1.1M | 51.26% | 1.41 | 1.1 |
| Europarl | 0.8M | 70.19% | 1.69 | 1.36 |
| EstRep. | 1.6M | 67.05% | 0.69 | 0.92 |
| Wikipedia | 2.2M | 69.11% | 0.49 | 0.87 |
| Wikisource | 1.5M | 61.08% | 0.71 | 0.89 |
| AFP news | 1.6M | 52.15% | 0.51 | 1.06 |

Table 3: Coverage, avg time, and avg ambiguity

To test the impact of factorization on parsing time, we have partially unfactorized FRMG, keeping all trees but #198. For #198, we have expanded the guards and the shuffle operators (but kept the disjunctions) and then intersected with the 195 verbal subcategoriztion frames present in our LEFFF lexicon. We got a version of FRMG with 5934 trees, 5729 being derived from #198. This grammar was already too large to be able to compile the left-corner relation in reasonable time and space, so Table 4 compares the evaluation times (in seconds) on the 3879 sentences of EasyDev for the unfactorized version and for the factorized version with no left corner. We see that the factorized version (with no left-corner) is slightly faster than the unfactorized one, which a contrario confirms that factorization induces no overhead. Table 4 also shows that FRMG with left-corner is around twice faster than the version with no left-corner. The fact that the left-corner relation can be computed for the factorized version but not easily for the (partially) unfolded one highlights another advantage of the factorization.

| parser | avg | median | 90% | 99% |
|---|---|---|---|---|
| factorized | 0.64 | 0.16 | 1.14 | 6.22 |
| fact. -lc | 1.33 | 0.46 | 2.63 | 12.24 |
| -fact -lc | 1.43 | 0.44 | 2.89 | 14.94 |

Table 4: Factorized vs non-factorized (in seconds)

[7] The results of the last campaign, run over a 100 million word corpus, are not yet available.

[8] including the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr).

## 7 Conclusion

The modularity of MGs combined with tree factoring offers an elegant methodology to design maintainable grammars that remain small in size, opening the way to more efficient parsers, as shown for a large coverage French MG. It should also be noted that the modularity of MGs makes easier the port of a MG for a close language.

The trees that are generated may be very complex but should rather seen as a side-product of simpler linguistic descriptions that are MGs. In other words, the TAGs tend to become an operational target formalism for MGs and the focus is now about improving the MG formalisms to get simpler notations for some constraints (such as node exclusion) and also to incorporate more powerful constraints.

Actually, TAGs as a target formalism are not powerful enough to capture some important syntactic phenomena, for instance deep genitive extractions. A natural evolution would be to use (local) Multi-Components TAGs instead of TAGs, as initiated by (Kallmeyer et al., 2008) for a German grammar. At MG level, the shift is relatively simple, mostly concerning the way the minimal trees are generated.

However, even with MGs, hand-crafting a large coverage grammar remains a complicated and long-standing task. In particular, while the modularity of MGs is a clear advantage for maintenance, tracking the cause of a non-analysis and of some over-generation may be very difficult because hidden in the interactions of several constraints coming from many classes. Besides large regression test suites, there is a need for a sophisticated debugging environment, allowing us to track, at parsing time, the origin of all constraints.

## References


A. Abeillé. 2002. *Une grammaire électronique du français*. CNRS Editions, Paris.

M. A. Alonso and V. J. Díaz. 2003. Variants of mixed parsing of TAG and TIG. *Traitement Automatique des Langues (T.A.L.)*, 44(3):41–65.

M.-H. Candito. 1999. *Organisation modulaire et paramétrable de grammaires électroniques lexicalisées*. Ph.D. thesis, Université Paris 7.

J. Carroll, N. Nicolov, M. Smets, O. Shaumyan, and D. Weir. 1998. Grammar compaction and computation sharing in automata-based parsing. In *Proceedings of Tabulation in Parsing and Deduction (TAPD'98)*, pages 16–25, Paris (FRANCE).

Benoît Crabbé. 2005. *Représentation informatique de grammaires d'arbres fortement lexicalisées : le cas de la grammaire d'arbres adjoints.* Ph.D. thesis, Université Nancy 2.

Ch. Doran, D. Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. 1994. XTAG system — a wide coverage grammar for English. In *Proc. of the 15th International Conference on Computational Linguistics (COLING'94)*, pages 922–928, Kyoto, Japan.

D. Duchier, J. Leroux, and Y. Parmentier. 2004. The metagrammar compiler: An nlp application with a multi-paradigm architecture. In *2nd International Mozart/Oz Conference*.

K. Harbusch and J. Woch. 2004. Integrated natural language generation with schema-tree adjoining grammars. In Christopher Habel and editors Thomas Pechmann, editors, *Language Production*. Mouton De Gruyter.

A. K. Joshi. 1987. An introduction to tree adjoining grammars. In Alexis Manaster-Ramer, editor, *Mathematics of Language*, pages 87–115. John Benjamins Publishing Co., Amsterdam/Philadelphia.

Laura Kallmeyer, Wolfgang Maier, Timm Lichte, Yannick Parmentier, Johannes Dellert, and Kilian Evang. 2008. TuLiPA: Towards a multi-formalism parsing environment for grammar engineering. In *proceedings of the 2nd workshop on Grammar Engineering Across Frameworks (GEAF 2008)*, Manchester, United Kingdom, August.

A. Kinyon. 2000. Hypertags. In *Proc. of COLING*, pages 446–452.

M.-J. Nederhof, G. Satta, and S. Shieber. 2003. Partially ordered multiset context-free grammars and free-word-order parsing. In *In 8th International Workshop on Parsing Technologies (IWPT'03)*, pages 171–182.

Patrick Paroubek, Isabelle Robba, Anne Vilnat, and Christelle Ayache. 2008. Easy, evaluation of parsers of french: what are the results? In *Proceedings of the $6^{th}$ International Conference on Language Resources and Evaluation (LREC)*, Marrakech, Morroco.

Carlos A. Prolo. 2002. Generating the xtag english grammar using metarules. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

Y. Schabes and R. C. Waters. 1995. Tree insertion grammar: a cubic-time, parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Fuzzy Sets Syst.*, 76(3):309–317.