LACL 2012

Logical Aspects of Computational Linguistics

System Demonstrations

July 2–4, 2012
Faculty of Sciences and Technology
University of Nantes
Nantes, France

LACL 2012

Logical Aspects of Computational Linguistics

System Demonstrations

Demo Session Committee:
Denis Béchet
Florian Boudin
Alexandre Dikovsky
Nicolas Hernandez

July 2–4, 2012
Faculty of Sciences and Technology
University of Nantes
Nantes, France

Table of Contents

Metagrammars As Logic Programs

Denys Duchier, Yannick Parmentier, and Simon Petitjean

LIFO, Université d'Orléans, Bâtiment 3IA
6, Rue Léonard De Vinci - BP 6759
F-45067 Orléans Cedex 2, France,

firstname.lastname@univ-orleans.fr

Abstract. In this paper, we introduce the eXtensible MetaGrammar (XMG), which corresponds to both a language for specifying formal grammars, and a compiler for this language. XMG has been developed over the last decade to provide linguists with a declarative and yet expressive way to specify grammars. It has been applied to the design of actual tree-based grammars for French, German or English. XMG relies on a modular architecture, which makes it possible to extend the formalism with additional levels of descriptions and / or linguistic properties. Thus, on top of syntax, XMG can also be used for the description of other linguistic information such as semantics, or morphology (the latter being currently explored for Ikota, an African language spoken in Gabon).

1 Introduction

Since Chomsky's seminal work on generative grammar [1], many formal systems have been proposed to describe the syntax of natural language (see *e.g.* [2]). These mainly differ in terms of expressivity and computational complexity, and generally rely either on rewriting rules (*e.g.* Tree-Adjoining Grammar), or on constraints (*e.g.* Head-driven Phrase Structure Grammar).[1]

An interesting family of formal grammars are *lexicalized* grammars [3]. Such grammars associate each elementary structure (*i.e.* grammar rule) with a lexical item (called anchor). Lexicalized grammars offer two main advantages: firstly, the grammar can be seen as a function mapping lexical items (*i.e.* words) with *uninstantiated* grammatical structures (the grammar is then called lexicon). Secondly, a subgrammar can be extracted from the input grammar according to the sentence to parse, thus speeding up parsing.

An example of lexicalized grammar is Lexicalized Tree-Adjoining Grammar (LTAG). In this formalism, the grammar is made of (thousands of) uninstantiated elementary trees (called tree *templates*), where the leaf nodes contain at least one anchor node (labelled with ⋄). These anchor nodes are attached to adequate lexical items at parsing. As an illustration, consider Fig. 1 depicting two tree templates to be anchored with a transitive verb such as *manger* (to eat).

[1] We do not discuss the distinction between constituency and dependency grammar here, nonetheless the latter can be seen as a constraint-based specification of syntax.

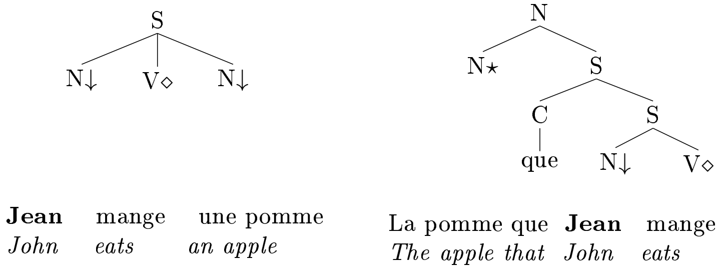


Fig. 1. Elementary structures of an LTAG

From a linguistic point of view, lexicalized grammars allow to express generalizations over lexical entries by gathering tree templates, whose anchor have similar syntactic properties, into *tree families.* From a computational point of view, lexicalized grammars are made of a huge number of structures, due to redundancy within the lexicon (*e.g.* tree templates sharing common subtrees).

The concept of *metagrammar* was introduced by *Candito* [4] in order to deal with structural redundancy by capturing generalizations over tree templates. Instead of directly describing the syntax of language via a formal grammar, the linguist *specifies* the structures of this formal grammar using a dedicated framework. This specification of the grammar is called a metagrammar and is automatically processed to generate the grammar. Many metagrammatical frameworks have been proposed for LTAG [4,5,6,7]. Here we introduce one of these, namely eXtensible MetaGrammar (XMG) [6]. XMG differs from other metagrammar approaches by its declarative specification language, and its modular architecture. The latter made it possible to extend the concept of metagrammars to other levels of description (*e.g.* morphology) and linguistic principles (*e.g.* constraints on word order), as we shall see below.

2 The XMG language

As mentioned above, the XMG language allows for a declarative specification of linguistic structures (including tree descriptions). More precisely, XMG offers a unification-based language *à la* Prolog to specify what a grammar is. This specification is then processed by the XMG compiler in order to produce a computational grammar (*e.g.* an LTAG), which can be saved in an XML file.

Capturing redundancy using abstractions. XMG relies on the concept of *abstraction* to allow the linguist to refer to reusable grammatical units (*e.g.* (combinations of) tree fragments for LTAG). Formally, an XMG specification corresponds to declarative *rules*, which can be defined using the following abstract syntax:

$$\begin{array}{rcl} \mathit{Rule} & := & \mathit{Name} \rightarrow \mathit{Content} \\ \mathit{Content} & := & \mathit{Contribution} \;\mid\; \mathit{Name} \;\mid\; \mathit{Content} \vee \mathit{Content} \;\mid\; \mathit{Content} \wedge \mathit{Content} \end{array}$$

Here, *Contribution* refers to a linguistic fragment of information of a given type (*e.g.* syntax), to be accumulated either conjunctively or disjunctively. Such a fragment is specified using a dedicated description language (*e.g.* a tree description language when describing syntax with LTAG). This language relies on unification variables to share information between distinct XMG rules (*i.e.* distinct grammatical units) or between distinct contributions (*i.e.* between syntax and semantics). The scope of these variables is by default restricted to the rule, but can be extended via *import / export* declarations. As a toy example of these variables and of XMG concrete syntax, consider the rules `CanonicalSubject` and `Subject` below, the latter specifies a generalization over the two possible realizations of a subject shown in Fig. 1 (-> is dominance and >> precedence).

```
class CanonicalSubject          %% (comment) a class is an XMG rule in the abstract syntax
export ?x ?y
declare ?x ?y ?z ?u
{<syn>                          %% contribution of type <syn>
  { node ?x [cat=S] ; node ?z [cat=N] ; node ?y (type=anchor)[cat=V] ; node ?u [cat=N] ;
    ?x -> ?z ; ?x -> ?y ; ?x -> ?u ; ?z >> ?y ; ?y >> ?u }
}

class Subject { CanonicalSubject[] | RelSubject[] }
```

Towards user-defined description languages. Metagrammars bring interesting insights in grammar engineering by offering an abstract view on language, made of combinations of grammatical units. So far, these units were described using a set of hard-coded descriptions languages. To reach extensibility, we are exploring another approach: permitting user-defined description languages (similarly to the grammar, these must be described). Some parts of the compiler thus have to be generated automatically.

3 The XMG compiler

General architecture. As mentioned above, the XMG language is nothing else than a logic language. Its compiler thus share some features with a compiler for logic programs. First, the classes composing the metagrammar (defined using the XMG language introduced above) are converted into clauses of an Extended Definite Clause Grammar (EDCG) [8], which corresponds to a DCG having multiple accumulators. This underlying EDCG explicits the accumulation of contributions of multiple types (*e.g.* syntax, semantics). Then, this EDCG is evaluated according to axioms defined in the metagrammar (comparable to Prolog queries). This produces a list of tuples of contributions (the arity of these tuples is the number of contribution types). Finally, each tuple of this list is optionally post-processed. For instance, tuples whose syntactic contribution is a tree description are fed to a solver in order to produce syntactic trees. During this solving step, it is possible to apply linguistic well-formedness principles (these can use information from other contributions of the tuple).

XMG 2. The first version of XMG (XMG 1.x) was developed between 2003 and 2010 in the Oz programming language, and included only three description languages: one for specifying syntactic trees (either LTAG tree templates or Interaction Grammar tree descriptions), one for specifying semantic representations, and one for specifying the syntax / semantics interface. The development of a new version of XMG from scratch in YAP Prolog started in 2010, in order to extend XMG with the ability to define an arbitrary number of types of contributions (and thus of user-defined description languages).[2]

4 Current state and future work

XMG can be used to describe tree structures, feature structures, predicates, or *properties* of the Property Grammar formalism. Version 2 of the XMG language superseeds Version 1 (being backward-compatible). XMG 2 can be used to compile grammars designed with XMG 1, including the French LTAG and French Interaction Grammar, whose XMG metagrammars are available on-line (along with toy examples of XMG input / output).[3] When describing LTAG tree templates, XMG 2 offers specific linguistic principles, namely ordering between sister nodes, uniqueness of a given node label, and node merging via polarities.

XMG 2 is being actively developed in order to allow for cross-framework grammar engineering, in the lines of [9], but also for linguistic experimentation by defining dynamically its own grammar formalism as mentioned in Section 2.

XMG 2 has been used recently to describe the morphology of verbs in Ikota, an agglutinative Bantu language spoken in Gabon [10]. The idea behind this work is to specify morphemes as contributions in terms of lexical phonology and inflection (morpho-syntactic features). In a next step, we plan to extend this metagrammar (*i.e.* this abstract linguistic account of morphology) to syntax.

References

1. Chomsky, N.: Syntactic Structures. Mouton, The Hague (1957)
2. Abeillé, A.: Les Nouvelles Syntaxes. Armand Colin, Paris (1993)
3. Schabes, Y., Abeillé, A., Joshi, A.K.: Parsing strategies with 'lexicalized' grammars: application to Tree Adjoining Grammars. In: 12th COLING. (1988) 578–583
4. Candito, M.: A Principle-Based Hierarchical Representation of LTAGs. In: 16th COLING. (1996) 194–199
5. Xia, F.: Automatic Grammar Generation from two Different Perspectives. PhD thesis, University of Pennsylvania (2001)
6. Duchier, D., Le Roux, J., Parmentier, Y.: The Metagrammar Compiler: An NLP Application with a Multi-paradigm Architecture. In: MOZ. (2004) 175–187
7. Villemonte De La Clergerie, É.: Building factorized TAGs with meta-grammars. In: TAG+10, New Haven, CO, United States (2010) 111–118
8. Van Roy, P.: Extended dcg notation: A tool for applicative programming in prolog. Technical report, Technical Report UCB/CSD 90/583, UC Berkeley (1990)
9. Duchier, D., Parmentier, Y., Petitjean, S.: Cross-framework Grammar Engineering using Constraint-driven Metagrammars. In: CSLP. (2011) 32–43
10. Duchier, D., Magnana Ekoukou, B., Parmentier, Y., Petitjean, S., Schang, E.: Describing Morphologically-rich Languages using Metagrammars: a Look at Verbs in Ikota. In: 4th Workshop on African Language Technology - LREC. (2012)

[2] Both implementations (XMG 1.x and XMG 2.x) are freely available on-line at `https://sourcesup.renater.fr/xmg` and `https://launchpad.net/xmg` respectively.

[3] `https://sourcesup.cru.fr/scm/viewvc.php/trunk/METAGRAMMARS/?root=xmg`

On Categorial Grammars and Logical Information Systems : using CAMELIS with linguistic data

Annie Foret and Sébastien Ferré

IRISA, Université de Rennes 1
Campus de Beaulieu, 35042 Rennes cedex, France
Email: `foret@irisa.fr`, `ferre@irisa.fr`

Abstract. We have explored in [FF10] different perspectives on how categorial grammars can be considered as Logical Information Systems (LIS)– where objects are organized and queried by logical properties – both theoretically, and practically. LIS have also been considered for the development of pregroup grammars [BF10].
We propose to illustrate these points with the CAMELIS tool that is an implementation of Logical Information Systems (LIS) and that has been developped at Irisa-Rennes. CAMELIS may give another view on linguistic data, and provide an easy help to browse, to update, to create and to maintain or test such data.

1 Logical Information Systems

Logical Information Systems (LIS) are based on Logical Concept Analysis (LCA) [FR04]. LCA is an extension of Formal Concept Analysis that allows to use logical formulas for rich object descriptions and expressive queries.

The LCA framework [FR04] applies to logics with a set-valued semantics similar to description logics [BCM$^+$03]. It is sufficient here to define a logic (see [FR04] for a detailed presentation) as a pre-order of formulas. The preordering is the logical entailment, called *subsumption*: e.g., an interval included in another one, a string matching some regular expression, a graph being a subgraph of another one.

A *logic* is a pre-order $\mathcal{L}_T = (L, \sqsubseteq_T)$, where L is a set of formulas, T is a customizable parameter of the logic, and $\sqsubseteq_T$ is a *subsumption* relation that depends on T. The relation $f \sqsubseteq_T g$ reads "f is more specific than g" or "f is subsumed by g", and is also used to denote the partial ordering induced from the pre-order.

A *logical context* is a tuple $K = (\mathcal{O}, \mathcal{L}_T, X, d)$, where $\mathcal{O}$ is a finite set of objects, $\mathcal{L}_T$ is a logic, $X \subseteq L$ is a finite subset of formulas called the *navigation vocabulary*, and $d \in (\mathcal{O} \to \mathcal{L}_T)$ is a mapping from objects to logical formulas. For any object o, the formula $d(o)$ denotes the *description* of o.

The *extent* of a query formula q in a logical context K is defined by $K.ext(q) = \{o \in \mathcal{O} \mid d(o) \sqsubseteq_T q\}$.

A key feature of LIS, is to allow the tight combination of querying and navigation [Fer09]. The system returns a set of query *increments* that suggest to users relevant ways to refine the query, i.e. navigation links between concepts, until a manageable amount of answers is reached.

A query is a logical formula, and its answers are defined as the extent of this formula, i.e. the set of objects whose description is subsumed by this formula.

Below, screenshots show CAMELIS [1], where the query box is at the top, the extent is presented as a list of object names at the right, and increments are shown as an expandable tree on the left.

Another important aspect of LIS is genericity w.r.t. the logic : LogFun[2] is a toolbox of logic functors [FR06] (logic components, that can be assembled at a high level) ; it can be used in CAMELIS. A dedicated logic has been used in [FF10] to represent *pregroup types*, in order to describe words, phrases, and sentences.

2 Categorial Grammars and their languages

A *categorial grammar* is a structure $G = (\Sigma, I, S)$ where: Σ is a finite alphabet (the words in the sentences); given a set of types Tp, $I : \Sigma \mapsto \mathcal{P}^f(Tp)$ is a function that maps a finite set of types from each element of Σ (the possible categories of each word); $S \in Tp$ is the *main type* associated to correct sentences.

Language. Given a relation on Tp^* called the derivation relation on types : a sentence $v_1 \ldots v_n$ then belongs to the *language of* G, written $\mathcal{L}(G)$, provided its words v_i can be assigned types X_i whose sequence $X_1 \ldots X_n$ derives S according to the derivation relation on types.

An *AB-grammar* is a categorial grammar $G = (\Sigma, I, S)$, such that its set of types Tp is constructed from Pr (primitive), using two binary connectives $/$, $\backslash$, and its language is defined using two deduction rules:

$A \, , \, A \backslash B \vdash B$ (Backward elimination, written $\backslash_e$)

$B \, / \, A \, , \, A \vdash B$ (Forward elimination, written $/_e$)

Lambek L AB-grammars are the basis of a hierarchy of type-logical grammars. The associative Lambek calculus (L) has been introduced in [Lam58], we refer to [Bus97] for details on (L) and its non-associative variant (NL).

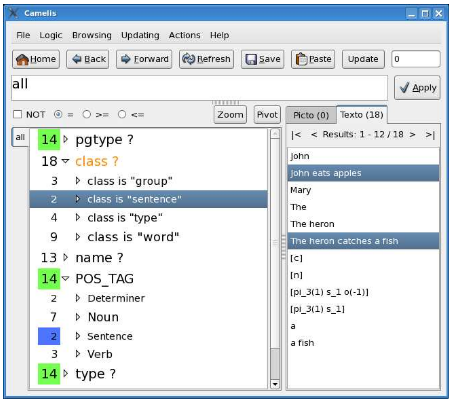
The pregroup formalism has been introduced [Lam99] as a simplification of Lambek calculus [Lam58]. It is considered for the syntax modeling of various natural languages and also practically with parsers [DP05,Oeh04,Béc07] and sofware tools for grammar construction [BF09]. See [Lam99] for a definition.

3 Modelling Approaches for linguistic data

We now illustrate a few uses of CAMELIS with linguistic data, that can be part of the system demonstration.

[1] `http://www.irisa.fr/LIS/ferre/camelis/`

[2] `http://www.irisa.fr/LIS/ferre/logfun/`.


Camelis
File Logic Browsing Updating Actions Help
Home Back Forward Refresh Save Paste Update
Apply
Zoom Pivot
Picto (0) Texto (18)
|< < Results: 1 - 12 / 18 > >|

0

all

□ NOT ● = ○ >= ○ <=

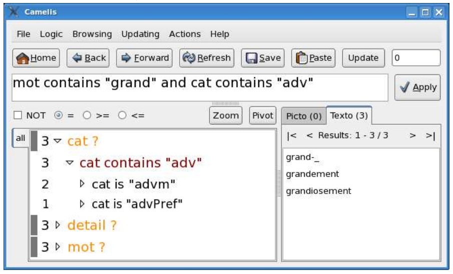
all
14 ▷ pgtype ?
18 ▽ class ?
3 ▷ class is "group"
2 ▷ class is "sentence"
4 ▷ class is "type"
9 ▷ class is "word"
13 ▷ name ?
14 ▽ POS_TAG
2 ▷ Determiner
7 ▷ Noun
2 ▷ Sentence
3 ▷ Verb
14 ▷ type ?

John
John eats apples
Mary
The
The heron
The heron catches a fish
[c]
[n]
[pi_3(1) s_1 o(-1)]
[pi_3(1) s_1]
a
a fish

Fig. 1. A toy grammar, with additional informations, as a LIS context.

Although there can be several modelling approaches for objects/properties, LIS do not force to separate different views : LIS are flexible w.r.t. heterogeneity and partial knowledge.
Objects can be words, sentence fragments, also types or macrotypes as used in[BF09] in lexicon conversion.

A direct application is to consider that the information attached to words[a], are like attributes.


Camelis
File Logic Browsing Updating Actions Help
Home Back Forward Refresh Save Paste Update
Apply
Zoom Pivot
Picto (0) Texto (3)
|< < Results: 1 - 3 / 3 > >|

0

mot contains "grand" and cat contains "adv"

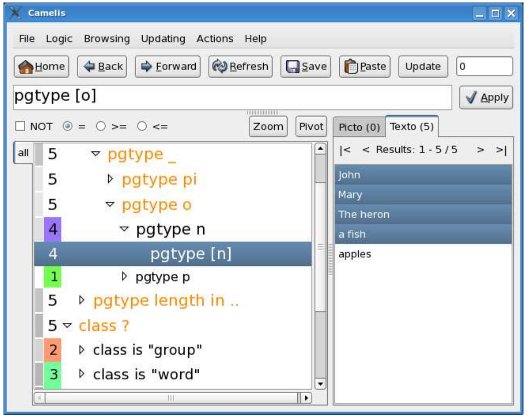
□ NOT ● = ○ >= ○ <=

all
3 ▽ cat ?
3 ▽ cat contains "adv"
2 ▷ cat is "advm"
1 ▷ cat is "advPref"
3 ▷ detail ?
3 ▷ mot ?

grand-_
grandement
grandiosement

Fig. 2. Words with their categories, from a fragment of Lefff3 : selection of adverbs containing “grand”

Another view is to take more advantage of the logic nature of categorial grammars.


Camelis
File Logic Browsing Updating Actions Help
Home Back Forward Refresh Save Paste Update
Apply
Zoom Pivot
Picto (0) Texto (5)
|< < Results: 1 - 5 / 5 > >|

0

pgtype [o]

□ NOT ● = ○ >= ○ <=

all
5 ▽ pgtype _
5 ▷ pgtype pi
5 ▽ pgtype o
4 ▽ pgtype n
4 pgtype [n]
1 ▷ pgtype p
5 ▷ pgtype length in ..
5 ▽ class ?
2 ▷ class is "group"
3 ▷ class is "word"

John
Mary
The heron
a fish
apples

Fig. 3. A toy grammar as a LIS context: `pgtype [o]` vs `pgtype [n]` ($n \leq o$ in the pregroup grammar)

[a] Lefff stands for: “Lexique des Formes Fléchies du Français / Lexicon of French inflected forms”(see `http://alpage.inria.fr/~sagot/lefff-en.html`)

The tool permits query, navigation, but also the construction of new objects and the attachment to properties. Another benefit consists in the execution of actions from CAMELIS, we can illustrate possible connexions with parsers : calls to several parsers of a selected sentence, such as C&C[3] for CCG or PPQ for pregroups [BF10] and help in the construction of type-logical grammars.

References

[BCM$^+$03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[Béc07] Denis Béchet. Parsing pregroup grammars and Lambek calculus using partial composition. *Studia logica*, 87(2/3), 2007.

[BF09] Denis Béchet and A. Foret. (PPQ) : a pregroup parser using majority composition. In *Proc. Parsing with Categorial Grammars, ESSLLI workshop, in Bordeaux, France*, 2009.

[BF10] Denis Béchet and A. Foret. A pregroup toolbox for parsing and building grammars of natural languages. *Linguistic Analysis Journal*, 36, 2010.

[Bus97] W. Buszkowski. Mathematical linguistics and proof theory. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 12, pages 683–736. North-Holland Elsevier, Amsterdam, 1997.

[DP05] Sylvain Degeilh and Anne Preller. Efficiency of pregroup and the french noun phrase. *Journal of Language, Logic and Information*, 14(4):423–444, 2005.

[Fer09] S. Ferré. Camelis: a logical information system to organize and browse a collection of documents. *Int. J. General Systems*, 38(4), 2009.

[FF10] A. Foret and S. Ferré. On categorial grammars as logical information systems. In L. Kwuida and B. Sertkaya, editors, *Int. Conf. Formal Concept Analysis*, LNCS 5986, pages 225–240. Springer, 2010.

[FR04] S. Ferré and O. Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3):383–419, 2004.

[FR06] S. Ferré and O. Ridoux. Logic functors: A toolbox of components for building customized and embeddable logics. Research Report RR-5871, INRIA, March 2006. (103 p.).

[Lam58] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65, 1958.

[Lam99] J. Lambek. Type grammars revisited. In Alain Lecomte, François Lamarche, and Guy Perrier, editors, *Logical aspects of computational linguistics: Second International Conference, LACL '97, Nancy, France, September 22–24, 1997; selected papers*, volume 1582. Springer-Verlag, 1999.

[Oeh04] Richard Oehrle. A parsing algorithm for pregroup grammars. In M. Moortgat and V. Prince, editors, *Proc. of Intern. Conf. on Categorial Grammars*, Montpellier, 2004.

[3] `http://svn.ask.it.usyd.edu.au/trac/candc`

Categorial grammars and wide-coverage semantics with Grail

Richard Moot

LaBRI, CNRS, Université de Bordeaux
351 cours de la Libération
33405 Talence Cedex FRANCE

Abstract. This demonstration shows Grail, a wide-coverage type-logical parser for French which produces Discourse Representation Structures. We will illustrate the different components by means of an example; the demonstration itself will allow people to test the current implementation on their favorite examples themselves.

1 Introduction

Since the work of Bos et al. (2004) for English, we know that it is possible to produce Discourse Representation Structure (DRS, Kamp and Reyle, 1993) semantics for arbitrary English text, going beyond simple wide-coverage parsing to wide-coverage *semantics*. This demonstration will show a system of wide-coverage semantics for French, using the logical view of categorial grammars exemplified by (Moortgat, 2011; Morrill, 2011; Moot and Retoré, 2012).

To illustrate the different computational steps followed and the different tools and models used, we will look at the following (simple) example sentence and show, step by step, how it is transformed into a DRS.

1. Chine: Chen Guangcheng demande à Barack Obama de l'aider à partir.[1]
   China: Chen Guancheng asks Barack Obama to help him leave.

2 Parsing

The first step of the treatment is the parser and it already needs to solve two problems: first, no matter the size of the lexicon, any text contains a number of words which do not appear in the lexicon (for the current example, neither "Chen" nor "Guangcheng" appears in the lexicon). Second, any lexicon large enough to be suitable for wide-coverage parsing necessarily assigns a very large number of formulas to many common words; so many that they become an important bottleneck to parsing. To give an indication, the comma "," already has 63 possible formulas in the current lexicon. A well-known solution to both

[1] Sentence found on `http://www.rtbf.be/info/monde` visited 3 may 2012.

problems is using a *supertagger* (Bangalore and Joshi, 2011), which uses the methods of (statistical) part-of-speech (POS) tagging, with a richer tagset, hence supertagging. In the current context, these richer tags are type-logical formulas.

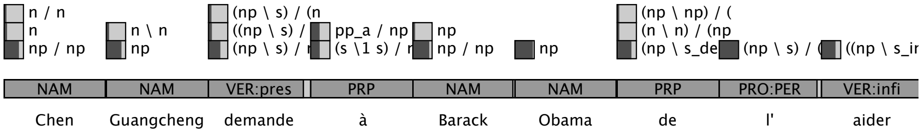
For training and evaluation we have annotated the French Treebank (FTB Abeillé et al., 2000, containing 12.902 sentences, 383.523) plus text from some additional sources (for a total of 13.762 sentences and 405.541 words) with type-logical formulas (see Moot, 2010, for more details on the grammar extraction).

The taggers of Clark and Curran (2004), which consists of a POS-tagger and a supertagger (in addition to several other tools, such as a tagger for Named Entity Recognition; annotation of the FTB with NER data is currently being performed) have been used for training and evaluation. Training was done on nine of out every ten sentences, with every tenth sentence used for evaluation, for a ten-fold cross-validation. The POS-tagger results (using the Treetagger tagset) are among the best-known results for French, 98.5% POS-tags correct (this is in part due to the fact that some difficult choices are left to the supertagger). Supertagger results are shown in Figure 1. The β value is a way of assigning multiple formulas per word (indicated f/w): given a first supertag with probability p, all supertags with probability $\geq \beta p$ are included.

These results compare favorably with the CCG supertagger results for English (Clark, 2002): similar precision but significantly fewer formulas per word (3.8 for the CCG supertagger versus 2.4 here). Note also the the CCG supertagger has a number of lexical rules to reduce the size of the lexicon as well as specific coordination rules whereas the type-logical supertagger has no lexical rules and leaves the correct type assignment for coordination and the interpunction symbols to the supertagger.

Correct	β	f/w
90.4	1.0	1.0
96.3	0.1	1.4
97.1	0.05	1.6
98.2	0.01	2.4

Fig. 1. Supertagger results



n / n		(np \ s) / (n				(np \ np) / (		
n	n \ n	((np \ s) /	pp_a / np	np		(n \ n) / (np		
np / np	np	(np \ s) / r	(s \1 s) / r	np / np	np	(np \ s_de	(np \ s) / (	((np \ s_ir
NAM	NAM	VER:pres	PRP	NAM	NAM	PRP	PRO:PER	VER:infi
Chen	Guangcheng	demande	à	Barack	Obama	de	l'	aider

Fig. 2. POS-tagger and Supertagger results for sentence 1 with $\beta = 0.1$

Figure 2 shows the user interface connecting Grail and the taggers with (part of) the results for sentence 1. The POS-tags are displayed directly above the corresponding words, with the percentage of the bar in a darker shade indicating the level of confidence of the POS-tagger (eg. "demande", which can occur both as a noun and as present tense verb, is the most difficult word for the tagger, with 92.8% confidence). The results of the supertagger are shown above the POS-tags. For "demande", the most likely supertag is $(np\backslash s)/np$ (62.3%, the part of the box before the formula with a darker shade indicates the confidence of the supertagger). With a β value of 0.1 this means that all supertags with a probability greater than 6.2% are included for this word — in the current case $((np\backslash s)/(np\backslash s_{deinf}))/pp_a$ (15.2%, the correct value in the current case) and $(np\backslash s)/(np\backslash s_{ainf})$ (12.9%).

The Grail parser returns a parse for the most likely sequence of formulas for which a parse can be found.

3 Semantics

Given that type-logical proofs correspond to lambda-terms in the simply typed lambda calculus, all that is needed to obtain a semantic recipe for the complete phrase is to provide a lexical lambda term for each of the word-formula pairs used in the proof and beta-reduce the term obtained after lexical substitution.

When providing the entries for the semantic lexicon, we use the fact that most open-class words (such as verbs, nouns and adjectives) have meaning recipes which differ only in the constant used: for example, the meaning of a noun w is simply $\lambda x.w'(x)$ (or, equivalently, simply w'). This means that the lexicon contains several "default" rules for open-class words (applying when no more specific rule does) and several more specific rules for the closed classes, such as determiners and conjunctions, but also for open-class words requiring special treatment (eg. words like "autre" (*other*), "ancien" (*former*) which do not follow the standard adjective meaning recipe). The lexicon also lists such semantic information as the distinction between raising and control verbs.

Figure 3 shows the LaTeX output of Grail for the example sentence (the figure is slightly simplified for ease of exposition and doesn't include temporal information).

$$\left\langle \left\{ \begin{array}{|l|}\hline z_3 \\ \hline \textbf{topic}(e_1, z_3) \\ \textbf{Chine}(z_3) \\ \hline \end{array}, \begin{array}{|l|}\hline x_0 \\ \hline \textbf{nommé}(x_0, \textbf{Barack}) \\ \textbf{nommé}(x_0, \textbf{Obama}) \\ \hline \end{array}, \begin{array}{|l|}\hline y_0 \\ \hline \textbf{nommé}(y_0, \textbf{Chen}) \\ \textbf{nommé}(y_0, \textbf{Guangcheng}) \\ \hline \end{array} \right\}, \begin{array}{|l|}\hline e_1\ y_1 \\ \hline y_1 : \begin{array}{|l|}\hline e_2\ e_3\ x_3 \\ \hline x_3 = ? \\ \textbf{aider\_à}(e_2, x_0, x_3, e_3) \\ \textbf{partir}(e_3, x_3) \\ \hline \end{array} \\ \textbf{demander}(e_1, y_0, x_0, y_1) \\ \hline \end{array} \right\rangle$$

Fig. 3. Semantic output of sentence 1

Proper names are treated as presuppositions and are projected to the topmost DRT, meaning that both "Barack Obama" (with discourse referent x_0) and "Chen Guangcheng" (with discourse referent y_0) are available for further anaphoric resolution. Due to the semantics of ":" with formula $(n\backslash s)/s$, "Chine" (China) is added as the topic of the main sentence. Looking at the main DRS on the right hand side of the figure, we see that there is a single predicate: the verb "demander" (to ask), which takes four arguments, a Davidsonian eventuality e_1, the one doing the asking y_0 (Guangcheng), the one being asked x_0 (Obama) and what is being asked, which is the embedded DRS labeled y_1. The embedded DRS labeled y_1 still contains an unresolved anaphor $x_3 =?$ corresponding to the clitic "l'/le" (him) which still needs to be resolved to Guancheng (y_0, though binding theory would allow z_3, China, as well). In addition the embedded DRS contains the predicate "aider à" (help ... to) with subject x_0 (Obama) object x_3 (the unresolved pronoun) and e_3, the event being helped with, which is "partir(e_3,x_3)", indicating "the event e_3 of x_3 leaving".

Though many refinements and improvements are still possible (and currently being actively developed along several axes), Grail still gets many of the basic semantic facts right, including doubly embedded control verbs and this demo session will allow the people to see the current system in action and to experiment with it using sentences of their choice.

Grail itself, as well as the supertagger and POS-tagger models and the semantic lexicon, are all licensed under the GNU Lesser General Public License.

`http://www.labri.fr/perso/moot/grail3.html`

Bibliography

Abeillé, A., Clément, L., and Kinyon, A. (2000). Building a treebank for French. In *Proceedings of the Second International Language Resources and Evaluation Conference*, Athens, Greece.

Bangalore, S. and Joshi, A. (2011). *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*. MIT Press.

Bos, J., Clark, S., Steedman, M., Curran, J. R., and Hockenmaier, J. (2004). Wide-coverage semantic representation from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-2004)*, pages 1240–1246, Geneva, Switzerland.

Clark, S. (2002). Supertagging for combinatory categorial grammar. In *Proceedings of the 6th International Workshop on Tree Adjoining Grammars and Related Formalisms*, pages 19–24, Venice.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd annual meeting of the Association for Computational Linguistics (ACL-2004)*, pages 104–111, Barcelona, Spain.

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic*. Kluwer Academic Publishers, Dordrecht.

Moortgat, M. (2011). Categorial type logics. In van Benthem, J. and ter Meulen, A., editors, *Handbook of Logic and Language*, chapter 2, pages 95–179. North-Holland Elsevier, Amsterdam, 2 edition.

Moot, R. (2010). Semi-automated extraction of a wide-coverage type-logical grammar for French. In *Proceedings of Traitement Automatique des Langues Naturelles (TALN)*, Montreal.

Moot, R. and Retoré, C. (2012). *The Logic of Categorial Grammars*. Lecture Notes in Artificial Intelligence. Springer. to appear.

Morrill, G. (2011). *Categorial Grammar: Logical Syntax, Semantics, and Processing*. Oxford University Press.

CatLog: A Categorial Parser/Theorem-Prover[⋆]

Glyn Morrill

Universitat Politècnica de Catalunya
morrill@lsi.upc.edu.edu
http://www.lsi.upc.edu/~morrill

Abstract. We present CatLog, a parser/theorem-prover for logical categorial grammar. The logical fragment implemented is a displacement logic the multiplicative basis of which is the displacement calculus of Morrill, Valentín & Fadda (2011)[8].

(Logical) categorial grammar (Morrill 1994[9], 2011[10]; Moortgat 1997[6]; Carpenter 1998[1]; Jäger 2005[4]) originated with Lambek's (1958[5]) insight that a calculus of grammatical types (constituting a residuated monoid) can be formulated using Gentzen's method. The result is an algebraic rendering of grammar as logic and parsing as deduction. Although the design is, really, architecturally perfect and, by now, well-understood, linguistically it is strictly limited to continuity by the fact that it deals with a residuated family with parent (the canonical extension of) concatenation: after all, the whole challenge of modern linguistics for 50 years has been the ubiquity in natural grammar of *dis*continuity. In this relation Morrill, Valentín & Fadda (2011)[8] provides for discontinuity the displacement calculus **D**, deductively a conservative extension of the Lambek calculus **L** with residuated families with respect to both concatenation and intercalation. Like **L**, **D** is free of structural rules and enjoys Cut-elimination and its corollaries the subformula property, decidability, and the finite reading property.

CatLog is a categorial parser/theorem prover implementing a categorial logic extending **D**. It employs Cut-free backward chaining sequent theorem-proving. For **L** deductive spurious ambiguity can be removed by normalization (Hendriks 1993[3]). Because **D** is based on the same design principles, the same techniques can be adopted (Morrill 2011[7]) and CatLog depends on this. In addition to normalization CatLog uses sequent search space pruning by the count invariance of van Benthem (1991[11]). The type-constructors of the displacement logic of CatLog are shown in Fig. 1.

Version f1.2 of CatLog is provisional in a number of respects. In particular, not all spurious ambiguity is eliminated for the categorial logic fragment, and non-duplication of results is achieved by filtering according to a brute force duplication check. Furthermore, bracketing structure must be specified in the input, rather than be induced. And the count-invariance check for multiplicatives

[⋆] This research was partially supported by BASMATI MICINN project (TIN2011-27479-C04-03) and by SGR2009-1428 (LARCA).

$I, \backslash, \cdot, /$	Lambek connectives
$J, \{\downarrow_k, \odot_k, \uparrow_k\}_{k\in\{>,<\}}$	displacement connectives
$\otimes, -$	nondeterministic continuous connectives
$\Downarrow, \circledcirc, \Uparrow$	nondeterministic discontinuous connectives
$\{\hat{}^{k}, \check{}^{k}\}_{k\in\{>,<\}}$	bridge and split
$\hat{}, \check{}$	nondeterministic bridge and split
$\triangleleft^{-1}, \triangleleft, \triangleright, \triangleright^{-1}$	left and right projection and injection
$\&, +$	semantically active additives
$\sqcap, \sqcup$	semantically inactive additives
$\forall, \exists$	first-order quantifiers
¡, $^{+}$	structural modalities
$[\,]^{-1}, \langle\,\rangle$	bracket modalities
$\Box, \blacksquare$	normal modalities
$\vert$	limited contraction for anaphora

Fig. 1. Type-constructors of CatLog

is not adapted to additives and structural modalities. These issues remain topics for future improvement. Nevertheless CatLog f1.2 already provides fast and wide-coverage Montague-like parsing.

The program comprises 3000 lines of Prolog implementing some 80 inference rules for the categorial logic fragment, LaTeX outputting, lexicon, and sample sentences. Among the examples four blocks are distininguished: Dutch examples (cross-serial dependencies), relativization including islands and parasitic gaps, the Montague example sentences of Dowty, Wall and Peters (1981)[2] Chapter 7, and the example sentences of Morrill, Valentín and Fadda (2011)[8].

The functionality is as follows. Once CatLog has been loaded into Prolog, the query `?- pplex.` will cause the lexicon to be pretty printed in the console window, the Dutch part of which is as shown in Fig. 2. The query `?- pplexlatex.` has no visible effect but will cause the lexicon to be output in LaTeX to a file named "s.tex". Querying $t(N)$ will test the examples unifying with term N. For example `?- t(rel(6)).` tests the relativization example 6, `?- t(rel(_)).` tests all the relativization examples, and `?- t(_).` tests all the examples. The analyses — the examples, the derivational proofs, and the semantic readings — appear in the Prolog window, and this information but without duplicate equivalent analyses is written in LaTeX to a file named "t.tex". LaTeXing the file "out.tex" will include s.tex and t.tex and format the lexicon and last analyses made. For example, `?- t(d(2)).` produces the contents in Fig. 3 in Prolog. The LaTeX output for the Dutch part of the lexicon and the same example is as shown in Figs. 4 and 5.

References

1. Bob Carpenter. *Type-Logical Semantics.* MIT Press, Cambridge, MA, 1997.
2. David R. Dowty, Robert E. Wall, and Stanley Peters. *Introduction to Montague Semantics*, volume 11 of *Synthese Language Library.* D. Reidel, Dordrecht, 1981.

```
wil: (NA\Si)in(NA\Sf): LBLC((want (B C)) C)
wil: Q/^(Sfex((NA\Si)in(NA\Sf))): LB(B LCLD((want (C D)) D))
alles: (SAexNt(s(n)))inSA: LBAC[(thing C) -> (B C)]
boeken: Np(n): books
cecilia: Nt(s(f)): c
de: Nt(s(A))/CNA: the
helpen: |>-1((NA\Si)in(NB\(NA\Si))): LCLD((help (C D)) D)
henk: Nt(s(m)): h
jan: Nt(s(m)): j
kan: (NA\Si)in(NA\Sf): LBLC((isable (B C)) C)
kunnen: |>-1((NA\Si)in(NA\Si)): LBLC((isable (B C)) C)
las: NA\(Nt(s(B))\Sf): read
lezen: |>-1(NA\(NB\Si)): read
nijlpaarden: CNp(n): hippos
voeren: |>-1(NA\(NB\Si)): feed
zag: (Nt(s(A))\Si)in(NB\(Nt(s(A))\Sf)): LCLD((saw (C D)) D)
```

Fig. 2. Dutch part of lexicon

```
(d(2)) jan+boeken+kan+lezen S_647


Nt(s(m)): j, Np(n): books, (NA\Si)in(NA\Sf): LBLC((isable (B C)) C), |>-1(ND\(NE\Si)): read => SF


Nt(s(m)), Np(n), (Nt(s(m))\Si)in(Nt(s(m))\Sf), |>-1(Np(n)\(Nt(s(m))\Si)) => Sf [inL]
    Np(n), 1, |>-1(Np(n)\(Nt(s(m))\Si)) => Nt(s(m))\Si [\R]
        Nt(s(m)), Np(n), 1, |>-1(Np(n)\(Nt(s(m))\Si)) => Si [|>-1L]
            Nt(s(m)), Np(n), Np(n)\(Nt(s(m))\Si){1} => Si [\L]
                Np(n) => Np(n)
                Nt(s(m)), Nt(s(m))\Si{1} => Si [\L]
                    Nt(s(m)) => Nt(s(m))
                    Si{1} => Si
    Nt(s(m)), Nt(s(m))\Sf => Sf [\L]
        Nt(s(m)) => Nt(s(m))
        Sf => Sf

((isable ((read books) j)) j)
```

Fig. 3. Dutch verb raising

wil : $(NA\backslash Si){\downarrow}(NA\backslash Sf) : \lambda B\lambda C((want\ (B\ C))\ C)$

wil : $Q/\hat{}(Sf{\uparrow}((NA\backslash Si){\downarrow}(NA\backslash Sf))) : \lambda B(B\ \lambda C\lambda D((want\ (C\ D))\ D))$

alles : $(SA{\uparrow}Nt(s(n))){\downarrow}SA : \lambda B\forall C[(thing\ C) \rightarrow (B\ C)]$

boeken : $Np(n) : books$

cecilia : $Nt(s(f)) : c$

de : $Nt(s(A))/CNA : the$

helpen : $\triangleright^{-1}((NA\backslash Si){\downarrow}(NB\backslash(NA\backslash Si))) : \lambda C\lambda D((help\ (C\ D))\ D)$

henk : $Nt(s(m)) : h$

jan : $Nt(s(m)) : j$

kan : $(NA\backslash Si){\downarrow}(NA\backslash Sf) : \lambda B\lambda C((isable\ (B\ C))\ C)$

kunnen : $\triangleright^{-1}((NA\backslash Si){\downarrow}(NA\backslash Si)) : \lambda B\lambda C((isable\ (B\ C))\ C)$

las : $NA\backslash(Nt(s(B))\backslash Sf) : read$

lezen : $\triangleright^{-1}(NA\backslash(NB\backslash Si)) : read$

nijlpaarden : $CNp(n) : hippos$

voeren : $\triangleright^{-1}(NA\backslash(NB\backslash Si)) : feed$

zag : $(Nt(s(A))\backslash Si){\downarrow}(NB\backslash(Nt(s(A))\backslash Sf)) : \lambda C\lambda D((saw\ (C\ D))\ D)$

Fig. 4. Dutch part of lexicon

(d(2)) **jan+boeken+kan+lezen** : $S_6 47$

$Nt(s(m)) : j, Np(n) : books, (NA\backslash Si){\downarrow}(NA\backslash Sf) : \lambda B\lambda C((isable\ (B\ C))\ C), \triangleright^{-1}(ND\backslash(NE\backslash Si)) : read \ \Rightarrow\ SF$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Np(n) \Rightarrow Np(n)} \qquad \dfrac{\dfrac{}{Nt(s(m)) \Rightarrow Nt(s(m))} \quad \dfrac{}{Si\{\mathtt{1}\} \Rightarrow Si}}{Nt(s(m)), \boxed{Nt(s(m))\backslash Si\{\mathtt{1}\}} \Rightarrow Si}\backslash L}{Nt(s(m)), Np(n), \boxed{Np(n)\backslash(Nt(s(m))\backslash Si)\{\mathtt{1}\}} \Rightarrow Si}\backslash L}{Nt(s(m)), Np(n), \mathtt{1}, \boxed{\triangleright^{-1}(Np(n)\backslash(Nt(s(m))\backslash Si))} \Rightarrow Si}\triangleright^{-1}L}{Np(n), \mathtt{1}, \triangleright^{-1}(Np(n)\backslash(Nt(s(m))\backslash Si)) \Rightarrow Nt(s(m))\backslash Si}\backslash R \qquad \dfrac{\dfrac{}{Nt(s(m)) \Rightarrow Nt(s(m))} \quad \dfrac{}{Sf \Rightarrow Sf}}{Nt(s(m)), \boxed{Nt(s(m))\backslash Sf} \Rightarrow Sf}\backslash L}{Nt(s(m)), Np(n), \boxed{(Nt(s(m))\backslash Si){\downarrow}(Nt(s(m))\backslash Sf)}, \triangleright^{-1}(Np(n)\backslash(Nt(s(m))\backslash Si)) \Rightarrow Sf}{\downarrow}L$$

$((isable\ ((read\ books)\ j))\ j)$

Fig. 5. Dutch verb raising

3. H. Hendriks. *Studied flexibility. Categories and types in syntax and semantics.* PhD thesis, Universiteit van Amsterdam, ILLC, Amsterdam, 1993.
4. Gerhard Jäger. *Anaphora and Type Logical Grammar*, volume 24 of *Trends in Logic – Studia Logica Library*. Springer, Dordrecht, 2005.
5. Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958. Reprinted in Buszkowski, Wojciech, Wojciech Marciszewski, and Johan van Benthem, editors, 1988, *Categorial Grammar*, Linguistic & Literary Studies in Eastern Europe volume 25, John Benjamins, Amsterdam, 153–172.
6. Michael Moortgat. Categorial Type Logics. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 93–177. Elsevier Science B.V. and the MIT Press, Amsterdam and Cambridge, Massachusetts, 1997.
7. Glyn Morrill. Logic Programming of the Displacement Calculus. In Sylvain Pogodalla and Jean-Philippe Prost, editors, *Proceedings of Logical Aspects of Computational Linguistics 2011, LACL'11, Montpellier*, number LNAI 6736 in Springer Lecture Notes in AI, pages 175–189, Berlin, 2011. Springer.
8. Glyn Morrill, Oriol Valentín, and Mario Fadda. The Displacement Calculus. *Journal of Logic, Language and Information*, 20(1):1–48, 2011. Doi 10.1007/s10849-010-9129-2.
9. Glyn V. Morrill. *Type Logical Grammar: Categorial Logic of Signs.* Kluwer Academic Publishers, Dordrecht, 1994.
10. Glyn V. Morrill. *Categorial Grammar: Logical Syntax, Semantics, and Processing.* Oxford University Press, 2011.
11. J. van Benthem. *Language in Action: Categories, Lambdas, and Dynamic Logic.* Number 130 in Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1991. Revised student edition printed in 1995 by the MIT Press.

Ygg, parsing French text using AB grammars

Noémie-Fleur Sandillon-Rezer

Université de Bordeaux
LaBRI/CNRS, 351 cours de la libération
33400 Talence, France {nfsr}@labri.fr
WWW home page: http://www.labri.fr/perso/nfsr

Abstract. This demonstration introduces Ygg, a sentence parser which uses an AB grammar with a probabilistic component and the CYK (Cocke–Younger–Kasami) algorithm. The grammar is extracted from the French Treebank, with a generalized tree transducer, SynTAB (Syntactic Trees to AB). Then, we use it for sentence analysis, from both the French Treebank and the *Est Républicain* corpus.

1 Introduction

This demo focuses on the use of Ygg, our sentence analyzer which uses a probabilistic version of the CYK algorithm [4, 9]. An overview of our various softwares is explained by the scheme 1.

In order to run our analyzer, we need an input grammar and sentences to analyze. The raw sentences come from the French Treebank [1] and the *Est Républicain* corpus [3], and will be typed by the Supertagger [6, 7]. The AB grammar [5] is extracted from the French Treebank, with SynTAB, described in detail in [8]: our tree transducer takes as input the syntactic trees of the French Treebank, and gives as output a forest of AB derivation trees. Among others, we choose an AB grammar for the links with semantics and the possibility to extract λ-terms from the derivation trees. An example of transduction is shown in figure 2.

By gathering the leaves of derivation trees, we can have the usual form of an AB grammar, a lexicon which links words and their various types. However, we decided, for the need of the CYK algorithm, to extract a more usual grammar. The AB grammar is already in Chomsky Normal Form, which is necessary for the algorithm. We added a stochastic component by subdividing the rules from their root (label plus type), and counting the occurrences of various instantiations of an AB grammar ($a \to a/b\ b$ and $a \to b\ b\backslash a$).

The next step, realized for the moment with the CYK algorithm, is to use this grammar to create from raw sentences derivation trees and λ-terms corresponding to them.

2 Sentence Analysis

The use of CYK algorithm has been motivated by the accordance between the grammar extracted from the derivation trees and the ease of adding the prob-

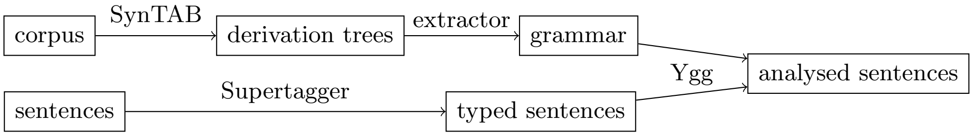


Fig. 1. Processing line.

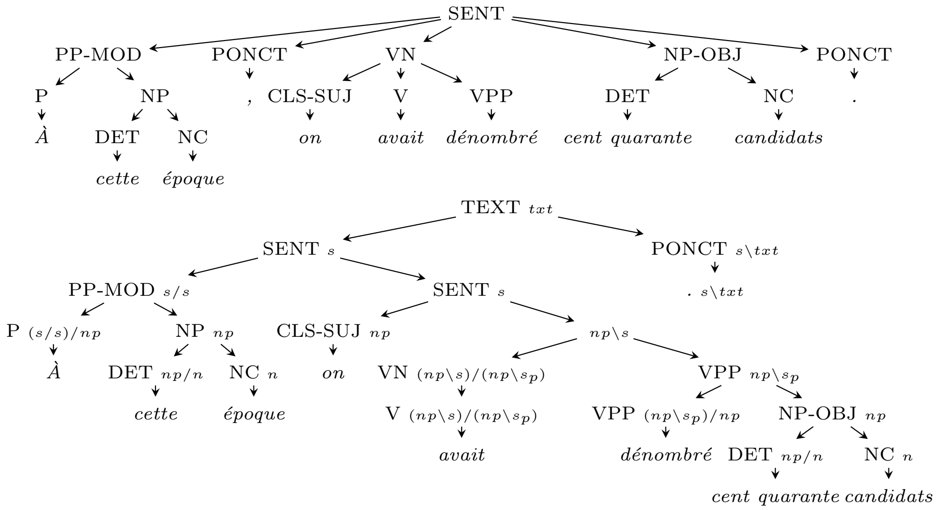


Fig. 2. Input and output of the transducer, for the sentence *"A cette époque, on avait dénombré cent quarante candidats."* ("A this time, 140 candidates have been counted").

abilities. However, we decided to separate the typing phase from the analysis phase.

The word typing done by the Supertagger, enables us to choose the number of types given to a word, by adjusting a β parameter. Indeed, the Supertagger selects the types, for a word, which have a probability greater than β times the greatest probability for this word. If $\beta = 1$ we limit ourselves to one type per word.

The CYK algorithm is known for generating highly ambiguous cases. The probabilistic aspect of it, however, enables us to put aside the trees which have the lowest probabilities, to keep only the best one. When two trees have the same probability, the algorithm will choose the first tree that it found. Having the same probabilities is possible because of the permutation of rules. For the moment, our software doesn't have a GUI and it is invoked with various parameters :

```
$>:./ygg [-a grammar_file] -o output_file [-l log file -p]
sentence_file
```

The bracketed parameters are optional. If the grammar file is not specified, the CYK algorithm will just create possible derivation trees, without probabilities. If a log file is provided, the trees with lesser probabilities will be stocked inside. The option "-p", useless without a grammar file, adds a really small probability on rules which are not in the grammar but are instantiations of AB rules. It enables the program to analyze correct sentences where some rules are not in the grammar, but it does not take priority over the rules of our grammar. At the moment, the probability given to inexistent rules is a C++ constant, the smallest possible value of a `float`.

3 Experimental Results

The tests have been made with a β parameter equal to 0.01. The sentences come from the French Treebank and the *Est Républicain* corpus, which gather sentences from the newspaper *Est Républicain* for the years 1999, 2002 and 2003. The sentences are not annotated or in a tree form, so we cannot apply our transducer on them. We decided to use an excerpt of 520 sentences (the whole corpus, in `XML` format, was not compatible with the Supertagger, and contains thousands sentences), but even if we chose sentences in the same vein as the one of the French Treebank, the vocabulary and the writing style is quite different. On the 12853 sentences of French Treebank, the analysis of 11456 succeed (**89.1**%). On the 520 sentences *Est Républicain* corpus, the success rate is **85.6**%, ie 445 sentences. We can see that the *Est Républicain* results are slightly worse than the French Treebank one.

The output of Ygg is derivation trees corresponding to analyzed sentences, with the probability for each tree. Indeed, Ygg selects the best tree, from a probabilistic point of view, but with the Supertagger the types given to words have their own probabilities, and generally it gives a greater probability to types that are in a similar context. The figure 3 shows the two most probable trees for the sentence *"Celui-ci a importé à tout va pour les besoins de la réunification."* ("This one imported without restraint for the reunification's need."). The main difference between the two trees is the prepositional phrase attachment. Fortunately, the best tree is more representative of the original treebank. The corresponding λ-terms are respectively:

`apl(., apl(apl(pour, apl(les, apl(de, apl(la, réunification)), besoins)), apl(a_tout_va, apl(apl(a, importé), celui-ci))))` and

`apl(., apl(apl(apl(a_tout_va, apl(a, importé)), apl(pour, apl(les, apl(de, apl(la, réunification)), besoins))), celui-ci))`.

4 Software Requirements and Licensing Conditions

Ygg has been implemented in C++, and has been tested on Mac OS X and Linux. The hope of reuse made us choose a GNU GPL licence for every part of the software. The French Treebankis provided by the Paris VII LLF laboratory.

Our work is available at [10], from the transducer to the analyzer.

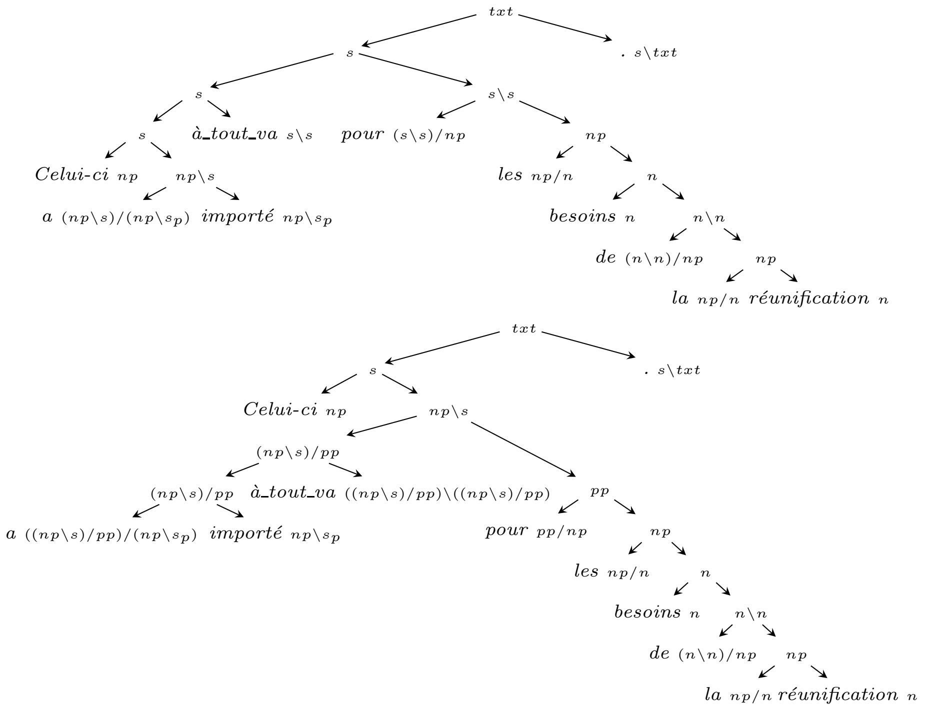


Fig. 3. Probability of the first tree: $5.1e^{-05}$; probability of the second one: $2.3e^{-08}$.

References

1. Abeillé, A., Clément, L., Toussenel, F.: Building a treebank for french. Treebanks, Kluwer, Dordrecht (2003)
2. Comon, H., Dauchet, M., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (1997)
3. B. Gaiffe and K. Nehbi: TEI Est Républicain (2009) http://www.cnrtl.fr/corpus/estrepublicain/
4. Knuth D.E.: The Art of Computer Programming Volume 2: Seminumerical Algorithms (3rd ed.), p.501, Adison-Wesley Professional (1997)
5. Lambek, J.: The mathematics of sentence structure. The American Mathematical Monthly 65 (1958)
6. Moot, R.: Automated extraction of type-logical supertags from the spoken dutch corpus. Complexity of Lexical Descriptions and its Relevance to Natural Language Processing: A Supertagging Approach (2010)
7. Moot, R.: Semi-automated extraction of a wide-coverage type-logical grammar for french. Proceedings TALN 2010, Monreal (2010)
8. Sandillon-Rezer, NF., Moot, R.: Using tree tranducers for grammatical inference. Proceedings LACL 2011, Montpellier (2011)
9. Younger D.: Context free grammar processing in n^3 (1967)
10. Sandillon-Rezer, NF.: Ygg http://www.labri.fr/perso/nfsr/ (2012)