Efficient and robust LFG parsing: SXLFG

Pierre Boullier and Benoît Sagot
INRIA-Rocquencourt, Projet Atoll,
Domaine de Voluceau, Rocquencourt B.P. 105
78 153 Le Chesnay Cedex, France
{pierre.boullier, benoit.sagot}@inria.fr

Abstract

In this paper, we introduce a new parser, called SXLFG, based on the *Lexical-Functional Grammars* formalism (*LFG*). We describe the underlying context-free parser and how functional structures are efficiently computed on top of the CFG shared forest thanks to computation sharing, lazy evaluation, and compact data representation. We then present various error recovery techniques we implemented in order to build a robust parser. Finally, we offer concrete results when SXLFG is used with an existing grammar for French. We show that our parser is both efficient and robust, although the grammar is very ambiguous.

1 Introduction

In order to tackle the algorithmic difficulties of parsers when applied to real-life corpora, it is nowadays usual to apply robust and efficient methods such as Markovian techniques or finite automata. These methods are perfectly suited for a large number of applications that do not rely on a complex representation of the sentence. However, the descriptive expressivity of resulting analyses is far below what is needed to represent, e.g., phrases or long-distance dependencies in a way that is consistent with serious linguistic definitions of these concepts. For this reason, we designed a parser that is compatible with a linguistic theory, namely LFG, as well as robust and efficient despite the high variability of language production.

Developing a new parser for LFG (*Lexical-Functional Grammars*, see, e.g., (Kaplan, 1989)) is not in itself very original. Several LFG parsers already exist, including those of (Andrews, 1990) or (Briffault et al., 1997). However, the most famous LFG system is undoubtedly the Xerox Linguistics Environment (XLE) project which is the successor of the Grammars Writer's Workbench (Kaplan and Maxwell, 1994; Riezler et al., 2002; Kaplan et al., 2004). XLE is a large project which concentrates a lot of linguistic and computational technology, relies on a similar point of view on the balance between shallow and deep parsing, and has been successfully used to parse large unrestricted corpora.

Nevertheless, these parsers do not always use in the most extensive way all existing algorithmic techniques of computation sharing and compact information representation that make it possible to write an efficient LFG parser, despite the fact that the LFG formalism, as many other formalisms relying on unification, is NP-hard. Of course our purpose is not to make a new XLE system but to study how robustness and efficiency can be reached in LFG parsing on raw text.

Building constituent structures (*c-structures*) does not raise any particular problem in theory,[1] because they are described in LFG by a context-free grammar (CFG), called *(CF) backbone* in this paper. Indeed, general parsing algorithms for CFGs are well-known (Earley, GLR,...). On the other hand, the efficient construction of functional structures (*f-structures*) is much more problematic. The first choice that a parser designer must face is that of when f-structures are processed: either during CF

[1] In practice, the availability of a *good* parser is sometimes less straightforward.

Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT), pages 1–10,
Vancouver, October 2005.

parsing (interleaved method) or in a second phase (two-pass computation). The second choice is between f-structures evaluation on single individual [sub-]parses ([sub-]trees) or on a complete representation of all parses. We choose to process all phrasal constraints by a CF parser which produces a shared forest[2] of polynomial size in polynomial time. Second, this shared forest is used, as a whole, to decide which functional constraints to process. For ambiguous CF backbones, this two pass computation is more efficient than interleaving phrasal and functional constraints.[3] Another advantage of this two pass vision is that the CF parser may be easily replaced by another one. It may also be replaced by a more powerful parser.[4] We choose to evaluate functional constraints directly on the shared forest since it has been proven (See (Maxwell and Kaplan, 1993)), as one can easily expect, that techniques which evaluate functional constraints on an enumeration of the resulting phrase-structure trees are a computational disaster. This article explores the computation of f-structures directly (without unfolding) on shared forests. We will see how, in some cases, our parser allows to deal with potential combinatorial explosion. Moreover, at all levels, error recovering mechanisms turn our system into a robust parser.

Our parser, called SXLFG, has been evaluated with two large-coverage grammars for French, on corpora of various genres. In the last section of this paper, we present quantitative results of SXLFG using one of these grammars on a general journalistic corpus.

2 The SXLFG parser: plain parsing

This section describes the parsing process for fully grammatical sentences. Error recovery mechanisms, that are used when this is not the case, are described in the next section.

[2]Informally, a shared forest is a structure which can represent a set of parses (even an unbounded number) in a way that shares all common sub-parses.

[3]This fact can be easily understood by considering that functional constraints may be constructed in exponential time on a sub-forest that may well be discarded later on by (future) phrasal constraints.

[4]For example, we next plan to use an RCG backbone (see (Boullier, 2004) for an introduction to RCGs), with the functional constraints being evaluated on the shared forest output by an RCG parser.

2.1 Architecture overview

The core of SXLFG is a general CF parser that processes the CF backbone of the LFG. It is an Earley-like parser that relies on an underlying left-corner automaton and is an evolution of (Boullier, 2003). The set of analyses produced by this parser is represented by a shared parse forest. In fact, this parse forest may itself be seen as a CFG whose productions are instantiated productions of the backbone.[5] The evaluation of the functional equations is performed during a bottom-up left-to-right walk in this forest. A disambiguation module, which discards unselected f-structures, may be invoked on any node of the forest including of course its root node.

The input of the parser is a word lattice (all words being known by the lexicon, including special words representing unknown tokens of the raw text). This lattice is converted by the *lexer* in a lexeme lattice (a lexeme being here a CFG terminal symbol associated with underspecified f-structures).

2.2 The context-free parser

The evolutions of the Earley parser compared to that described in (Boullier, 2003) are of two kinds: it accepts lattices (or DAGs) as input and it has syntactic error recovery mechanisms. This second point will be examined in section 3.1. Dealing with DAGs as input does not require, at least from a theoretical standpoint, considerable changes in the Earley algorithm.[6] Since the Earley parser is guided by a left-corner finite automaton that defines a regular super-set of the CF backbone language, this automaton also deals with DAGs as input (this corresponds to an intersection of two finite automata).

[5]If A is a non-terminal symbol of the backbone, A_{ij} is an instantiated non-terminal symbol if and only if $A_{ij} \overset{+}{\underset{G}{\Rightarrow}} a_{i+1} \dots a_j$ where $w = a_1 \dots a_n$ is the input string and $\overset{+}{\underset{G}{\Rightarrow}}$ the transitive closure of the *derives* relation.

[6]If i is a node of the DAG and if we have a transition on the terminal t to the node j (without any loss in generality, we can suppose that $j > i$) and if the Earley item $[A \rightarrow \alpha.t\beta, k]$ is an element of the table $T[i]$, then we can add to the table $T[j]$ the item $[A \rightarrow \alpha t.\beta, k]$ if it is not already there. One must take care to begin a PREDICTOR phase in a $T[j]$ table only if all Earley phases (PREDICTOR, COMPLETOR and SCANNER) have already been performed in all tables $T[i]$, $i < j$.

2.3 F-Structures computation

As noted in (Kaplan and Bresnan, 1982), if the number of CF parses (c-structures) grows exponentially w.r.t. the length of the input, it takes exponential time to build and check the associated f-structures. Our experience shows that the CF backbone for large LFGs may be highly ambiguous (cf. Section 4). This means that (full) parsing of long sentences would be intractable. Although in CF parsing an exponential (or even unbounded) number of parse trees can be computed and packed in polynomial time in a shared forest, the same result cannot be achieved with f-structures for several reasons.[7] However, this intractable behavior (and many others) may well not occur in practical NL applications, or some techniques (See Section 2.4) may be applied to restrict this combinatorial explosion.

Efficient computation of unification-based structures on a shared forest is still a evolving research field. However, this problem is simplified if structures are monotonic, as is the case in LFG. In such a case the support (i.e., the shared forest) does not need to be modified during the functional equation resolution. If we adopt a bottom-up left-to-right traversal strategy in the shared forest, information in f-structures is cumulated in a synthesized way. This means that the evaluation of a sub-forest[8] is only performed once, even when this sub-forest is shared by several parent nodes. In fact, the effect of a complete functional evaluation is to associate to each node of the parse forest a set of partial f-structures which only depends upon the descendants of that node (excluding its parent or sister nodes).

The result of our LFG parser is the set of (complete and consistent, if possible) *main f-structures* (i.e., the f-structures associated to the root of the shared forest), or, when a partial analysis occurs, the sets of (partial) f-structures which are associated with *maximal* internal nodes). Such sets of (partial or not) f-structures could be factored out in a single f-structure containing disjunctive values, as in XLE. We decided not to use these complex disjunctive values, except for some atomic types, but rather to associate to any (partial) f-structure a unique identifier: two identical f-structures will always have the same identifier throughout the whole process. Experiments (not reported here) show that this strategy is worth using and that the total number of f-structures built during a complete parse remains very tractable, except maybe in some pathological cases.

As in XLE, we use a lazy copying strategy during unification. When two f-structures are unified, we only copy their *common* parts which are needed to check whether these f-structures are unifiable. This restricts the quantity of copies between two daughter nodes to the parts where they interact. Of course, the original daughter f-structures are left untouched (and thus can be reused in another context).

2.4 Internal and global disambiguation

Applications of parsing systems often need a disambiguated result, thus calling for disambiguation techniques to be applied on the ambiguous output of parsers such as SXLFG. In our case, this implies developing disambiguation procedures in order to choose the most likely one(s) amongst the main f-structures. Afterwards, the shared forest is pruned, retaining only c-structures that are compatible with the chosen main f-structure(s).

On the other hand, on any internal node of the forest, a possibly huge number of f-structures may be computed. If nothing is done, these numerous structures may lead to a combinatorial explosion that prevents parsing from terminating in a reasonable time. Therefore, it seems sensible to allow the grammar designer to point out in his or her grammar a set of non-terminal symbols that have a linguistic property of (quasi-)saturation, making it possible to apply on them disambiguation techniques.[9] Hence, some non-terminals of the CF backbone that correspond

[7] As an example, it is possible, in LFG, to define f-structures which encode individual parses. If a polynomial sized shared forest represents an exponential number of parses, the number of different f-structures associated to the root of the shared forest would be that exponential number of parses. In other words, there are cases where no computational sharing of f-structures is possible.

[8] If the CF backbone G is cyclic (i.e., $\exists A$ s.t. $A \overset{+}{\underset{G}{\Rightarrow}} A$), the forest may be a general graph, and not only a DAG. Though our CF parser supports this case, we exclude it in SXLFG. Of course this (little) restriction does not mean that cyclic f-structures are also prohibited. SXLFG does support cyclic f-structures, which can be an elegant way to represent some linguistic relations.

[9] Such an approach is indeed more satisfying than a blind *skimming* that stops the full processing of the sentence whenever the amount of time or memory spent on a sentence exceeds a user-specified limit, replacing it by a partial processing that performs a bounded amount of work on each remaining non-terminal (Riezler et al., 2002; Kaplan et al., 2004).

to linguistically *saturated* phrases may be associated with an ordered list of disambiguation methods, each of these non-terminals having its own list. This allows for swift filtering out on relevant internal nodes of f-structures that could arguably only lead to inconsistent and/or incomplete main f-structures, or that would be discarded later on by applying the same method on the main f-structures. Concomitantly, this leads to a significant improvement of parsing times. This view is a generalization of the classical disambiguation method described above, since the pruning of f-structures (and incidentally of the forest itself) is not reserved any more to the axiom of the CF backbone. We call *global disambiguation* the pruning of the main f-structures, and *internal disambiguation* the same process applied on internal nodes of the forest. It must be noticed that neither disambiguation type necessarily leads to a unique f-structure. *Disambiguation* is merely a shortcut for *partial or total disambiguation*.

Disambiguation methods are generally divided into probabilistic and rule-based techniques. Our parsing architecture allows for implementing both kinds of methods, provided the computations can be performed on f-structures. It allows to associate a weight with all f-structures of a given instantiated non-terminal.[10] Applying a disambiguation rule consists in eliminating of all f-structures that are not optimal according to this rule. Each optional rule is applied in a cascading fashion (one can change the order, or even not apply them at all).

After this disambiguation mechanism on f-structures, the shared forest (that represent c-structures) is filtered out so as to correspond exactly to the f-structure(s) that have been kept. In particular, if the disambiguation is complete (only one f-structure has been kept), this filtering yields in general a unique c-structure (a tree).

[10]See (Kinyon, 2000) for an argumentation on the importance of performing disambiguation on structures such as TAG derivation trees or LFG f-structures and not constituent(-like) structures.

3 Techniques for robust parsing

3.1 Error recovery in the CF parser

The detection of an error in the Earley parser[11] can be caused by two different phenomena: the CF backbone has not a large enough coverage or the input is not in its language. Of course, although the parser cannot make the difference between both causes, parser and grammar developers must deal with them differently. In both cases, the parser has to be able to perform *recovery* so as to resume parsing as well as, if possible, to correctly parse valid portions of incorrect inputs, while preserving a *sensible* relation between these valid portions. Dealing with errors in parsers is a field of research that has been mostly addressed in the deterministic case and rarely in the case of general CF parsers.

We have implemented two recovery strategies in our Earley parser, that are tried one after the other. The first strategy is called *forward recovery*, the second one *backward recovery*.[12] Both generate a shared forest, as in the regular case.

The mechanism is the following. If, at a certain point, the parsing is blocked, we then *jump forward* a certain amount of terminal symbols so as to be able to resume parsing. Formally, in an Earley parser whose input is a DAG, an error is detected when, whatever the *active* table $T[j]$, items of the form $I = [A \rightarrow \alpha.t\beta, i]$ in this table are such that in the DAG there is no out-transition on t from node j. We say that a recovery is possible in k on β if in the suffix $\beta = \beta_1 X \beta_2$ there exists a derived phrase from the symbol X which starts with a terminal symbol r and if there exists a node k in the DAG, $k \geq j$, with an out-transition on r. If it is the case and if this possible recovery is selected, we put the item $[A \rightarrow \alpha t \beta_1 . X \beta_2, i]$ in table $T[k]$. This will ensure

[11]Let us recall here that the Earley algorithm, like the GLR algorithm, has the valid prefix property. This is still true when the input is a DAG.

[12]The combination of these two recovery techniques leads to a more general algorithm than the *skipping* of the GLR* algorithm (Lavie and Tomita, 1993). Indeed, we can not only skip terminals, but in fact replace any invalid prefix by a valid prefix (of a right sentential form) with an increased span. In other words, both terminals and non-terminals may be skipped, inserted or changed, following the heuristics described later on. However, in (Lavie and Tomita, 1993), considering only the skipping of terminal symbols was fully justified since their aim was to parse spontaneous speech, *full of noise and irrelevances that surround the meaningful words of the utterance*.

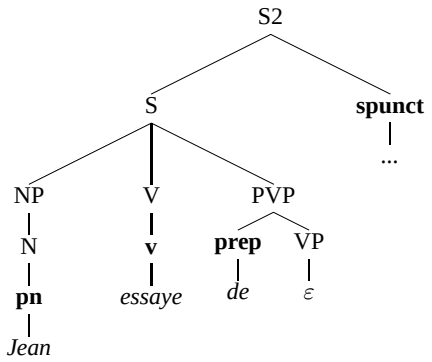

Figure 1: Simplified constituents structure for incomplete sentence *Jean essaye de...* (“*Jean tries to...*”). The derivation of VP in the empty string is the result of a forward recovery, and will lead to an incomplete functional structure (no “pred” in the sub-structure corresponding to node VP).

at least one valid transition from $T[k]$ on r. The effect of such a recovery is to assume that between the nodes j and k in the DAG there is a path that is a phrase generated by $t\beta_1$. We select only nodes k that are as close as possible to j. This *economy principle* allows to skip (without analysis) the smallest possible number of terminal symbols, and leads pretty often to no skipping, thus deriving β_1 into the empty string and producing a recovery in $k = j$. This recovery mechanism allows the parsing process to go forward, hence the name *forward recovery*.

If this strategy fails, we make use of *backward recovery*.[13] Instead of trying to apply the current item, we *jump backward* over terminal symbols that have already been recognized by the current item, until we find its calling items, items on which we try to perform a forward recovery at turn. In case of failure, we can go up recursively until we succeed. Indeed, success is guaranteed, but in the worst case it is obtained only at the axiom. In this extreme case, the shared forest that is produced is only a single production that says that the input DAG is derivable from the axiom. We call this situation *trivial recovery*. Formally, let us come back to the item $I = [A \rightarrow \alpha.t\beta, i]$ of table $T[j]$. We know that there exists in table $T[i]$ an item J of the form $[B \rightarrow \gamma.A\delta, h]$ on which we can hazard a forward recovery in l on δ, where $i \leq j \leq l$. If this fails, we go on coming back further and further in the past, until we reach the initial node of the DAG and the root item $[S' \rightarrow .S\$, 0]$ of table $T[0]$ ($ is the end-of-sentence mark and S' the super-axiom). Since any input ends with an $ mark, this strategy always succeeds, leading in the worst case to trivial recovery.

An example of an analysis produced is shown in Figure 1: in this case, no out-transition on **spunct** is available after having recognized **prep**. Hence a forward recovery is performed that inserts an “empty” VP after the **prep**, so as to build a valid parse.

3.2 Inconsistent or partial f-structures

The computation of f-structures fails if and only if no consistent and complete main f-structure is found. This occurs because unification constraints specified by functional equations could not have been verified or because resulting f-structures are inconsistent or incomplete. Without entering into details, inconsistency mostly occurs because sub-categorization constraints have failed.

$$\left[\begin{array}{l} \text{pred = \textbf{'essayer <subj, de-vcomp>'}, v[2..3]} \\ \\ \text{subj = } \left[\begin{array}{l} \text{pred = \textbf{'Jean <(subj)>'}, pn[1..2]} \\ \text{det = } + \\ \text{hum = } + \\ A_{ij} = \left\{R_9^{182}, R_{26}^{177}, R_{28}^{170}\right\} \end{array}\right]_{F68} \\ \\ \text{de-vcomp = } \left[\begin{array}{l} \text{pred = \textbf{'de <obj|...>'}, prep[3..4]} \\ \text{vcomp = } \left[\begin{array}{l} \text{subj = } []_{F68} \\ A_{ij} = \left\{R_{84}^{162}\right\}_2 \end{array}\right]_{F69} \\ \text{pcase = de} \\ A_{ij} = \{\}_2 \end{array}\right]_{F70} \\ \\ \text{number = sg} \\ \text{person = 3} \\ \text{mode = indicative} \\ \text{tense = present} \\ A_{ij} = \left\{R_{33}^{130}, R_{48}^{119}, R_{49}^{134}\right\} \end{array}\right]$$

Figure 2: Simplified incomplete functional structure for incomplete sentence *Jean essaye de...* (“*Jean tries to...*”). Sub-structure identifiers are indicated as subscripts (like $F70$). In the grammar, a rule can tell the parser to store the current instantiated production in the special field A_{ij} of its associated left-hand side structure. Hence, atoms of the form R_p^q represent instantiated production, thus allowing to link sub-structures to non-terminals of the c-structure.

[13] This second strategy could be also used before or even in parallel with the forward recovery.

A first failure leads to a second evaluation of f-structures on the shared forest, during which consistency and completeness checks are relaxed (an example thereof is given in Figure 2). In case of success, we obtain inconsistent or incomplete main f-structures. Of course, this second attempt can also fail. We then look in the shared forest for a set of *maximal nodes* that have f-structures (possibly incomplete or inconsistent) and whose mother nodes have no f-structures. They correspond to partial disjoint analyses. The disambiguation process presented in section 2.4 applies to all maximal nodes.

3.3 Over-segmentation of unparsable sentences

Despite all these recovery techniques, parsing sometimes fails, and no analysis is produced. This can occur because a time-out given as parameter has expired before the end of the process, or because the Earley parser performed a trivial recovery (because of the insufficient coverage of the grammar, or because the input sentence is simply too far from being correct: grammatical errors, incomplete sentences, too noisy sentences, ...).

For this reason, we developed a layer over SXLFG that performs an *over-segmentation* of ungrammatical sentences. The idea is that it happens frequently that portions of the input sentence are analyzable as sentences, although the full input sentence is not. Therefore, we split in *segments* unparsable sentences (level 1 segmentation); then, if needed, we split anew unparsable level 1 segments[14] (level 2 segmentation), and so on with 5 segmentation levels.[15] Such a technique supposes that the grammar recognizes both chunks (which is linguistically justified, e.g., in order to parse nominal sentences) and isolated terminals (which is linguistically less relevant). In a way, it is a generalization of the use of a FRAGMENT grammar as described in (Riezler et al., 2002; Kaplan et al., 2004).

[14] A sentence can be split into two level 1 segments, the first one being parsable. Then only the second one will be over-segmented anew into level 2 segments. And only unparsable level 2 segments will be over-segmented, and so on.

[15] The last segmentation level segments the input string into isolated terminals, in order to guarantee that any input is parsed, and in particular not to abandon parsing on sentences in which some level 1 or 2 segments are parsable, but in which some parts are only parsable at level 5.

4 Quantitative results

4.1 Grammar, disambiguation rules, lexicon

To evaluate the SXLFG parser, we used our system with a grammar for French that is an adaptation of an LFG grammar originally developed by Clément for his XLFG system (Clément and Kinyon, 2001). In its current state, the grammar has a relatively large coverage. Amongst complex phenomena covered by this grammar are coordinations (without ellipsis), juxtapositions (of sentences or phrases), interrogatives, post-verbal subjects and double subjects (*Pierre dort-il ?*), all kinds of verbal kernels (including clitics, auxiliaries, passive, negation), completives (subcategorized or adjuncts), infinitives (including raising verbs and all three kinds of control verbs), relatives or indirect interrogatives, including when arbitrarily long-distance dependencies are involved. However, comparatives, clefts and elliptical coordinations are not specifically covered, inter alia. Moreover, we have realized that the CF backbone is too ambiguous (see below).

Besides the grammar itself, we developed a set of disambiguation heuristics. Following on this point (Clément and Kinyon, 2001), we use a set of rules that is an adaptation and extension of the three simple principles they describe and that are applied on f-structures, rather than a stochastic model.[16] Our rules are based on linguistic considerations and can filter out functional structures associated to a given node of the forest. This includes two special rules that eliminate inconsistent and incomplete f-structures either in all cases or when consistent and complete structures exist (these rules are not applied during the second pass, if any). As explained above, some non-terminals of the CF backbone, that correspond to linguistically saturated phrases, have been associated with an ordered list of these rules, each of these non-terminal having its own list.[17].

[16] As sketched before, this could be easily done by defining a rule that uses a stochastic model to compute a weight for each f-structure (see e.g., (Miyao and Tsujii, 2002)) and retains only those with the heaviest weights (Riezler et al., 2002; Kaplan et al., 2004). However, our experiments show that structural rules can be discriminative enough to enable efficient parsing, without the need for statistical data that have to be acquired on annotated corpora that are rare and costly, in particular if the considered language is not English.

[17] Our rules, in their order of application on main f-structures, i.e. on the axiom of the backbone, are the following (note that

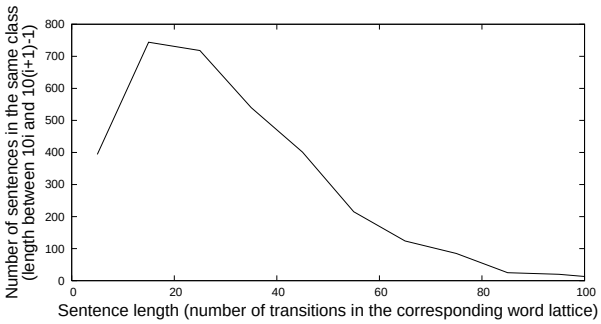

Figure 3: Repartition of sentences of the test corpus w.r.t. their length. We show the cardinal of classes of sentences of length $10i$ to $10(i+1)-1$, plotted with a centered x-coordinate ($10(i+1/2)$).

The lexicon we used is the latest version of Le*fff* (Lexique des formes fléchies du français[18]), which contains morphosyntactic and syntactic information for more than 600,000 entries corresponding to approximately 400,000 different tokens (words or components of multi-word units).

The purpose of this paper is not however to validate the grammar and these disambiguation rules, since the grammar has only the role of enabling evaluation of parsing techniques developed in the current work.

4.2 Results

As for any parser, the evaluation of SXLFG has been carried out by testing it in a real-life situation. We used the previously cited grammar on a raw journalistic corpus of 3293 sentences, not filtered and processed by the SXPipe pre-parsing system described in (Sagot and Boullier, 2005). The repartition of sentences w.r.t. their length is plotted in Figure 3.

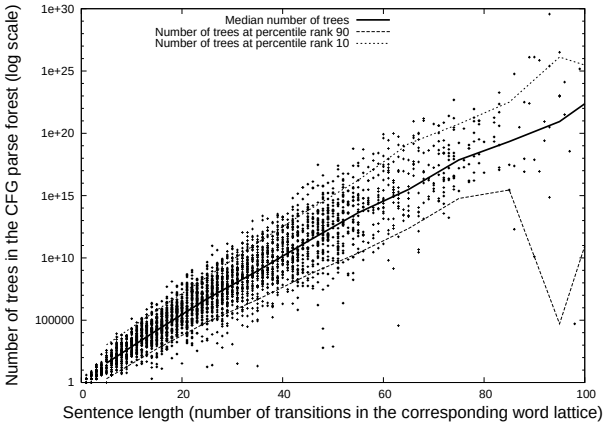

Figure 4: CFG ambiguity (medians are computed on classes of sentences of length $10i$ to $10(i+1)-1$ and plotted with a centered x-coordinate ($10(i+1/2)$).

In all Figures, the x-coordinate is bounded so as to show results only on statistically significant data, although we parse all sentences, the longest one being of length 156.

However, in order to evaluate the performance of our parser, we had to get rid of, as much as possible, the influence of the grammar and the corpus in the quantitative results. Indeed, the performance of the SXLFG parser does not depend on the quality and the ambiguity of the grammar, which is an *input* for SXLFG. On the contrary, our aim is to develop a parser which is as efficient and robust as possible given the input grammar, and in spite of its (possibly huge) ambiguity and of its (possibly poor) intrinsic coverage.

4.2.1 CFG parser evaluation

Therefore, Figure 4 demonstrates the level of ambiguity of the CF backbone by showing the median number of CF parses given the number of transitions in the lattice representing the sentence. Although the number of trees is excessively high, Figure 5 shows the efficiency of our CF parser[19] (the maximum number of trees reached in our corpus is as high as $9.12\,10^{38}$ for a sentence of length 140, which

when used on other non-terminal symbols than the axiom, some rules may not be applied, or in a different order):

Rule 1: *Filter out inconsistent and incomplete structures, if there is at least one consistent and complete structure*.

Rule 2: *Prefer analyses that maximize the sum of the weights of involved lexemes*; amongst lexical entries that have a weight higher than normal are multi-word units.

Rule 3: *Prefer nominal groups with a determiner.*

Rule 4: *Prefer arguments to modifiers, and auxiliary-participle relations to arguments* (the computation is performed recursively on all (sub-)structures).

Rule 5: *Prefer closer arguments* (same remark).

Rule 6: *Prefer deeper structures.*

Rule 7: *Order structures according to the mode of verbs* (we recursively prefer structures with indicative verbs, subjunctive verbs, and so on).

Rule 8: *Order according to the category of adverb governors.*

Rule 9: *Choose one analysis at random* (to guarantee that the output is a unique analysis).

[18] Lexicon of French inflected forms

[19] Our experiments have been performed on a AMD Athlon 2100+ (1.7 GHz).

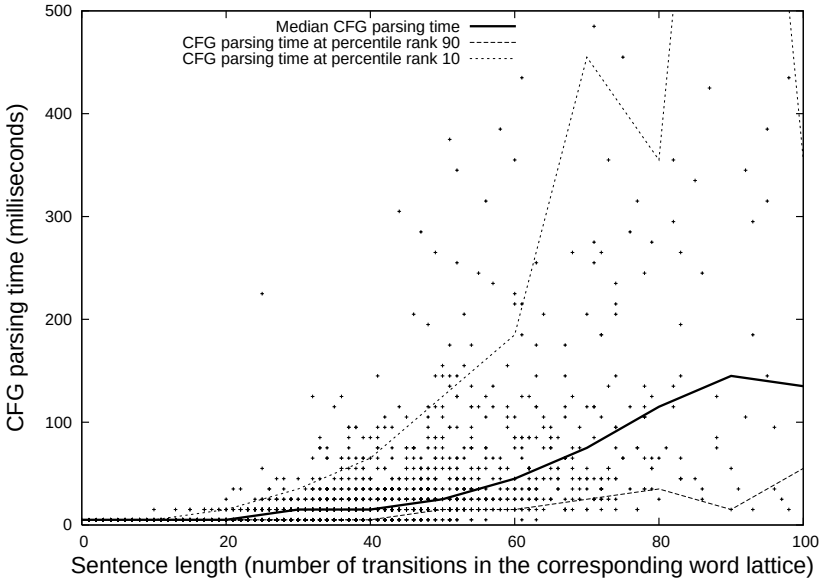

Figure 5: CF parsing time (same remark as for Fig. 4).

is parsed in only 0.75 s). Moreover, the error recovery algorithms described in section 3.1 are successful in most cases where the CF backbone does not recognize the input sentences: out of the 3292 sentences, 364 are not recognized (11.1%), and the parser proposes a non-trivial recovery for all but 13 (96.3%). We shall see later the relevance of the proposed recovered forests. We should however notice that the ambiguity of forests is significantly higher in case of error recovery.

4.2.2 Evaluation of f-structures computation

Although the CF backbone is massively ambiguous, results show that our f-structures evaluation system is pretty efficient. Indeed, with a timeout of 20 seconds, it takes only 6 301 seconds to parse the whole corpus, and only $5,7\%$ of sentences reach the timeout before producing a parse. These results can be compared to the result with the same grammar on the same corpus, but without internal disambiguation (see 2.4), which is 30 490 seconds and 41.2% of sentences reaching the timeout.

The coverage of the grammar on our corpus with internal disambiguation is 57.6%, the coverage being defined as the proportion of sentences for which a consistent and complete main f-structure is output by the parser. This includes cases where the sentence was agrammatical w.r.t. the CF backbone, but for which the forest produced by the error recovery techniques made it possible to compute a consistent and complete main f-structure (this concerns 86 sentences, i.e., 2.6% of all sentences, and 24.5% of all agrammatical sentences w.r.t. the backbone; this shows that CF error recovery gives relevant results).

The comparison with the results with the same grammar but without internal disambiguation is interesting (see Table 1): in this case, the high proportion of sentences that reach the timeout before being parsed leads to a coverage as low as 40.2%. Amid the sentences covered by such a system, 94.6% are also covered by the full-featured parser (with internal disambiguation), which means that only 72 sentences covered by the grammar are lost because of the internal disambiguation. This should be compared with the 645 sentences that are not parsed because of the timeout when internal disambiguation is disabled, but that are covered by the grammar and correctly parsed if internal disambiguation is used: the risk that is taken by pruning f-structures during the parsing process is much smaller than the benefit it gives, both in terms of coverage and parsing time.

Since we do not want the ambiguity of the CF backbone to influence our results, Figure 6 plots the total parsing time, including the evaluation of features structures, against the number of trees produced by the CF parser.

Results	With internal disambiguation		Without internal disambiguation	
Total number of sentences	3293			
Recognized by the backbone	2929		88.9%	
CF parsing with non-trivial recovery	351		10.6%	
CF parsing with trivial recovery	13		0.4%	
Consistent and complete main f-structure	1896	57.6%	1323	40.2%
Inconsistent and incomplete main f-structure	734	22.3%	316	9.6%
Partial f-structures	455	13.8%	278	8.4%
No f-structure	6	0.2%	6	0.2%
No result (trivial recovery)	13	0.4%	13	0.4%
Timeout (20 s)	189	5.7%	1357	40.2%

Table 1: Coverage results with and without internal ranking, with the same grammar and corpus.

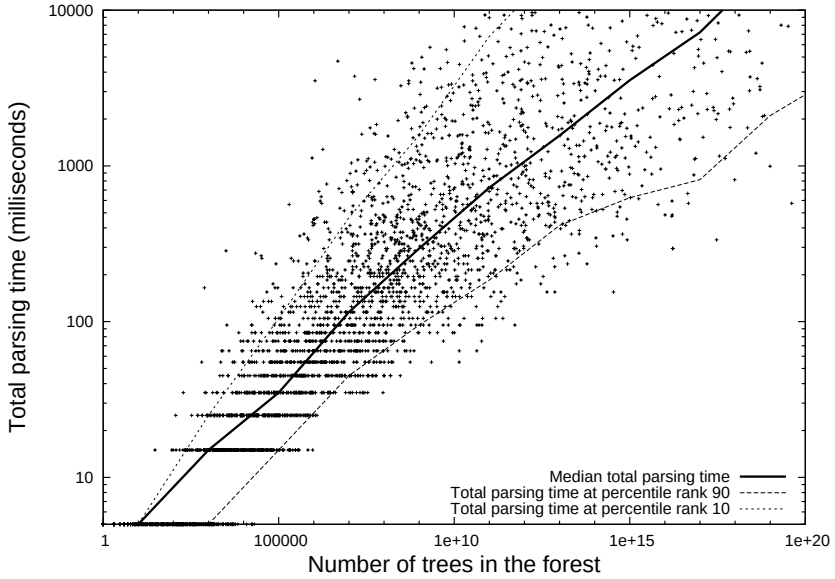

Figure 6: Total parsing time w.r.t. the number of trees in the forest produced by the CF backbone (medians are computed on classes of sentences whose number of trees lies between 10^{2i} and $10^{2i+2} - 1$ and plotted with a centered x-coordinate (10^{2i+1})).

5 Conclusion

This paper shows several important results.

It shows that wide-coverage unification-based grammars can be used to define natural languages and that their parsers can, in practice, analyze raw text.

It shows techniques that allow to compute feature structures efficiently on a massively ambiguous shared forest.

It also shows that error recovery is worth doing both at the phrasal and functional levels. We have shown that a non-negligible portion of input texts that are not in the backbone language can nevertheless, after CF error recovery, be qualified as valid sentences for the functional level.

Moreover, the various robustness techniques that are applied at the functional level allow to gather (partial) useful information. Note that these robust techniques, which do not alter the overall efficiency of SXLFG, apply in the two cases of incomplete grammar (lack of covering) and agrammatical phrases (w.r.t. the current definition), though it seems to be more effective in this latter case.

References

Avery Andrews. 1990. Functional closure in LFG. Technical report, The Australian National University.

Pierre Boullier. 2003. Guided Earley parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT'03)*, pages 43–54, Nancy, France, April.

Pierre Boullier. 2004. Range concatenation grammars. In *New developments in parsing technology*, pages 269–289. Kluwer Academic Publishers.

Xavier Briffault, Karim Chibout, Gérard Sabah, and Jérôme Vapillon. 1997. An object-oriented linguistic engineering environment using LFG (Lexical-Functional Grammar) and CG (Conceptual Graphs). In *Proceedings of Computational Environments for Grammar Development and Linguistic Engineering, ACL'97 Workshop*.

Lionel Clément and Alexandra Kinyon. 2001. XLFG – an LFG parsing scheme for French. In *Proceedings of LFG'01*, Hong Kong.

Ronald Kaplan and Joan Bresnan. 1982. Lexical-functional grammar: a formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.

Ronald M. Kaplan and John T. Maxwell. 1994. Grammar writer's workbench, version 2.0. Technical report, Xerox Corporation.

Ronald Kaplan, Stefan Riezler, Tracey King, John Maxwell, Alex Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of HLT/NAACL*, Boston, Massachusetts.

Ronald Kaplan. 1989. The formal architecture of lexical functionnal grammar. *Journal of Informations Science and Engineering*.

Alexandra Kinyon. 2000. Are structural principles useful for automatic disambiguation ? In *Proceedings of in COGSCI'00*, Philadelphia, Pennsylvania, United States.

Alon Lavie and Masaru Tomita. 1993. GLR* – an efficient noise-skipping parsing algorithm for context-free grammars. In *Proceedings of the Third International Workshop on Parsing Technologies*, pages 123–134, Tilburg, Netherlands and Durbuy, Belgium.

John Maxwell and Ronald Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–589.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of HLT*, San Diego, California.

Stefan Riezler, Tracey King, Ronald Kaplan, Richard Crouch, John Maxwell, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the Annual Meeting of the ACL*, University of Pennsylvania.

Benoît Sagot and Pierre Boullier. 2005. From raw corpus to word lattices: robust pre-parsing processing. In *Proceedings of L&TC 2005*, Poznań, Pologne.