Efficient 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 for French, called SxLfg, based on the *Lexical-Functional Grammars* formalism (*LFG*). We first describe the LFG parser we developed, the underlying CFG 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 present some concrete results when SxLfgis used with an existing grammar for French that we slightly adapted. We show that our parser is both efficient and robust, although the grammar is still improvable.

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 to 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., syntagms or long-distance dependancies in a way that is coherent 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. [1] is not in itself very original. Several LFG parsers already exist, including those of [2], [3], or [4]. However, they 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-complete.

To be able to use these techniques as much as possible, we had to slightly constraint the formalism, without changing its expressive power. Associated with a tabular parser, our variant of the LFG formalism allows us to parse efficiently complex sentences. Building consituency structures does not raise any particular problem in theory,[1] because they are described by context-free grammars (CFGs) underlying LFGs and called *support grammars* in this paper. Indeed, general parsing algorithms for CFGs are well-known (Earley, GLR,...). On the contrary, the efficient construction of the functional structures is much more problematic. We developed a module to compute these structures that shares sub-structures which are common to different analyses. Moreover, error recovering mechanisms at all levels turn our system into a robust parser.

This parser, called SxLFG, has been evaluated with two medium- to large-coverage grammars for French, on various kinds of corpora. 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: regular parsing

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

2.1 Architecture overview

The core of SxLFG is a general context-free parser that deals with the support grammar of the LFG. It is an Earley-like parser that relies on an underlying left-corner automaton and is an evolution of [5]. The set of analyses produced by this parser are is represented by a parse shared forest; i.e., a CFG whose productions are instantiated productions of the support grammar.[2] The evaluation of the functional equations is performed during a bottom-up left-to-right walk in this forest. The input of the parser is a words lattice (all words being known by the lexicon, including special words representing unknown tokens of the raw text). This lattice is converted by the *l*exer in a lexemes lattice (a lexeme being here a CFG terminal symbol associated with associated underspecified functional structures). An optional post-processing allows to disambiguate the parser's output. This architecture is summarized in Figure 1.

2.2 The CFG parser

The evolutions of the Earley parser compared to that described in [5] are of two kinds: it takes into account lattices (or DAGs) as input and has syntactic errors 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 point

---

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

[2] If A is a symbol of the support grammar, A_{ij} is an instantiated symbol if and only if $A_{ij} \underset{w}{\overset{*}{\Rightarrow}} a_{i+1} \dots a_j$ where $w = a_1 \dots a_n$ is the input string and $\underset{w}{\overset{*}{\Rightarrow}}$ the transitive closure of the derivation relation.

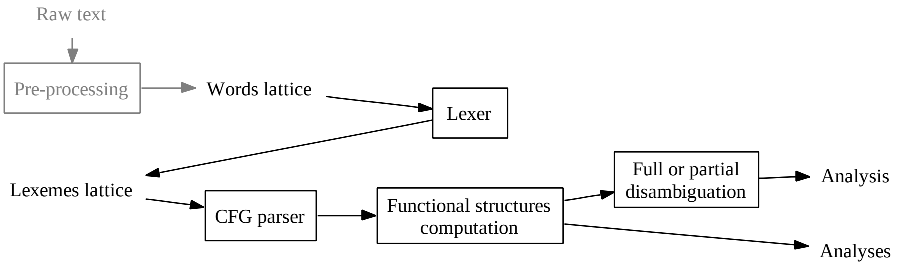

Fig. 1. Overall architecture of SxLfg.

of view, considerable changes in the Earley algorithm.[3] Since the Earley parser is guided by a left-corner finite automaton that defines a regular super-set of the context-free support language, this automaton also deals with DAGs as input (this corresponds to an intersection of two finite automata).

2.3 Calcul des structures fonctionnelles

Disposant d'une forêt partagée en sortie de l'analyse CF[4], nous devons maintenant calculer les structures fonctionnelles. Bien entendu, la méthode qui consiste à *déplier* la forêt pour en extraire chaque arbre sur lequel on évalue les structures fonctionnelles est impratiquable en termes de temps de calcul. En revanche, l'autre possibilité, une évaluation des structures fonctionnelles directement sur la forêt partagée, est toujours un sujet de recherche. Le problème se simplifie cependant si l'on suppose, comme c'est le cas dans SxLfg, que l'information calculée est uniquement synthétisée. Dans ce cas, le support (c'est-à-dire la forêt partagée) n'a pas besoin d'être modifié en cours d'évaluation.

La conséquence directe de cette évaluation bas-haut des structures fonctionnelles est que toute sous-forêt[5] n'est évaluée qu'une seule fois et que son calcul est partagé entre tous ses parents. L'autre conséquence est qu'à chaque nœud de

---

[3] 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. It is of course needed to take care to begin a prediction phase in a $T[j]$ table only if all Earley phases (*prédiction*, *complétion* and *scanning*) have already been performed in all tables $T[i]$, $i < j$.

[4] Rappelons que cette structure permet de représenter en une taille polynomiale en n, longueur du texte source ou nombre d'arcs du DAG, l'ensemble potentiellement non borné des arbres d'analyse.

[5] En toute généralité, chaque (sous-)forêt est un graphe. Nous excluons cependant ce cas et nous ne traitons donc que des (sous-)forêts qui sont des DAG. La grammaire support ne doit donc pas être cyclique Cette (petite) restriction n'empêche pas, bien sûr, les structures fonctionnelles d'être elles-mêmes cycliques.

la forêt est associée non pas une valeur unique mais une disjonction de valeurs que l'on doit savoir gérer. Très souvent, le résultat de ce calcul est donc un grand nombre de structures fonctionnelles (valides) associées à la racine de la forêt.

2.4 Disambiguation

The output of the previous step (if it does not fail, see next section) is a shared forest of constituent structures associated with a set of functional structures that share common sub-structures. In the general case, this is the description of more than one analyses. Therefore, we need to be able to disambiguate, i.e., to chose amongst these analyses the most likely one. Two families of techniques can be thought of: probabilistic techniques and rule-based techniques. Following on this point [6], we use a set of rules that is an adaptation and extension of the three simple principles they describe and that are applied on functional structures.[6] Each rule is applied one after the other (one can change the order, or even not apply them at all). Applying a rule consists in the elimination of all analyses that are not optimal according to this rule.[7].

After this disambiguation mechanism on functional structures, the shared forest (that represent constituent structures) is filtered so as to correspond exaclty to the functional structure(s) that have been kept. In particular, if the disambiguation is complete (only one functional structure has been retained), this filtering returns in general a unique constituent structure (a tree).

---

[6] See [7] for an argumentation on the importance of performing disambiguation on structures such as TAG derivation trees or LFG functional structures and not constituent(-like) structures.

[7] Our rules, in their default order of application, are:

règle 1: *Prefer analyses that maximize the sum of the weights of involved lexemes*; amongst lexical entries that have a wieght higher than normal are multi-word units.

règle 2: *Prefer nominal groups with determiner.*

règle 3: *Prefer arguments to modifiers, and relations auxiliary-participle to arguments* (the computation is performed recursively on all (sub-)structures).

règle 4: *Prefer closer arguments* (same remark).

règle 5: *Prefer deeper structures.*

règle 6: *Order structures according to the mode of verbs* (we recursively prefer structures with indicative verbs, subjunctive verbs, and so on).

règle 7: *Order according to the category of adverb governors.*

règle 8: *Chose randomly one analysis* (to guarantee that the output is a unique analysis).

3 Mécanismes pour l'analyse robuste

3.1 Rattrapage d'erreur dans l'analyseur CFG

The detection of an error in the Earley parser[8] can be caused by two different phenomena: the support grammar has not a large enough coverage and/or the input is not proper French. Of course, although the parser can not make the difference between both situations, parser and grammar developers have to deal with them differently. In both cases, the parser has to be able to perform $r$ecovery so as to go on the parsing and to be able, if possible, to parse correctly valid portions of incorrect inputs, while preserving a $s$ensible relation between these valid portions. Dealing with errors in parsers is a field of reserach that has been mostly adressed 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*. Both generate a shared forest, as in the regular case.

The forward recovery mechanism is the following. If, at a certain point, the parsing is blocked, we then "jump" a certain amount of terminal symbols so as to be able to go on parsing. Formally, in an Earley parser whose input is a DAG, an error is detected when, in active tables $T[j]$, items of the form $I = [A \rightarrow \alpha.t\beta, i]$ are such that in the DAG there is no transition t going out of the node j. We say that a recovery in k is possible on I in table $T[j]$ if in the suffix $\beta = \beta_1 X \beta_2$ there exists a symbol X whose derived phrases begin with a terminal symbol r and if there exists a node k in the DAG, $k \geq j$, that has a 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 $T[k]$, which will ensure at least one valid transition from $T[k]$ on r. This means that we suppose that in the DAG, between the nodes j and k, there is a path that is a phrase generated by $t\beta_1$. This allows the parsing process to go forward rightwards, hence the name *forward recovery*: we try the recovery only on nodes k that are on the right of j ($k \geq j$).

If this fails, we use the second strategy,[9] called *backward recovery* because it can put again in the balance previous choices. Instead trying to go on applying the current production, we "jump" terminals already concerned by this production by going up to the callin production and performing on it a forward recovery: although previously chosen, the left-hand side non-terminal of the production where the error occured is thus eliminated. In case of failure, we can go up again, and so on until we reach 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 exsits in table $T[i]$ an item J of the form $[B \rightarrow \gamma.A\delta, h]$ on which we can try a

---

[8] 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.

[9] Of course, this strategy could be also used before or even in parallel with forward recovery.

forward recovery on l, where $i \leq j \leq l$. If it 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 $T[0]$ (\$ being the end-of-sentence mark and S' the super-axiom). Since any input ends with an end-of-sentence mark, this strategy always succeeds, leading in the worst case to trivial recovery.

3.2 Unconsistent or partial functional structures

The computation of functional structures fails if and only if no functional structure is associated with the root of the shared forest. This occurs because unification constraints specified by functional equations could not have been verified or because resulting functional structures are unconsistent. Without entering into the details, unconsistency mostly occurs because sub-categorization constraints have failed.

A first failure leads to a second evaluation of functional structures on the shared forest, during which consistency checks are not performed. In case of success, we obtain inconsistent functional structures associated with the root of the shared forest. Of course, this second attempt can also fail. We then look in the shared forest for a set of *m*aximal nodes that have functional structures (consistent or not) and whose fathers have no functional structure. They correspond to partial disjoint analyses, possibly inconsistent. The disambigation process presented in section 2.4 applies to all maximal nodes. s'applique à tous les nœuds maximaux.

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 really too far from being correct: grammatical errors, incomplete sentences, noisy sentence,...).

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 analysable 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[10] (level 2 segmentation), and so on with 5 segmentation levels which are the following:

level 1: segmentation on (quasi-)sentence boundaries: this includes sentence boundaries detected by a segmenter if it has not been used before (e.g., because the text was already segmented, although with a different segmenter), or quasi-sentence boundaries (";", list items identifiers,...),

---

[10] 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.

level 2: segmentation on strong punctuations (";", isolated hyphens, exclamation or question marks, quotes, etc.),
level 3: segmentation on weak punctuations (typically ","),
level 4: segmentation on coordinations (such as "*et*" – "*and*" – or "*ou*" – "*or*" –),
level 5: segmentation on word boundaries.

The quality of the resulting analysis decreases of course with the level of segmentation. The level 1 segmentation does not lead to any particular problem, but difficulties appear with level 2. Levels 3 and 4 are really recovery. And level 5 is only here to be able to parse all sentences, 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.[11]

4 Résultats et statistiques

4.1 Grammar

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 L. Clément for his XLFG system [6].

In its current state, the grammar has a medium coverage. Amongst complex phenomena covered by this grammar are coordinations (without ellipsis), juxtapositions (of sentences or phrases), interrogatives (but not clefts), postverbal subjects or 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, coordiations with ellipsis are not covered (amongst others). Moreover, we have realized that the support grammar is too ambiguous (see section 4).

But the purpose of this paper is not to validate the grammar, but the parsing techniques that we developed, the grammar having only the role of enabling evaluation of these techniques.

5 Results

Dans cette section, nous n'évaluerons pas la *qualité* d'une analyse qui dépend pour l'essentiel[12] de la grammaire qui nécessiterait de disposer d'un corpus de référence annoté manuellement. Nous nous concentrons ici sur l'efficacité de notre

---

[11] Of course, level 5 segmentation works correctly only if all terminals are individually part of the language.

[12] Cette qualité dépend aussi des heuristiques de désambiguïsation utilisées et du traitement de la robustesse.

système en présentant les résultats obtenus sur un corpus journalistique de 3 213 phrases.

Puisque chaque phrase est représentée sous forme de DAG, nous donnons une évaluation de sa complexité non en terme de longueur, mais en comptant le nombre d'arcs du DAG. Chaque arc représente une transition sur un mot. Le nombre moyen d'arcs par phrase est de 29,1 avec une médiane à 25 alors que la plus longue phrase contient 144 transitions. Il faut noter que le lexeur, à son tour, transforme chaque mot en un ensemble de symboles terminaux de la grammaire support. Cette ambiguïté lexicale produit elle-même en moyenne une expansion d'un facteur 1,5. Cette expansion peut, pour certains mots courants, dépasser assez largement ce facteur. C'est en particulier le cas des mots inconnus qui produisent chacun 5 catégories différentes. Malgré la qualité du pré-traitement, lexique et correction des erreurs (cf. figure 1), il reste en moyenne 1,5 mot inconnu par phrase avec une médiane à 1 et un maximum à 29 !

Sur les 3 213 phrases, 409 (12,7 %) sont agrammaticales pour la grammaire support (CFG) et déclenchent donc le traitement d'erreur décrit à la section 3.1. Notre grammaire support est excessivement (anormalement !) ambiguë: le nombre médian d'analyses support dépasse les 11 millions d'arbres alors que le maximum dépasse les 10^{40} analyses ! Il y a quand même 35% des phrases qui dépassent le milliard d'arbres et 19% qui dépassent les mille milliards (10^{12}). Malgré ces nombres gigantesques et en prenant en compte le lexeur et le rattrapage d'erreur eventuel, l'analyseur Earley guidé reste très efficace. La forêt partagée prend en moyenne 0,016s pour se construire avec une médiane inférieure à 0,01s et un maximum de 1.98s. Seulement 3% des phrases prennent plus de 0,1s et 1% plus de 0,5s[13].

Nous n'évaluons pas ici la phase de construction des structures fonctionnelles car nous n'avons pas encore mis en place tous les outils qui le permettraient. Il manque en particulier la mesure du nombre de structures fonctionnelles incohérentes ou partielles (cf. 3.2).

Nous avons trouvé que 41% des phrases nécessitent une sur-segmentation (cf. 3.3), qui provient principalement d'un dépassement du temps maximal que l'on a imparti à cette phase. Ce résultat montre que le gain en temps obtenu par le partage du calcul des structures fonctionnelles ne suffit pas à compenser l'ambiguïté massive de la grammaire support. Parmi les phrases qui ont nécessité de la sur-segmentation, certains morceaux de phrase ont dû eux-mêmes être resegmentés. Au total, et parmi les phrases nécessitant une sur-segmentation, la segmentation de niveau 2 a suffi dans 18% des cas et celle de niveau 3 ou 4 dans 62% des cas. Celle de niveau 5 a donc dû être utilisée pour une partie au moins de la phrase dans 20% des cas.

---

[13] Nos tests ont été réalisés sur une architecture AMD Athlon XP 2100+ (1.7 GHz) avec GCC sans optimisation.

6 Conclusion

Dans cet article, nous avons d'une part introduit une variante de LFG, et d'autre part l'analyseur SxLfg qui exploite une spécification de cette variante. À notre connaissance, c'est la première fois qu'un système d'analyse fondé sur le modèle LFG traite du texte tout venant de façon efficace et robuste sans que le pouvoir expressif du formalisme ne soit dégradé.

Il est en outre possible de décrire des phénomènes complexes dans SxLfg en accord avec les nombreux travaux linguistiques qui s'y rapportent. Les expériences relatées utilisent une grammaire du français (et un lexique morpho-syntaxique) que nous avons également réalisée. Les résultats extrêmement encourageants obtenus ne doivent bien entendu pas masquer qu'il s'agit d'une première tentative qui doit se poursuivre et qui peut être améliorée. Les perfectionnements possibles concernent le formalisme, la grammaire du français et l'analyseur SxLfg lui-même.

Nous avons quelques idées pour étendre notre variante de LFG et qui pourrait faciliter certains traitements, en particulier celui des coordonnées. La grammaire doit être étendue et améliorée. En effet, certaines constructions comme les clivées, les comparatives, les coordinations à ellipse, et d'autres, ne sont pas couvertes.

D'autre part, la grammaire support doit être affinée car son ambiguïté actuelle est déraisonnable (cf. section 4). Même si notre analyseur Earley y est relativement peu sensible, elle peut rendre prohibitif le temps d'évaluation des structures fonctionnelles associées.

Les autres pistes de recherche sur l'analyseur proprement dit concernent essentiellement l'amélioration de la robustesse et du temps de calcul des structures fonctionnelles.

References

1. Kaplan, R.: The formal architecture of lexical functionnal grammar. Journal of Informations Science and Engineering (1989)
2. Kaplan, R.M., Maxwell, J.T.: Grammar writer's workbench, version 2.0. Technical report, Xerox Corporation (1994)
3. Andrews, A.: Functional closure in LFG. Technical report, The Australian National University (1990)
4. Briffault, X., Chibout, K., Sabah, G., Vapillon, J.: 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. (1997)
5. Boullier, P.: Guided Earley parsing. In: Proceedings of the 8th International Workshop on Parsing Technologies (IWPT'03), Nancy, France (2003) 43–54
6. Clément, L., Kinyon, A.: XLFG – an LFG parsing scheme for French. In: Proceedings of LFG'01, Hong Kong (2001)
7. Kinyon, A.: Are structural principles useful for automatic disambiguation ? In: Proceedings of in COGSCI'00, Philadelphia, Pennsylvania, United States (2000)