Generating parallel multilingual LFG-TAG grammars from a MetaGrammar

Lionel Clément
Inria-Roquencourt France
lionel.clement@inria.fr

Alexandra Kinyon
CIS Dpt - Univ. of Pennsylvania
kinyon@linc.cis.upenn.edu

Abstract

We introduce a **MetaGrammar**, which allows us to automatically generate, from a single and compact MetaGrammar hierarchy, parallel Lexical Functional Grammars (LFG) and Tree-Adjoining Grammars (TAG) for French and for English: the grammar writer specifies in compact manner syntactic properties that are potentially framework-, and to some extent language-independent (such as subcategorization, valency alternations and realization of syntactic functions), from which **grammars for several frameworks and languages are automatically generated offline**.[1]

1 Introduction

Expensive dedicated tools and resources (e.g. grammars, parsers, lexicons, etc.) have been developed for a variety of grammar formalisms, which all have the same goal: model the syntactic properties of natural language, but resort to a different machinery to achieve that goal. However, there are some core syntactic phenomena on which a cross-framework (and to some extent a cross-language) consensus exists, such as the notions of subcategorization, valency alternations, syntactic function. From a theoretical perspective, a *MetaGrammatical* level of representation allows one to encode such consensual pieces of syntactic knowledge and to compare different frameworks and languages. From a practical perspective, encoding syntactic phenomena at a metagrammatical level, from which grammars for different frameworks and languages are generated offline, has several advantages such as portability among grammatical frameworks, better parallelism, increased coherence and consistency in the grammars generated and less need for human intervention in the grammar development process.

In section 2, we explain the notion of **MetaGrammar** (MG), present the MG tool we use to generate TAGs, and how we extend the approach to generate LFGs. In section 3, we justify the use of a MetaGrammar for generating LFGs and explore several options, i.e. domains of locality, for doing so. In sections 4 and 5, we discus the handling of valency alternations without resorting to LFG lexical rules, and the treatment of long-distance dependencies. In sections 6 and 7, we discuss the advantages of a MG approach and the automatic generation of parallel TAG-LFG grammars for English and for French with an explicit sharing of both cross-language and cross-framework syntactic knowledge in the MG.

2 What is a MetaGrammar ?

The notion of MetaGrammar was originally presented in (Candito, 1996) to automatically generate wide-coverage TAGs for French and Italian[2], using a compact higher-level layer of linguistic description which imposes a general organization for syntactic information in a three-dimensional hierarchy:

- Dimension 1: initial subcategorization
- Dimension 2: valency alternations and redistribution of functions
- Dimension 3: surface realization of arguments.

Each terminal class in dimension 1 encodes an initial subcategorization (i.e. transitive, ditransitive etc...); Each terminal class in dimension 2 - a list of ordered redistributions of functions (e.g. to add an argument for causatives, to erase one for passive with no agents ...); Each terminal class in dimension 3 - the surface realization of a syntactic function (e.g. declares if a direct-object is pronominalized, wh-extracted, etc.). Each class in the hierarchy is associated to the partial description of a tree (Rogers and Vijay-Shanker, 1994) which encodes **father**, **dominance**, **equality** and **precedence** relations between nodes. A well-formed tree is generated by inheriting from exactly one terminal class from dimension 1, one terminal class from dimension 2[3], and n terminal classes from dimension 3 (where n is the number of arguments of the elementary tree being generated). For instance, the elementary tree for "Par qui sera accompagnée Marie" (*By whom will Mary be accompanied*) is generated by inheriting from **transitive** in dimension 1, from **passive** in dimension 2 and **subject-nominal-inverted** for its subject and **Wh-questioned-object** for its object in dimension 3. This particular tool was used to develop from a compact hand-coded hierarchy of a few dozen nodes, a wide-coverage TAG for French of 5000 elementary trees (Abeillé et al., 1999), as well as a medium-size

[1] We assume the reader has a basic knowledge of TAGs and LFGs and refer respectively to (Joshi, 1987) and (Bresnan and Kaplan, 1982) for an introduction to these frameworks.

[2] A Similar MetaGrammar type of organization for TAGs was independently presented in (Xia, 2001) for English.

[3] This terminal class may be the result of the crossing of several super-classes, to handle complex phenomena such as *Passive+Causative*.

TAG for Italian (Candito, 1999). The compactness of the hierarchy is due to the fact that nodes are defined only for **simple** syntactic phenomena: classes for complex syntactic phenomena (e.g. Topicalized-object+Pronominalized) are generated by automatic crossings of classes for simple phenomena. In addition to proposing a compact representation of syntactic knowledge, (Candito, 1999) explored whether some components of the hierarchy could be re-used across similar languages (French and Italian). However, she developed two distinct hierarchies to generate grammars for these two languages and generated only TAG grammars. We extend the use of the MetaGrammar to generate LFGs and also push further its cross-language and cross-framework potential by generating parallel TAGs and LFGs for English and French from one single hierarchy [4].

2.1 HyperTags

The grammar rules we generate are sorted by syntactic phenomena, thanks to the notion of *HyperTag*, introduced in (Kinyon, 2000). The main idea behind HyperTags is to keep track, when trees (i.e. grammar rules) are generated from a MetaGrammar hierarchy, of which terminal classes were used for generating the tree. This allows one to obtain a framework-independent feature structure containing the salient syntactic characteristics of each grammar rule[5]. For instance, the verb *give* in *A book was given to Mary* could be assigned the HyperTag:

$$\begin{bmatrix} \text{Subcat} & \text{Ditransitive} \\ \text{Valency alternations} & \text{Passive no Agent} \\ \text{Argument Realization} & \begin{bmatrix} \text{Subject:} & \text{Canonical NP} \\ \text{Object:} & \text{Not realized} \\ \text{By-Phrase:} & \text{Canonical PP} \end{bmatrix} \end{bmatrix}$$

Although we retain the linguistic insights presented in (Candito, 1996), that is the three dimensions to model syntax, (subcategorization, valency alternation, realization of syntactic arguments), we slightly alter it, and add sub-dimensions for the realization of predicates as well as modifiers. Moreover, we use a different MetaGrammar tool which is less framework-dependent and supports the notion of HyperTag.

2.2 The LORIA MetaGrammar tool

To generate TAGs and LFGs, we use the MG compiler presented in (Gaiffe et al., 2002)[6]. Each class in the MG hierarchy encodes:

- Its SuperClasse(s)
- A HyperTag which captures the salient linguistic characteristics of that class.
- What the class needs and provides.
- A set of quasi-nodes (i.e. variables)
- Topological relations between these nodes (*father, dominates, precedes, equals*)[7]
- A function for each quasi-nodes to decorate the tree (e.g. traditional agreement features and/or LFG functional equations).

The MG tool automatically crosses the nodes in the hierarchy, looking to create "balanced" classes, that is classes that do not need nor provide any resource[8]. Then for each balanced terminal class, the HyperTags are unified, and the structural constraints between quasi-nodes are unified; If the unification succeeds, one or more <HyperTag, tree> pairs are generated. When generating a TAG, $tree$ is interpreted as a TAG elementary tree (i.e. a grammar rule). When generating an LFG, $tree$ is a tree decorated with traditional LFG functional annotations (in a way which is similar to constituent trees decorated with functional annotation e.g. by (Frank, 2000)), and is in a second step broken down into one or more LFG rules. Figure 1 illustrates how a simple decorated tree is generated with the MG compiler, and how the decorated tree corresponds to one TAG elementary tree and to two LFG rewriting rules for a canonical transitive construction. In addition, to facilitate the grammar-lexicon interface, each decorated tree yields an LFG lexical template (here, SubjObj:V (↑Pred='x<(↑Subj)(↑Obj)>').

3 Why use a MetaGrammar for LFGs

3.1 Redundancies in LFG

Because TAGs are a tree rewriting system, there are intrinsic redundancies in the rules of a TAG. E.g., all the rules for verbs with a canonical NP subject and a canonical realization of the verb will have a redundant piece of structure **(S NP0↓ (VP (V◇)))** . This piece of structure will be present not only for each new subcategorization frame (intransitive, transitive, ditransitive...), but also for all related non-canonical syntactic constructions such as in each grammar rule encoding a Wh-extracted object. This redundancy justifies the use of a MetaGrammar for TAGs. Since LFG rules rely on a context free backbone, it is generally admitted that there is less redundancy in LFG than in TAG. However, there are still redundancies, at the level of rewriting rules, at the level of functional equations, and at the level of lexical entries. To illustrate such redundancies, we take the example of French ditransitives with the insertion of one or more modifiers. The direct object is realized as an NP, the second object as a PP. Both orders *NP PP* and *PP NP* are acceptable. On top of that, one or more modifiers may be inserted before, after or between the two arguments, and can be of almost any category (PP, ADVP,

[4] We also generate Range Concatenation Grammars (Boullier, 1998), but do not develop this point here.

[5] The notion of HyperTag was inspired by that of supertags (Srinivas, 1997), which consists in assigning a TAG elementary tree to lexical items, hence enriching traditional POS tagging. However, HyperTags are framework-independent.

[6] This compiler is freely available on http://www.loria.fr/equipes/led/outils/mgc/mgc.html

[7] We have augmented the tool to support free variables for nodes, optional resources, as well as additional relations such as sister and c-command. We do not detail these technical points for sake of brevity.

[8] Another way to see this is by analogy to a resource allocation graph.

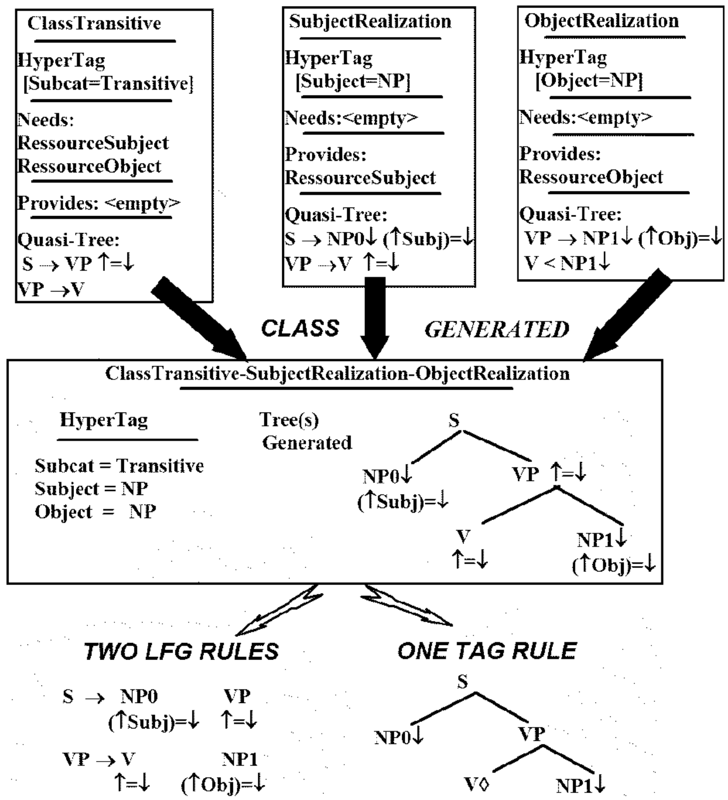


Figure 1: A simple hierarchy which yields one decorated tree, corresponding to one TAG rule and two LFG rules ($\rightarrow$ stands for father, $<$ for precedes in the MG hierarchy. $\diamond$ $\downarrow$ resp. stand for "anchor" and substitution nodes in TAGs. $\downarrow$ and $\uparrow$ stand for standard LFGs functional equations.

NP etc.). Here is a non exhaustive list of acceptable word-order variations:

- Jean donne une pomme à Marie (lit: J. gives an apple to M.)
- Jean donne à Marie une pomme (lit: J. gives to M. an apple)
- Jean aujourd'hui donne à Marie une pomme (lit: J. today gives to M. an apple)
- Jean donne à Marie chaque matin une pomme avant le départ du train (lit: J gives to M. every morning an apple before the departure of the train)
- Jean donne chaque matin à Marie une pomme (lit: J. gives each morning to M. an apple)
- Aujourd'hui Jean donne à Marie une pomme (lit: Today J. gives to M. an apple)

A first rule for VP expansion, accounting for the free order between the first and second object without modifiers, is shown below:

$$\begin{array}{lllll} VP \rightarrow & V & (NP) & PP & (NP) \\ & \uparrow=\downarrow & (\uparrow Obj)=\downarrow & (\uparrow SecondObj)=\downarrow & (\uparrow Obj)=\downarrow \end{array}$$

This VP rule is redundant: the NP is mentioned twice, with its associated functional equation. The NPs are both marked optional because at least one of them has to be not realized, else no well-formed F-structure could be built since the uniqueness condition would be violated by the presence of two direct-objects: for a sentence such as "*Jean donne une pomme à Mary une pomme"/*J. gives an apple to M. an apple*, a C-structure would be built but, as expected, no corresponding well-formed F-structure. Let us now enrich the rule to account for modifier insertion. This yields the VP expansion shown in 2(a).

The rule for VP expansion is now highly redundant, although the syntactic phenomena handled by this rule are very simple ones: the NP for the direct object is repeated twice, along with its functional equation, the disjunction (ADVP|NP|PP) is repeated 5 times, again with its functional equation. This gives us grounds to support a MetaGrammar type of organization for LFG. In practice, as described in (Kaplan and Maxwell, 1996), additional LFG notation is available such as operators like "insert or ignore", "shuffle" "ID/LP", "Macros" etc. However, these operators, which are motivated from a formal perspective, but not so much from a linguistic perspective, yield two major problems: first, not all LFG parsers support those additional operators. Second, the proliferation of operators allows for a same rule to be expressed in many different ways, which is helpful for grammar writing purpose, but not so desirable for maintenance purpose [9]. Although nothing pre-

[9] This can be compared to computer programs written in Perl, which are easy to develop, but hard to read and maintain. A

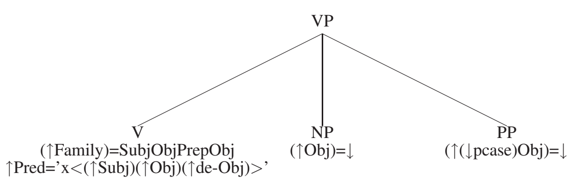
<table>
<tr><td>(a) VP</td><td>→</td><td>(ADVP|NP|PP)*
(↑Modif) ∋↓</td><td>V
↑=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td>(NP)
(↑Obj)=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td>PP
(↑SecObj)=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td>(NP)
(↑Obj)=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td></tr>
<tr><td>(b) VP</td><td>→</td><td>(ADVP|NP|PP)*
(↑Modif) ∋↓</td><td>V
↑=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td>NP
(↑Obj)=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td>PP
(↑SecObj)=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td></td><td></td></tr>
<tr><td>(c) VP</td><td>→</td><td>(ADVP|NP|PP)*
(↑Modif) ∋↓</td><td>V
↑=↓</td><td></td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td></td><td>PP
(↑SecObj)=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td><td>NP
(↑Obj)=↓</td><td>(ADVP|NP|PP)*
(↑Modif)∋↓</td></tr>
</table>

Figure 2: VP expansion

vents the MG generator to create rules with operators such as "ignore or insert", we chose not to do so. Instead of generating rules with operators or rules like (2a), we generate two rules (2b) and (2c) in order to have uniqueness, completeness and coherence not only at the F-structure level but also at the C-structure level.[10]. Moreover, for lexical organization, practical LFGs resort to the notion of lexical template but from a linguistic perspective, the lexicon is not cleanly organized in LFG[11].

3.2 Exploring different domains of locality

We have seen in section 2.2 that the MG tool we use outputs <HyperTag, tree> pairs, where $tree$ is decorated with functional equations and corresponds to one or more LFG rewriting rules (Figure 1).



VP → V PP N2
↑=↓ (↑(↓ pcase)Obj)=↓ (↑ object)=↓

SubjObjectPrepObject:V
(↑ pred = 'x <(↑ Subj) (↑ Obj) (↑ de-Obj)>'

Figure 3: LFG Rule and a lexical entry

In order to generate LFG rules with a MG, we have two options. The first option consists in generating "standard" LFG rules, that is trees of depth 1 decorated with functional equations. Figure 3 illustrates such as decorated tree, which yields one LFG rewriting rule, and one lexical entry for French verbs such as "éloigner" (*take away from*), which take an NP object and a PP object introduced by "de". (Ex: "Peter éloigne son enfant de la fenêtre"/ *P. takes his child away from the window*). The second option, which is the one we have opted for, consists in generating constituent trees which may be of depth superior to one, decorated with feature equations. It has the following advantages:

- It allows for a more natural parallelism between the TAG and LFG grammars generated
- It allows for a more natural encoding of syntax at the MetaGrammar level
- It allows us to generate LFGs without Lexical Rules
- It allows us to easily handle long-distance dependencies.

The trees decorated with LFG functional annotations are then decomposed into standard LFG rewriting rules and lexical entries[12]. The grammar we obtain is then interfaced with a parser [13]. Concerning the first point (TAG-LFG parallelism), the trees decorated with functional equations and TAG elementary trees are very similar, as was first discussed in (Kameyama, 1986). Concerning the second point (more natural encoding of the MetaGrammar level), the "resource model" of the MetaGrammar, based on "needs" and "provides", allows for a natural encoding and enforcement of LFG coherence, completeness and uniqueness principles: A transitive verb needs exactly one resource "Subject" and one resource "Object". Violations result in invalid classes which do not yield any rules. So from that perspective, it makes little sense, apart from practical reasons such as interfacing the grammar with an existing parser, to force the rules generated to be trees of depth one. Moreover, classical completeness/coherence

detailed discussion of the (Kaplan and Maxwell, 1996) operators is found in (Clément and Kinyon, 2003).

[10]Thus the grammars we generate exhibit redundancies for modifiers, but, since the MG hierarchy has relatively few redundancies, and since these grammars are automatically generated, the problem is minor.

[11]As opposed for instance to lexical organization not only in TAGs and TAG related framework (e.g. DATR (Evans et al., 2000)), but in HPSG (Flickinger, 1987).

[12]Non terminal symbols symbols are renamed and, in a second phase, rules which differ only by the name of their non terminals are merged, in a manner similar to that used in (Hepple and van Genabith, 2000). For space reasons, we do not detail the algorithm here.

[13]We use the freely available XLFG parser described in (Clément and Kinyon, 2001) and have also experimented with the Xerox parser (Kaplan and Maxwell, 1996).

conditions have received a similar resource-sensitive re-interpretation in LFG to compute semantic structures using linear logic (Dalrymple et al., 1995). We devote the next two sections to the third (lexical rules) and fourth (wh) points.

4 Lexical rules

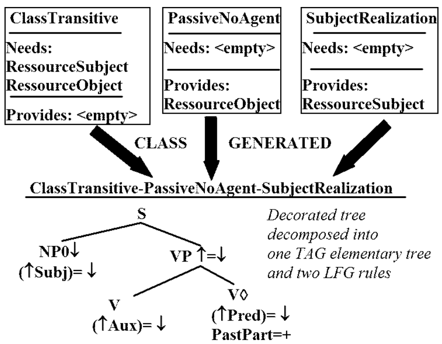


Figure 4: An alternative to lexical rules

Traditional LFGs encode phrase structure realizations of syntactic functions such as the wh-extraction or pronominalization of an object in phrase structure rules. In the MetaGrammar, these are encoded in the "Argument Realization" dimension (dimension 3 in Candito's terminology). For valency alternations, i.e. when initial syntactic functions are modified, LFG resorts to the additional machinery of lexical rules [14]. However, these valency alternations are encoded directly in the MetaGrammar in the "valency alternation" dimension (dimension 2 in Candito's terminology). Hence, when a rule is generated for a canonical transitive verb, rules are generated not only for all possible argument realization for the subject and direct object (wh-questioned, relativized, cliticized for French etc.), but also for all the valency alternations allowed for the subcategory frame concerned (here, passive with/without agent, causative etc). Therefore, there is no need to generate usual LFG lexical rules, and the absence of lexical rules has no effect on interfacing the grammars we generate with existing LFG parsers. Fig. 4 illustrates the generation of a decorated tree for passive-with-no-agent.

5 Long distance dependencies

When generating TAGs and LFGs from a single MG hierarchy, we must make sure that long-distance phenomena are correctly handled. The only difference between TAG and LFG is that for TAG, we must make sure that bridge verbs are auxiliary trees, i.e. have a foot node, whereas for LFG we must make sure that extraction rules have a node decorated with a functional uncertainty equation. In TAGs, long

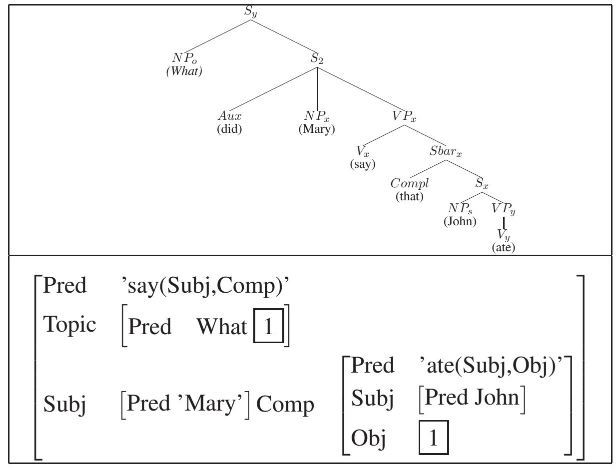


Figure 6: Long distance dependencies in LFG: C and F structures for *What did M. say that J. ate*

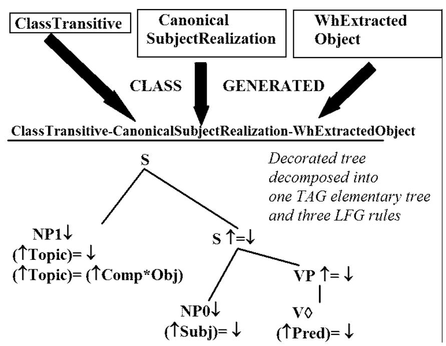


Figure 7: Tree decorated with f. uncertainty

distance dependencies are handled through the domain of locality of elementary trees, the argument-predicate co-occurrence principle and the adjunction operation (Joshi and Vijay-Shanker, 1989). Figure 5 illustrates the TAG analysis of *What did Mary say that John ate*: the extracted element is in the same grammar rule as its predicate "ate" [15] and the tree anchored by the bridge verb is inserted in the "ate" tree thanks to the adjunction operation. More trees can adjoin in to analyze *What does P. think that M. said ... that John ate* using the same mechanism, which we retain in the TAGs we generate by generating auxiliary tree for bridge verbs (i.e. trees with a foot node). In LFG, long-distance dependencies are handled by functional uncertainty (Kaplan and Zaenen, 1989). Here is a small LFG grammar to analyze *What did M. say that John ate*.

[14] Or, alternatively, some notion of lexical mapping, which we do not discuss here.

[15] Although a trace is present in rule for "ate", following the convention of the Xtag project, it is not compulsory and not needed from a formal point of view.

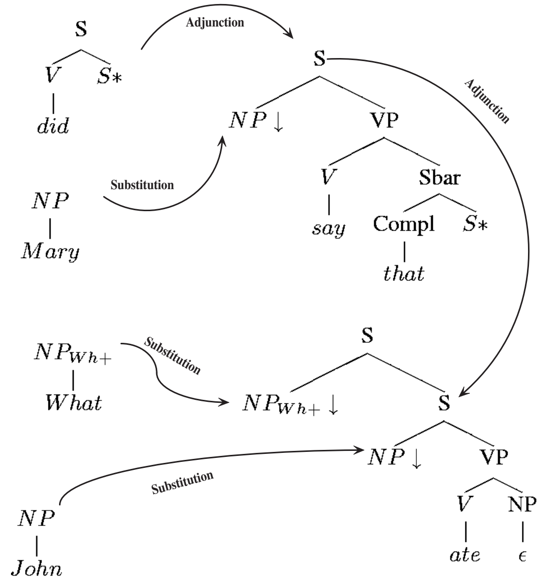


Figure 5: Long distance dependencies in TAGs (*What did M. say that J. ate*)

1- S_x	$\rightarrow$ Aux	NP_x $(\uparrow$Subj$)=\downarrow$	VP_x $\uparrow=\downarrow$
2- VP_x	$\rightarrow$ V_x $\uparrow=\downarrow$	$Sbar_x$ $(\uparrow$Comp$)=\downarrow$	
3- $Sbar_x$	$\rightarrow$ Compl	S_x $\uparrow=\downarrow$	
4- S_y	$\rightarrow$ **NP_o** **$(\uparrow$topic$)=\downarrow$** **$(\uparrow$topic$)=(\uparrow$Comp*.Obj$)$**	S_2 $\uparrow=\downarrow$	
5- S_2	$\rightarrow$ NP_s $(\uparrow$Subj$)=\downarrow$	VP_y $\uparrow=\downarrow$	
6- VP_y	$\rightarrow$ V_y $\uparrow=\downarrow$		

The extracted element (node NP_o in rule 4) is associated to a function path (in bold characters), which is unknown since an arbitrary number of clauses can appear between "NP_o" and its regent (V_y in rule 6). The result of the LFG analysis for *What did M. say that J. ate*, using this standard LFG grammar is shown in Figure 6. A constituent structure is built using the the rewriting rules. The functional equations associated to nodes compute an F-structure which ensures that each predicate of the sentence (i.e. "say" and "ate") have their arguments realized. The need for functional uncertainty results from the fact that in LFG, contrary to TAGs, the extracted element (NP_o) and its governor (V_y) are located in different grammar rules. Hence, when generating LFGs, we must make sure that the decorated tree bears a functional uncertainty equation at the site of the extraction. 7 illustrates the generation of such a decorated tree (identical to the TAG tree for "ate" modulo the functional equations), which will be decomposed into rules 4, 5 and 6.[16]

[16]Because the MG does not impose a restricted domain of locality, (Kinyon, 2003) proposes an alternative to functional uncertainty, which we do not present here for space reasons.

6 Advantages of a MetaGrammatical level

A first advantage of using a MetaGrammar, discussed in (Kinyon and Prolo, 2002), is that the syntactic phenomena covered are quite systematic: if rules are generated for "transitive-passive-whExtractedByPhrase" (e.g. *By whom was the mouse eaten*), and if the hierarchy includes ditransitive verbs, then the automatic crossing of phenomena ensures that sentences will be generated for "ditransitive-passive-whExtractedByPhrase" (i.e. *By whom was Peter given a present*). All rules for word order variations are automatically generated by underspecifying relations between quasi-nodes in the MG hierarchy (e.g. precedence relation between first and second object for ditransitives in French). A second advantage of the MG is to minimize the need for human intervention in the grammar development process. Humans encode the linguistic knowledge in a compact manner i.e. the MG hierarchy, and then verify the validity of the rules generated. If some grammar rules are missing or incorrect, then changes are made directly in the MG hierarchy and never in the generated rules[17]. This ensures a homogeneity not necessarily present with traditional hand-crafted grammars. A third and essential advantage is that it is straightforward to obtain from a single hierarchy parallel multi-lingual grammars similar to the parallel LFG grammars presented in (Butt et al., 1999) and (Butt et al., 2002), but with an explicit sharing

[17]Exceptionality is handled in the MG hierarchy as well. We do not have much to say about it: only that the MG does not impose any additional burden to handle syntactic "exceptions" compared to hand-crafted grammars.

of classes [18] in the MetaGrammar hierarchy plus a cross-framework application. [19]

7 Cross-language and -framework generation

So far, we have implemented a non trivial hierarchy which consists of 189 classes. A fragment of the hierarchy is shown in Figure 8. From this hierarchy, we generate 550 decorated trees, which correspond to approx. 550 TAG trees and 140 LFG rules. We cover the following syntactic phenomena: 50 verb subcategorization frames (including auxiliaries, modals, sentential and infinitival complements), dative-shift for English, clitics (and their placement) for French, passives with and without agent, long distance dependencies (relatives, wh-questions, clefts) and a few idiomatic expressions. A more detailed presentation of the LFG grammar is presented in (Clément and Kinyon, 2003). A more detailed discussion of the cross-language aspects with a comparison to related work such as the LFG ParGram project, or HPSG matrix grammars (Bender et al., 2002) may be found in (Kinyon and Rambow, 2003a)[20]. The cross-language and cross-framework parallelism is insured by the HyperTags: Most classes in the hierarchy are shared for French and for English. Language specific classes are marked using the binary features “English” and “French” in their HyperTag. So for instance, classes encoding clitic placement are marked [French=+;English=-] and classes pertaining to dative-shift are marked [French=-;English=+]. This prevents the crossing of incompatible classes and hence the generation of incorrect rules (such as “Dative-shift-withCliticizedObject”). Similarly, most classes in the hierarchy are shared for TAGs and LFGs. Classes specific to TAGs are marked [TAG=+;LFG=-] (and conversely for LFGs)[21]

8 Conclusion

We have presented a MetaGrammar tool which allows us to automatically generate parallel TAG and LFG grammars for English and French. We have discussed the handling of long-distance dependencies. We keep enriching our hierarchy in order to increase the coverage of our grammars, are adding new languages (German) and exploring the extension of the domain of locality to sentence level (Kinyon and Rambow, 2003a). The ultimate goal of this work is twofold: first, to maximize cross-language rule-sharing at the metagrammatical level; Second, to automatic extract MetaGrammars from a treebank (Kinyon, 2003), and then automatically generate grammars for different frameworks.

[18]To the best of our knowledge, (Butt et al., 2002) apply similar linguistic choices for grammars in different languages when possible, but do not explicitly resort to rule-sharing.

[19](Kinyon and Rambow, 2003b) have used the tool to generate from a single hierarchy cross-framework and cross-language annotated test-suites, including English and German sentences annotated for F-structure, as well as for constituent and dependency structure

[20]The main difference with HPSG approaches such as Matrix is that HPSG type-hierarchies are an inherent part of the grammar, and deal only with one framework:HPSG, whereas our MG hierarchy is not an inherent part of the grammar, since it is used to generate cross-framework grammars offline.

[21]We use binary features in order to add more languages and frameworks to the hierarchy. E.g. when adding German, some classes are shared for English and German, but not French and are marked [English=+;German=+;French=-]. This would not be possible if we had a non binary feature [Language=X]. The same reasoning applies for generating additional frameworks.

References

A. Abeillé, M. Candito, and A. Kinyon. 1999. FTAG: current status and parsing scheme. In *Proc. Vextal-99*, Venice.

E. Bender, D. Flickinger, and S. Oepen. 2002. The Grammar Matrix: an open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proc. GEE-COLING*, Taipei.

P. Boullier. 1998. Proposal for a natural language processing syntactic backbone. Technical report, Inria. France.

J. Bresnan and R. Kaplan. 1982. Introduction: grammars as mental representations of language. In *The Mental Representation of Grammatical Relations*, pages xvii–lii. MIT Press, Cambridge, MA.

M. Butt, S. Dipper, A. Frank, and T. Holloway-King. 1999. Writing large-scale parallel grammars for English, French, and German. In *Proc. LFG-99*.

M. Butt, H. Dyvik, T.H. King, H. Masuichi, and C. Rohrer. 2002. The parallel grammar project. In *proc. GEE-COLING*, Taipei.

M.H. Candito. 1996. A principle-based hierarchical representation of LTAGs. In *Proc. COLING-96*, Copenhagen.

M.H. Candito. 1999. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées*. Ph.D. thesis, Univ. Paris 7.

L. Clément and A. Kinyon. 2003. Generating LFGs with a MetaGrammar. In *Proc. LFG-03*, Saratoga Springs.

L. Clément and A. Kinyon. 2001. XLFG: an LFG parsing scheme for french. In *Proc LFG-01*, Hong-Kong.

M. Dalrymple, J. Lamping, F. Pereira, and V. Saraswat. 1995. Linear logic for meaning assembly. In *Proc. CLNLP*, Edinburgh.

R. Evans, G. Gazdar, and D. Weir. 2000. Lexical rules are just lexical rules. In Abeille Rambow, editor, *Tree Adjoining Grammars*, CSLI.

D. Flickinger. 1987. *Lexical rules in the hierarchical lexicon*. Ph.D. thesis, Stanford.

A. Frank. 2000. Automatic F-Structure annotation of treebank trees. In *Proc. LFG-00*, Berkeley.

B. Gaiffe, B. Crabbé, and A. Roussanaly. 2002. A new metagrammar compiler. In *Proc. TAG+6*, Venice.

M. Hepple and J. van Genabith. 2000. Experiments in structure preserving grammar compaction. In *Proc. 1st meeting on Speech Technology Transfer*, Sevilla.

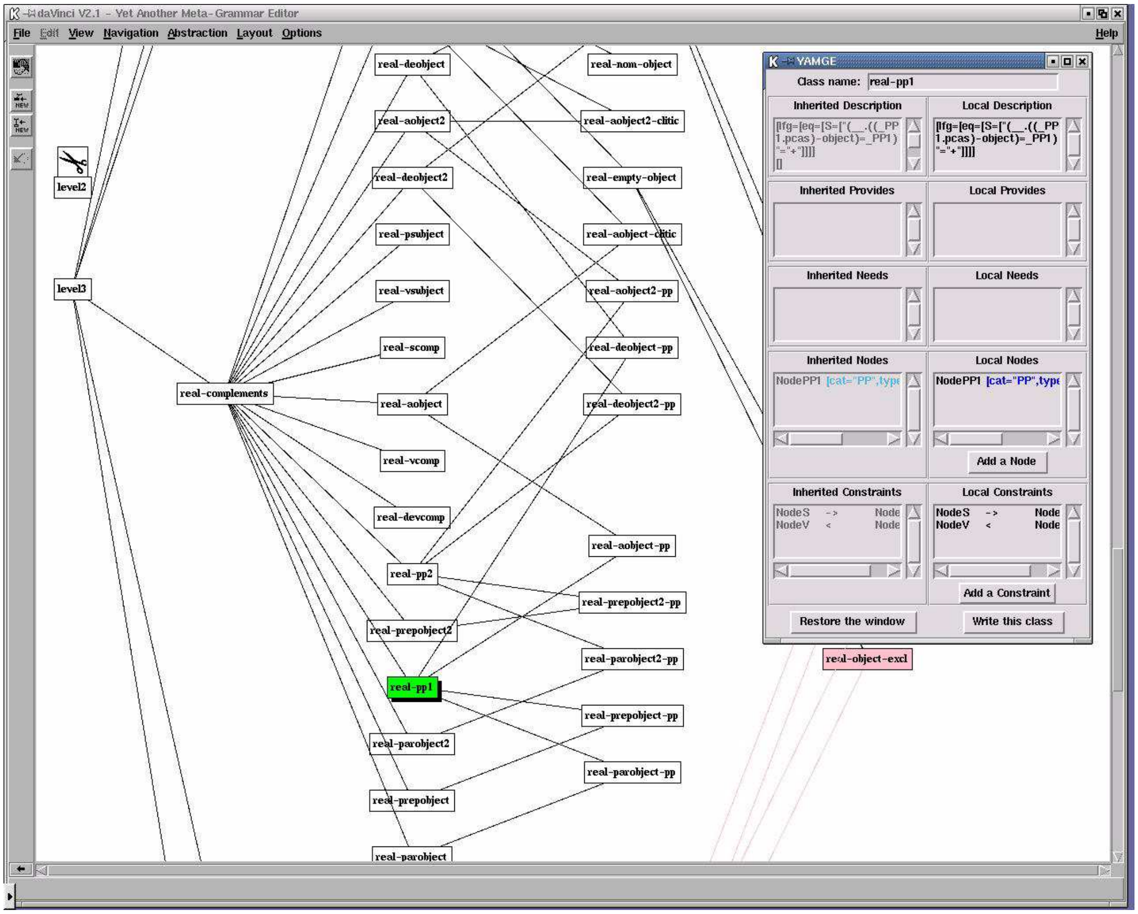


Figure 8: Screen capture of a fragment of our MetaGrammar hierarchy

A. K. Joshi and K. Vijay-Shanker. 1989. Treatment of long distance dependencies in LFG and TAG: Functional uncertainty in LFG is a corollary in TAG. In *Proc. ACL-89*, Vancouver.

A.K. Joshi. 1987. An introduction to tree adjoining grammars. In *Mathematics of language*, John Benjamins Publishing Company.

M. Kameyama. 1986. Characterising LFG in terms of TAG. In *Unpublished Manuscript*, Univ. of Pennsylvania.

R. Kaplan and J. Maxwell. 1996. LFG grammar writer's workbench. Technical Report version 3.1, Xerox corporation.

R Kaplan and A. Zaenen. 1989. Long distance dependencies, constituent structure and functional uncertainty. In *Alternatives conceptions of phrase-structure*, Univ. of Chicago press.

A. Kinyon and C. Prolo. 2002. A classification of grammar development strategies. In *Proc. GEE-COLING*, Taipei.

A. Kinyon and O. Rambow. 2003a. Using the metagrammar for parallel multilingual grammar development and generation. In *Proc. ESSLLI workshop on multilingual grammar engineering*, Vienna.

A. Kinyon and O. Rambow. 2003b. Using the MetaGrammar to generate cross-language and cross-framework annotated test-suites. In *Proc. LINC-EACL*, Budapest.

A. Kinyon. 2000. Hypertags. In *Proc. COLING-00*, Sarrebrucken.

A. Kinyon. 2003. *MetaGrammars for efficient development, extraction and generation of parallel grammars*. Ph.D. thesis, Proposal. Univ. of Pennsylvania.

J. Rogers and K. Vijay-Shanker. 1994. Obtaining trees from their description: an application to TAGS. In *Computational Intelligence 10:4*.

B. Srinivas. 1997. *Complexity of lexical descriptions and its relevance for partial parsing*. Ph.D. thesis, Univ. of Pennsylvania.

F. Xia. 2001. *Automatic grammar generation from two perspectives*. Ph.D. thesis, Univ. of Pennsylvania.