UNIVERSITE D'ORLEANS

LABORATOIRE D'INFORMATIQUE
LIFO
FONDAMENTALE D'ORLEANS

4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

Rapport de Recherche

High-level Methodologies for Grammar Engineering, ESSLLI 2013 Workshop

Denys Duchier and Yannick Parmentier (Eds)
LIFO, Université d'Orléans

Rapport n^o **RR-2013-07**

Preface

These are the proceedings of the International Workshop on High-level Methodologies for Grammar Engineering (HMGE), held at ESSLLI 2013 in Düsseldorf, Germany . This workshop follows a series of workshops dedicated to Grammar Engineering Across Frameworks (GEAF), which took place in 2007 (Stanford), 2008 (Manchester) and 2009 (Singapore). These workshops offer a place to report on research about the design and use of deep grammars for the representation and processing of language.

HMGE was open to submissions on topics including, but not limited to:

- development, maintenance and enhancement of grammars
- semi-automatic acquisition of grammars
- debugging environments for grammar design
- dedicated description languages for grammar engineering
- applications of large-scale grammars

10 papers have been selected via the rigorous efforts of a Program Committee composed of renowned researchers. Each paper has been reviewed by three independent committee members.

HMGE would not have been possible without the precious help of the program committee, the organizing committee, the ESSLLI local organizing committee and their sponsors. Thanks to them all.

Wishing you a fruitful reading,
yours faithfully

June 2013

Denys Duchier & Yannick Parmentier
Local Chairs
HMGE 2013

Organization

Executive Committee

Workshop Chairs: Denys Duchier and Yannick Parmentier
(Université d'Orléans)

Program Committee

Emily M. Bender, Washington, USA
Philippe Blache, Aix-en-Provence, France
Miriam Butt, Konstanz, Germany
Eric De La Clergerie, Paris, France
Benoît Crabbé, Paris, France
Berthold Crysmann, Paris, France
Stefanie Dipper, Bochum, Germany
Denys Duchier, Orléans, France
Claire Gardent, Nancy, France
Josef van Genabith, Dublin, Ireland
Timm Lichte, Düsseldorf, Germany
Montserrat Marimon, Barcelona, Spain
Yusuke Miyao, Tokyo, Japan
Stefan Müller, Berlin, Germany
Gertjan van Noord, Groningen, The Netherlands
Yannick Parmentier, Orléans, France

Table of contents

Demo papers

A Type-Logical Treebank for French

Richard Moot

LaBRI (CNRS), Bordeaux University
Richard.Moot@labri.fr

1 Introduction

Categorial grammars have interesting theoretical advantages, most notably their very clean syntax-semantics interface. In the last decade, research in Combinatory Categorial Grammar has shown that this is not merely a *theoretical* advantage, but that, with the appropriate resources and tools — an annotated treebank, the CCGbank [13], a very efficient parser [10] and a semantic lexicon [4]) — we can use categorial grammars for wide-coverage, deep semantic analysis. Applications of the resulting wide-coverage semantics include natural-language question-answering [5] and computing textual entailments [6].

A key element has been the development of the CCGbank, which has allowed both parameter-optimization for the wide-coverage parser and provided a framework (in types and in derivations) for the semantic applications.

Categorial grammars in the logical tradition [15, 18, 17] have stayed somewhat behind in terms of their application to large-scale linguistic data. The goal of the current paper is to describe the TLGbank, a semi-automatically extracted treebank containing type-logical proofs, created with the explicit goal of making similar wide-coverage parsing and semantics possible in the type-logical context.

2 The French Treebank

The French Treebank (FTB, [1]) is a set of syntactically annotated news articles from the newspaper *Le Monde*. The FTB consists of 12,891 annotated sentences with a total of 383,227 words. The FTB has previously been used to extract lexical-functional grammars [20] and tree adjoining grammars [11].

For the annotation, the FTB uses simple, rather flat trees with some functional syntactic annotation (subject, object, infinitival argument, etc.). Consecutive multiword-expression have been merged in the annotation and neither traces nor discontinuous dependencies have been annotated. Figure 1 in Section 4.1 shows a fragment of a sentence from the FTB. Verb clusters are treated as a constituents (labeled *VN*) and the arguments of the verb occur as sisters of the verbal cluster (eg. the infinitival argument with functional role *OBJ* in Figure 1).

3 Type-Logical Grammars

This section is a very short introduction to (multimodal) type-logical grammars. More detailed introductions can be found in Section 2.4 of [15] and in Chapter 5 of [17].

The atomic formulas are n (for nouns), np (for noun phrases), pp_x (for prepositional phrases, with x the preposition heading the phrase) and s_x for sentences (distinguishing between several types s_{main} for main, tensed sentence, s_{whq} for a wh-question, s_q for a sentence introduced by (*that*) and further types for passives s_{pass}, infinitives s_{inf}, and past s_{ppart} and present s_{ppres} participles; this is inspired by the FTB annotation, though passives are not annotated as such, and the categorial treatments of [9, 13] implemented using first-order logic [16]).

An intransitive verb is assigned $np\backslash s_{main}$, indicating that it requires a noun phrase to its left in order to form an inflected sentence. Similarly, transitive verbs are assigned the formula $(np\backslash s_{main})/np$, requiring a noun phrase to their right in order to form an intransitive verb.

Table 1 lists (a slightly simplified version) of the most common rules used in the extracted treebank. Section 3.1 sketches some linguistic phenomena requiring additional rules and gives some references as to where to find these rules.

$$\frac{}{w \vdash A}\ Lex \qquad\qquad \frac{}{x \vdash A}\ Hyp$$

$$\frac{X \vdash A/B \quad Y \vdash B}{X \circ Y \vdash A}\ /E \qquad\qquad \frac{X \vdash B \quad Y \vdash B\backslash A}{X \circ Y \vdash A}\ \backslash E$$

$$\begin{array}{c} x \vdash B \\ \vdots \\ \dfrac{X \circ x \vdash A}{X \vdash A/B} \end{array}\ /I \qquad\qquad \begin{array}{c} x \vdash B \\ \vdots \\ \dfrac{x \circ X \vdash A}{X \vdash B\backslash A} \end{array}\ \backslash I$$

$$\frac{X[Y] \vdash B \quad Z \vdash B\backslash_1 A}{X[Y \circ Z] \vdash A}\ \backslash_1 E \qquad \begin{array}{c} x \vdash B \\ \vdots \\ \dfrac{X[Y \circ x] \vdash A}{X[Y] \vdash A/\Diamond_1\Box_1 B} \end{array}\ /\Diamond_1\Box_1 I$$

Table 1. Logical rules for multimodal categorial grammars

We will abbreviate the lexicon rule as $\frac{w}{A}$. The rule for $/E$ simply states that whenever we have shown an expression X to be of type A/B and we have shown an expression Y to be of type B, then the tree with X as its immediate subtree on the left and Y as its immediate subtree of the right is of type A.

An easy instantiation of this rule (with $X :=$ *the*, $Y :=$ *student*, $A := np$, $B := n$) would be the following (the $\backslash E$ rule is symmetric).

$$\frac{\mathit{the} \vdash np/n \quad \mathit{student} \vdash n}{\mathit{the} \circ \mathit{student} \vdash np}\,/E$$

The two rules on the bottom of the figure require some special attention. The $\backslash_1 E$ rule is an *infixation rule*. This rule is used for adverbs (and other VP modifiers) occurring after the verb. Like the $\backslash E$ rule, it takes a B formula as its argument, but infixes itself to the right of any subtree Y of X ($X[Y]$ denotes a tree X with a designated subtree Y[1]). An example is shown below for the VP "*impoverishes the CGT dangerously*". The interest of this rule is that it allows a uniform type assignment for adverbs occurring post-verbally, regardless of other verb arguments.

$$\frac{\dfrac{\mathit{appauvrit} \vdash (np\backslash s)/np \quad \mathit{la} \circ \mathit{CGT} \vdash np}{\mathit{appauvrit} \circ \circ(\mathit{la} \circ \mathit{CGT}) \vdash np\backslash s}\,/E \qquad \mathit{dangereusement} \vdash (np\backslash s)\backslash_1(np\backslash s)}{(\mathit{appauvrit} \circ \mathit{dangereusement}) \circ (\mathit{la} \circ \mathit{CGT})}\,/E$$

Finally, the $/\Diamond_1\Box_1$ rule is an *extraction* rule, extracting a B constituent from any right branch inside an X constituent. Section 4.3 shows an example.[2]

3.1 Additional Linguistic Phenomena

The rules listed in Table 1 correspond to the most frequently used rules for the type-logical treebank. The additional rules are a) for the product (primarily used for coordination of multiple arguments (as shown in sentence (1) below, where the two verb arguments *np* and *pp* are conjoined, see Section 2.4 of [18]), b) for gapping (as shown in sentence (2) below, where the transitive verb "atteindre" is absent from the second clause; a multimodal solution is proposed in [12]), and c) for some special rules to treat past-perfect quoted speech, as shown in sentence (3) below. The parenthesized sentence is argument of the past participle "ajouté" and, in addition, this argument is discontinuous. The solution is essentially to analyse the entire verb group missing the s argument "a ajouté ..." as $s_{main}\backslash_1 s_{main}$.

(1) ... augmenter $[_{\mathrm{np}}$ ses fonds propres $]$ $[_{\mathrm{pp}}$ de 90 millions de francs $]$ et
... increase $[_{\mathrm{np}}$ its equity $]$ $[_{\mathrm{pp}}$ by 90 million francs $]$ and
$[_{\mathrm{np}}$ les quasi-fonds propres $]$ $[_{\mathrm{pp}}$ de 30 millions $]$ ...
$[_{\mathrm{np}}$ its quasi-equity $]$ $[_{\mathrm{pp}}$ by 30 million $]$ ...

(2) Le salaire horaire atteint dorénavant 34,06 francs et le
The wages per hour reach from now on 34,06 francs and the
SMIC mensual brut $[_{\mathrm{tv}}$ $]$ 5756,14 francs.
gross minimum monthly wage $[_{\mathrm{tv}}$ $]$ 5756,14 francs.

[1] For adverbs, as here, Y is typically the verb, but in principle infixation is possible anywhere (an admitted simplification)

[2] For readers familiar with the displacement calculus [19], the infixation construction $A\backslash_1 B$ corresponds to $\check{}B \downarrow A$ and the extraction construction $A/\Diamond_1\Box_1 B$ to $\hat{}(A \uparrow B)$

(3) $[_{\mathrm{sl}}$ Les conservateurs], a ajouté le premier ministre ..., $[_{\mathrm{sr}}$ “ne sont pas des opportunistes qui virevoltent d’une politique à l’autre]
$[_{\mathrm{sl}}$ The Conservatives], has added the Prime Minister ..., $[_{\mathrm{sr}}$ “ are not opportunists who flip-flop from one policy to another]

4 Grammar Extraction

Grammar extraction algorithms for categorial grammars follow a general methodology (see, for example, [7, 13], shown as item 2 below) with some additional rules to deal with the quirks of the format of the input treebank. A high-level description of the grammar extraction algorithm used for the FTB is given below.

1. split multiword expressions,
2. binarize the tree, keeping track of the distinction between modifiers and arguments, arguments are assigned formulas based on their syntactic label (eg. np for a noun phrase argument, $np\backslash s_{inf}$ for an infinitival argument, etc.)
3. reattach verb cluster arguments,
4. rearrange coordinations,
5. insert traces in the appropriate places and assign the appropriate formulas to relative pronouns and clitics

Unfortunately, nearly all of these steps require at least some human intervention: the FTB annotation makes the distinction between modifiers and arguments only for certain categories (sentences, infinitive phrases, present participle phrases, but not past participle phrases or noun phrases), meaning that for many major categories this information is not explicitly annotated and needs to be verified manually.

4.1 Verb Clusters

As discussed in Section 2, verb clusters (which include clitics and adverbs) and their arguments are sisters in the FTB annotation trees. Figure 1 shows an example corresponding to sentence (4).

(4) Ils ont déjà pu constater que (...)
They have already been able to note that (...)

In a categorial setting, we obtain a much simpler analysis if these VN arguments are arguments of the embedded verbs instead (in the current case, we’d like the infinitival group to be the argument of the past participle “pu” (of the verb “pouvoir”, *can*). At the bottom of Figure 1 we see the rightward branching structure which results from the corpus transformation. Note also how the adverb “déjà” (*already*) is assigned the VP-modifier formula $(np\backslash s_x)/(np\backslash s_x)$ which is parametric for the type of sentence (in essence, this is a formula with an implicit first-order quantifier ranging over the different sentence types, see [16] or Section 2.7 of [15]; in the figure, x is instantiated to *ppart*).

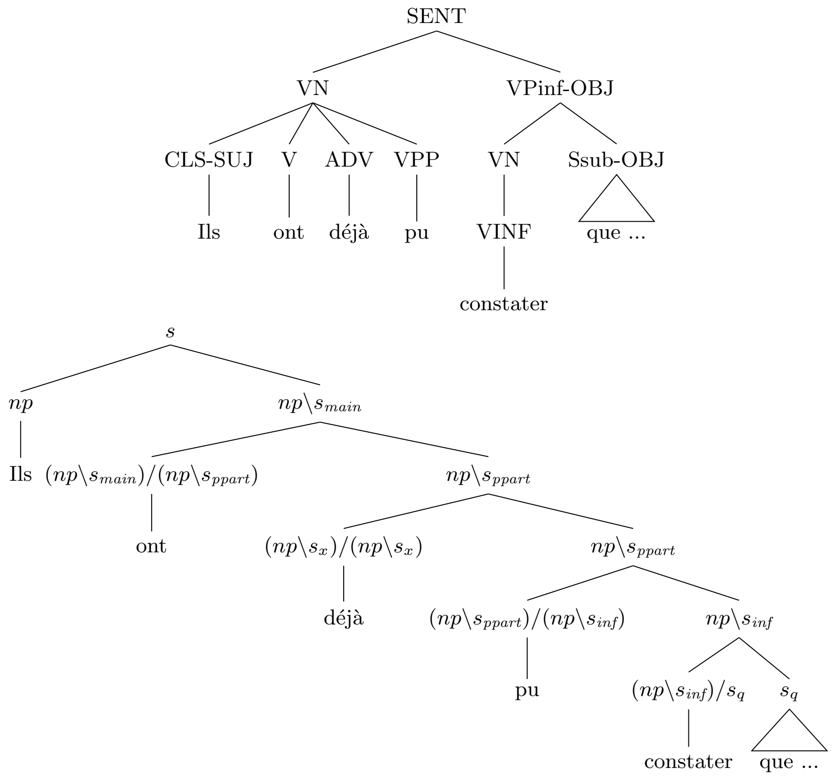


Fig. 1. Rebracketing a verbal group and its arguments

4.2 Coordination and Interpunction Symbols

The sentences below illustrate some of the problems with coordinations which we will discuss in this section.

(5) Elles reprennent et amplifient des programmes existants ou
They resume and amplify programs existing or
en cours d' adaptation
currently being adapted

(6) Les lieux où les deux derniers morts ont été recensés,
The places where the two last deaths have been reported,
lundi 30 décembre, La Yougoslavie et La Colombie, (...)
Monday 30 December, Yugoslavia and Colombia,

Figure 2 shows the FTB syntactic structure of sentence (5). In categorial grammars, conjunctions like "ou" (*or*) are generally assigned instances of the formula $(X\backslash X)/X$ (for a contextually appropriate choice of the formula X). The first conjunction is of the two transitive verbs (instantiating X with the formula $(np\backslash s_{main})/np$) who share both the subject and the object. For the second coordination it is the adjective and the prepositional phrase which are conjoined (though this is not so clear from the annotation only, where it seems an unlike coordination between an np and a pp). As is standard in categorial grammars, we assign both the adjective and the PP the formula $n\backslash n$ (this is the standard assignment for a PP modifying a noun), turning this seemingly unlike coordination into a trivial instance of the general coordination scheme.

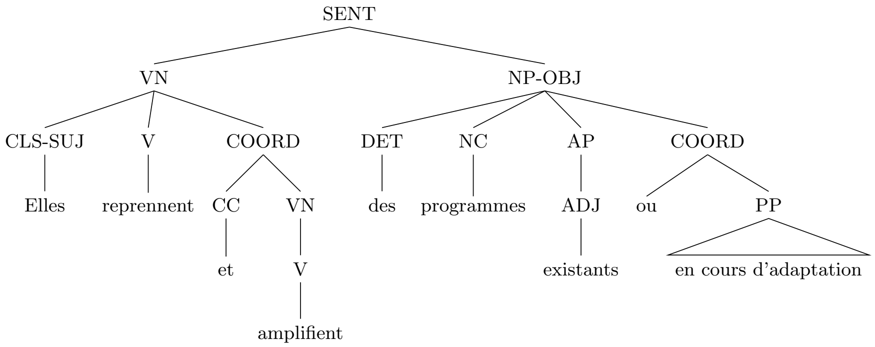


Fig. 2. Coordination

The (somewhat simplified) FTB annotation of sentence (6) of Figure 3 on the next page, shows another problem: appositives, which are treated by assigning a coordination-like formula to the interpunction symbol preceding them (a similar solution is used for parentheticals and for most extrapositions[3]) Additionally, we have to distinguish between the NP-MOD temporal adverb (which modifies the verb "recensés" and the NP-MOD for the appositive (which conjoins to "Les lieux", *the places*)

As the example shows, these cases are difficult to infer from the information provided by the FTB annotation alone, and therefore must be verified manually; in total a bit over 20% of the interpunction symbols — over ten thousand interpunction symbols — are assigned coordination-like categories.

[3] Not all extrapositions can be analysed as coordinations this way. In the example below

(i) A celà s'ajoute une considération générale : (...)
To that adds-itself a general consideration

"A cela" is assigned $s/(s/\Diamond_1\Box_1 pp_a)$ allowing it to function as a long-distance pp argument to "s'ajoute".

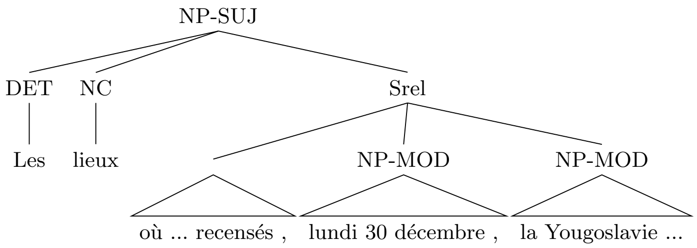


Fig. 3. Appositives

4.3 Traces and Long-Distance Dependencies

As an example of a simple long-distance dependency in the corpus, consider the example below.

(7) Premier handicap auquel il convenait de s'attaquer : l'inflation
First handicap to which it was agreed to attack : the inflation

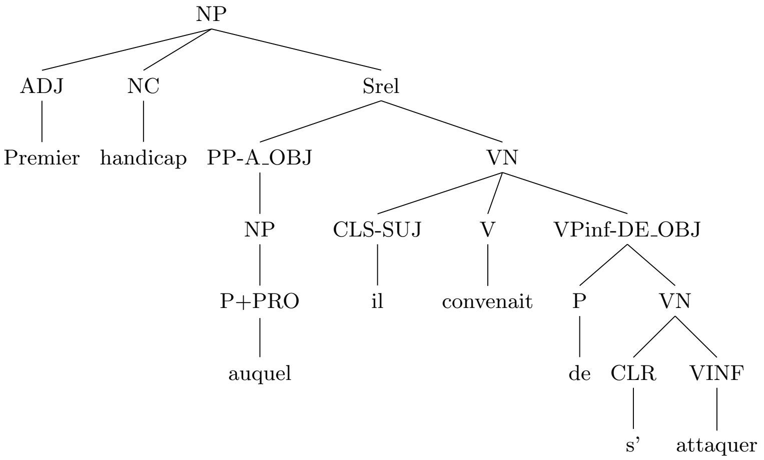
Figure 4 on the next page shows how the insertion of traces works. In the input structure on the top of the figure, "auquel" (*to which*) is assigned a preposition+pronoun POS-tag and assigned the role of a prepositional object with the preposition "à" (*to*). However, this preposition is an argument of the verb "s'attaquer à" (*to attack*), which occurs much lower in the annotation tree. Since none of these dependencies are annotated in the French Treebank, all relative pronouns, wh-pronouns and clitics — a total of over 3,000 occurrences in the corpus — have been manually annotated with the correct long-distance dependencies. At the bottom of Figure 4, the manually added long-distance dependency is shown.

4.4 Analysis

Categorial grammars, much like lexicalized tree adjoining grammars and other strongly lexicalized formalisms, use very construction-specific lexical entries. This means, for example, that when a verb can be used both as a transitive verb and as an intransitive verb, it will have (at least) two distinct lexical entries. For extracted grammars, this generally means a very high level of lexical ambiguity.

Using the most detailed extraction parameters, the final lexicon uses 1101 distinct formulas (though only 800 of these occur more than once and, 684 more than twice and 570 at least five times).

Using a slightly less detailed extraction (which, for example, distinguishes only pp_{de}, pp_a and pp_{par} and uses simply pp for prepositional phrases headed by other prepositions) there are 761 different formulas used in the lexicon (of which only 684 occur more than once, 546 occur more than twice and 471 occur at least five times)



$$\dfrac{\dfrac{\text{auquel}}{(n\backslash n)/(s_m/\Diamond_1\Box_1^{\downarrow}pp_{à})}\,[\mathit{Lex}] \qquad \dfrac{\dfrac{\dfrac{\text{il}}{np}\,[\mathit{Lex}] \qquad \dfrac{\dfrac{\text{convenait}}{(np\backslash s_m)/(np\backslash s_{di})}\,[\mathit{Lex}] \qquad \dfrac{\dfrac{\text{de}}{(np\backslash s_{di})/(np\backslash s_i)}\,[\mathit{Lex}] \qquad \dfrac{\dfrac{\text{s'}}{cl_r}\,[\mathit{Lex}] \qquad \dfrac{\dfrac{\text{attaquer}}{(cl_r\backslash(np\backslash s_i))/pp_{à}}\,[\mathit{Lex}] \qquad \dfrac{}{\mathrm{p}_0 \vdash pp_{à}}\,[\mathit{Hyp}]_1}{\mathrm{a}\circ\mathrm{p}_0 \vdash cl_r\backslash(np\backslash s_i)}\,[/E]}{\text{s'}\circ(\mathrm{a}\circ\mathrm{p}_0)\vdash np\backslash s_i}\,[\backslash E]}{\text{de}\circ(\text{s'}\circ(\mathrm{a}\circ\mathrm{p}_0))\vdash np\backslash s_i}\,[/E]}{\mathrm{c}\circ(\text{de}\circ(\text{s'}\circ(\mathrm{a}\circ\mathrm{p}_0)))\vdash np\backslash s_m}\,[/E]}{\text{il}\circ(\mathrm{c}\circ(\text{de}\circ(\text{s'}\circ(\mathrm{a}\circ\mathrm{p}_0))))\vdash s_m}\,[\backslash E]}{\text{il}\circ(\mathrm{c}\circ(\text{de}\circ(\text{s'}\circ\mathrm{a})))\vdash s_m/\Diamond_1\Box_1^{\downarrow}pp_{à}}\,[/I]_1}{\text{auquel}\circ(\text{il}\circ(\mathrm{c}\circ(\text{de}\circ(\text{s'}\circ\mathrm{a}))))\vdash n\backslash n}\,[/E]$$

Fig. 4. Adding traces to the output

Even in this second lexicon, many frequent words have a great number of lexical assignments. The conjunction "et" (*and*) has 86 different lexical formulas, the comma "," (which, as we have seen, often functions much like a conjunction) is assigned 72 distinct formulas, the adverb "plus" (*more*) 44 formulas (in part because of possible combinations with "que", *than*), the prepositions "pour", "en" and "de" 43, 42 and 40 formulas respectively, and the verb "est" (*is*) 39 formulas.

Though this kind of lexical ambiguity may seem like a problem when using the lexicon for parsing, well-known techniques such as *supertagging* [2], which assign the contextually most probable set of formulas (supertags) to each word, can be used to reduce the lexical ambiguity to an acceptable level. To give an idea as to how effective this strategy is in the current context and with the reduced lexicon of 761 formulas, when assigning only the most likely formula to each word, 90.6% of the words are assigned the correct formula, when assigning each word all formulas with probability greater than 1% of the most likely supertag (for an average of 2.3 formulas per word), the supertagger assigns 98.4% (complete treebank, using ten-fold cross-validation).

4.5 Comparison With the CCGbank

Apart from the obvious theoretical differences between CCG and type-logical grammars and the different treatment of certain linguistic phenomena — such as extraction — that this implies, it is worth spending some time on some of the less obvious differences between the two treebanks.

Whereas the CCGbank uses a certain number of non-combinatory rules (notably for extraposition and coordination, but also to transform passives $np\backslash s_{pass}$ into adjectives $n\backslash n$ and (bare) nouns n into noun phrases np, the current treebank uses no non-logical rules. As a result, the lexicon of the type-logical treebank does more of the work (and consequently, the taks of the supertagger is more difficult).

If we want to reduce the size of the lexicon in a way similar to the CCGbank, there are two basic options:

- the first option is to allow non-logical rules in the same spirit as the CCGbank,
- the second option, more in line with the general spirit of type-logical grammars, is to exploit the derivability relation and to replace the analysis of passives by a formula F such that $F \vdash n\backslash n$ (see Section 4.4.2 of [18] for a particularly nice solution).

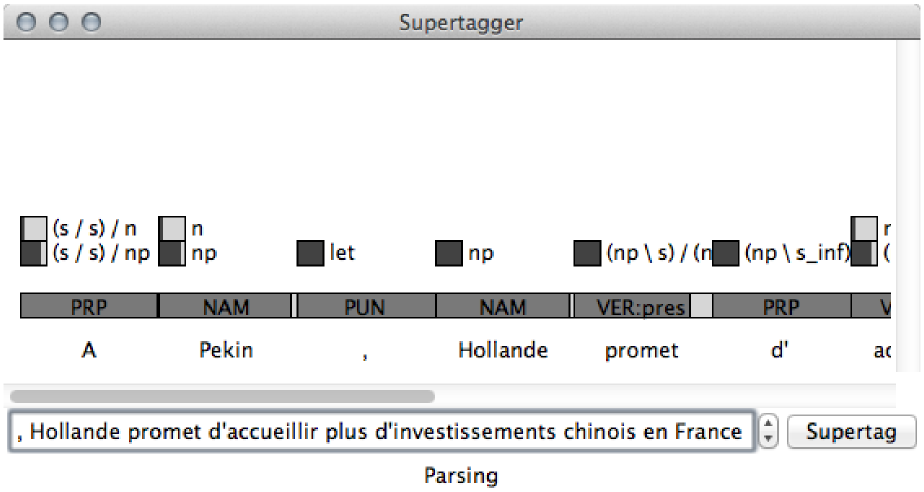
However, we leave the transformation of the proofs in the corpus in these two ways to future research.

5 Tools

To facilite annotation, correction and parsing, several tools have been developed, using a combination of Prolog and TclTk. In addition, several well-known tools have been used for the exploitation of the corpus: the Stanford Tregex tool [14] for browsing and querying the French Treebank (as well as some of its transformations) and the C&C tools [10] for training POS-tag and supertag models using the annotated corpus.

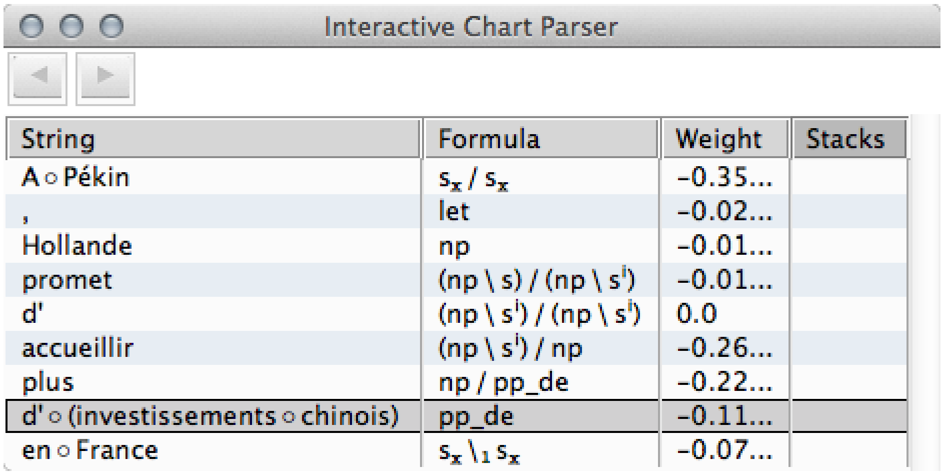
Figure 5 on the next page shows a screenshot of the interface to the supertagger and parser. This "horizontal" interface allows the user to type in sentences and see the resulting semantic output from the parser. The darker-shader percentage of the block to the left of the formula gives a visual indication of the probability assign to the formula (the exact numbers can be seen by moving the mouse over the corresponding area). Apart from some configuration options, this interface is not interactive.

Figure 6 shows a screenshot of the "vertical" interface to the parser and supertagger. This is an interactive interface, allowing the user to select (or type in) the desired formula — to help prevent errors, the current frequency of the chosen formula for the current word is displayed after a manual choice of formula— as well as allowing the user the select the parser rule applications by clicking on one of the premisses for a rule (an additional dialog pops up in case the rule choice is ambiguous). The weight column shows the log-probability of the item.



, Hollande promet d'accueillir plus d'investissements chinois en France

Fig. 5. Screenshot of the supertagger interface



String	Formula	Weight	Stacks
A ○ Pékin	s_x / s_x	-0.35...	
,	let	-0.02...	
Hollande	np	-0.01...	
promet	$(np \backslash s) / (np \backslash s^i)$	-0.01...	
d'	$(np \backslash s^i) / (np \backslash s^i)$	0.0	
accueillir	$(np \backslash s^i) / np$	-0.26...	
plus	np / pp_de	-0.22...	
d' ○ (investissements ○ chinois)	pp_de	-0.11...	
en ○ France	$s_x \backslash_1 s_x$	-0.07...	

Fig. 6. Screenshot of the interactive parser

6 Bootstrapping

Given that the French Treebank is somewhat small compared to other treebanks and given that the conversion of the FTB to the type-logical treebank was rather labour-intensive, it makes sense to look at more effective and efficient ways of increasing the size of the treebank. The tools described in the previous section, interfacing with the supertagger and the parser for the core corpus are useful in this respect.

Currently, slightly over 1,600 additional sentences have been annotated (for a total annotated corpus of 14,539 sentences and 421,348 words). Most of these sentences come from the Sequoia treebank [8] and the French Timebank [3]. The observed accuracy of the supertagger for these sentences from the *L'Est Républicain* newspaper is slightly lower than the results reported in Section 4.4: in 88.1% of cases, the best supertag is correct, and 97.6% of cases the correct supertag has probability greater than 1% of the best supertag (compared to 90.6 and 98.4% respectively for the cross-validated results). Part of this difference might be attributed to stylistic differences between the two newspapers (initial experiments with annotating unseen sentences from *Le Monde* seem to confirm this) but it may also be the case that cross-validation gives a somewhat optimistic picture of actual performance on unseen data from other sources (the different training and test sets not being completely independent).

7 Obtaining the Tools and Resources

All tools, as well as the POS-tagger and supertagger models and a semantic lexicon in the style of [4], are available from the author's website under the LGPL licence. The TLGbank, being a derived work, is available under the same licensing conditions as the French Treebank. The Sequoia/L'Est Républicain part of the treebank is available under the LGPL-LR licence.

8 Conclusions

We have shown how the French Treebank has been semi-automatically transformed into a set of derivations in multimodal type-logical grammars. This is an important first step in training an evaluating wide-coverage type-logical parsers and we hope to see several competitive type-logical parsers in the future.

References

1. Abeillé, A., Clément, L., Kinyon, A.: Building a treebank for French. In: Proceedings of the Second International Language Resources and Evaluation Conference. Athens, Greece (2000)
2. Bangalore, S., Joshi, A.: Supertagging: Using Complex Lexical Descriptions in Natural Language Processing. MIT Press (2011)
3. Bittar, A.: Building a TimeBank for French: A Reference Corpus Annotated According to the ISO-TimeML Standard. Ph.D. thesis, Université Paris Diderot (2010)
4. Bos, J., Clark, S., Steedman, M., Curran, J.R., Hockenmaier, J.: Wide-coverage semantic representation from a CCG parser. In: Proceedings of the 20th International Conference on Computational Linguistics (COLING-2004). pp. 1240–1246. Geneva, Switzerland (2004)

5. Bos, J., Curran, J.R., Guzzetti, E.: The Pronto QA system at TREC-2007: harvesting hyponyms, using nominalisation patterns, and computing answer cardinality. In: Voorhees, E.M., Buckland, L.P. (eds.) The Sixteenth Text RETrieval Conference, TREC 2007. pp. 726–732. Gaitersburg, MD (2007)
6. Bos, J., Markert, K.: Recognising textual entailment with logical inference. In: Proceedings of the 2005 Conference on Empirical Methods in Natural Language Processing (EMNLP 2005). pp. 628–635 (2005)
7. Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica 49, 431–454 (1990)
8. Candito, M., Seddah, D.: Le corpus Sequoia : Annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical. In: Proceedings of Traitement Automatique des Langues Naturelles (TALN). Grenoble (2012)
9. Carpenter, B.: Categorial grammars, lexical rules and the english predicative. In: Levine, R. (ed.) Formal Grammar: Theory and Practice. No. 2 in Vancouver Studies in Cognitive Science, University of British Columbia Press, Vancouver (1991)
10. Clark, S., Curran, J.R.: Parsing the WSJ using CCG and log-linear models. In: Proceedings of the 42nd annual meeting of the Association for Computational Linguistics (ACL-2004). pp. 104–111. Barcelona, Spain (2004)
11. Dybro-Johansen, A.: Extraction automatique de grammaires à partir d'un corpus français. Master's thesis, Université Paris 7 (2004)
12. Hendriks, P.: Ellipsis and multimodal categorial type logic. In: Morril, G., Oehrle, R.T. (eds.) Proceedings of Formal Grammar 1995. pp. 107–122. Barcelona, Spain (1995)
13. Hockenmaier, J., Steedman, M.: CCGbank, a coxbrpus of CCG derivations and dependency structures extracted from the Penn Treebank. Computational Linguistics 33(3), 355–396 (2007)
14. Levy, R., Andrew, G.: Tregex and tsurgeon: tools for querying and manipulating tree data structures. In: 5th International Conference on Language Resources and Evaluation (LREC 2006). (2006)
15. Moortgat, M.: Categorial type logics. In: van Benthem, J., ter Meulen, A. (eds.) Handbook of Logic and Language, chap. 2, pp. 95–179. North-Holland Elsevier, Amsterdam (2011)
16. Moot, R.: Extended lambek calculi and first-order linear logic. to appear in Springer Lecture Notes in Artificial Intelligence (2013)
17. Moot, R., Retoré, C.: The Logic of Categorial Grammars. Lecture Notes in Artificial Intelligence, Springer (2012)
18. Morrill, G.: Categorial Grammar: Logical Syntax, Semantics, and Processing. Oxford University Press (2011)
19. Morrill, G., Valentín, O., Fadda, M.: The displacement calculus. Journal of Logic, Language and Information 20(1), 1–48 (2011)
20. Schluter, N., van Genabith, J.: Treebank-based acquisition of LFG parsing resources for French. In: Proceedings of the Sixth International Language Resources and Evaluation (LREC'08). Marrakech, Morocco (2008)

Clustering for categorial grammar induction

Noémie-Fleur Sandillon-Rezer
nfsr@labri.fr

LaBRI, CNRS
351 cours de la Libération, 33405 Talence

Abstract. In this article, we describe the way we use hierarchical clustering to learn an AB grammar from partial derivation trees. We describe AB grammars and the derivation trees we use as input for the clustering, then the way we extract information from Treebanks for the clustering. The unification algorithm, based on the information extracted from our clusters, will be explained and the results discussed.

1 Introduction

The aim of this article is to present a new grammatical inference method.

The input of our inference procedure is partially specified derivation trees, which are then guided by a hierarchical clustering to know which variables must be unified. The basic idea is that words in similar context get similar formulas. The numerous information from the treebanks are used in the clustering step and we not use a supertagger to tame the lexical ambiguity, given we already did it in [Sa2] and we wanted to stay close to a Buszkowski and Penn inference.

The categorial grammars we use are the AB grammars, defined by Ajdukiewicz [Aj1] and Bar-Hillel [Ba1] from the core of categorial grammars and are a subsystem of both the Lambek calculus [La1] and Combinatory Categorial Grammar[1] [Ho1]. They have only two elimination rules (see table 1).

Table 1. The elimination rules for AB grammar

$$\frac{A/B \quad B}{A}\,[/E] \qquad \frac{B \quad B\backslash A}{A}\,[\backslash E]$$

They are used since Lambek's work to represent formally naturals language, and if some constructions are complex to treat, some solutions are summed up in the article of Moot [Mo2].

Grammatical inference for categorial grammars is performed in three ways, depending on the method and the input structures.

[1] Though note that we follow Lambek's convention of always having the result category above the slash.

The method of Adrians [Ad1, TA1] starts from sentences with no structure. Though this works well in many cases, it is hard for such grammars to correctly infer PP attachments and since there are treebanks where this information has been correctly annotated, it makes sense to exploit this information.

Methods using partial structures are described by Buzkowski and Penn [Bu1, BP1] or Kanazawa [Ka1], which are clearly in a Gold paradigm learning [Go1]. The input structures are described later (section 3), but the output is a rigid grammar in the case of Buszkowski and Penn or a k-valued grammar in the case of Kanazawa's algorithm. A rigid grammar is not representative of a natural langage, and when $k \geq 2$, it has been proved that the problem of grammatical inference is NP-hard [CF1]. However, even a 2-valued grammar cannot represent natural languages: experience with extracted grammars shows that the maximum number of types per word is typically large, with many frequent words having over forty types [Ho1, SM1].

Finally, we have the methods using fully specified structure like the one of Hockenmaier [Ho2], which computes a Combinatory Categorial Grammar, or Sandillon-Rezer and Moot [SM1], which uses a generalized tree transducer to transform syntactic trees into derivation trees of an AB grammar [La1]. It is the output of the last method that serves both as a gold standard for evaluation and, after erasing the manually annotated formulas while keeping only the rule names, as input to our inference algorithm, and we focus on the way we get trees in section 2.

In this article we combine the second method which uses partial structures, complemented with some information from the treebanks, with clustering. We evaluate both the complexity of the task and the quality of the obtained lexicon. Clustering is done using a similarity measure based on the local context of the words, directly extracted from a treebank.

The derivation trees are extracted from annotated treebanks. We use two different treebanks for this purpose: the French Treebank [AC1] and Sequoia [CS1]. The two corpora are syntactically annotated using similar annotation protocols [AC2]. The main differences between the two are the number of sentences and the origin of these sentences. The French Treebank is composed by 12.351 sentences which come from a selection of articles from the newspaper *Le Monde*, and Sequoia is composed by 3.204 sentences coming from various sources, like Wikipedia, the newspaper *L'Est Républicain*, or medical references. Figure 1 shows an example of a syntactic tree. The pre-terminal nodes contain the POS-tag of the word, the other internal nodes contain the phrasal type of their subtree and the leaves represent the words of the sentence.

Since the annotation format does not correspond to AB derivation trees, we use a generalized tree transducer to transform the trees from both treebanks into AB derivation trees.

We will start by describing the AB grammar generated by the tree transducer, then we will briefly recall the general unification algorithm for AB grammars and describe the one we use. In the fourth section, we will describe the format of the vectors we use as input for the clustering step. The evaluation of our method will be shown just after, and a discussion about possible extensions of this work will precede the conclusion.

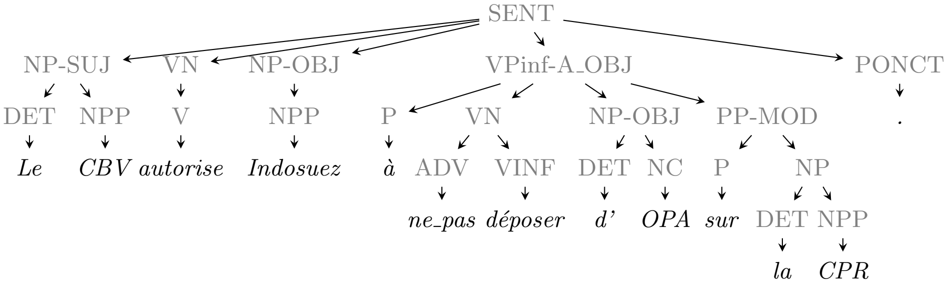


Fig. 1. Example of syntactic tree from the French Treebank: "The CBV allows Indosuez not to make a tender offer on the CPR."

2 Getting AB derivations

The derivation trees of an AB grammar represent the successive applications of elimination rules. To get them, we use a generalized top-down tree transducer, which transforms the trees from the two treebanks into AB derivation trees. Basics of transducers can be found in [Co1], and specifications of the *G*-transducer that we use is explained in [SM1, Sa1]. Figure 2 shows an example of the output of the transducer when given the tree of figure 1. Less than 1.650 transduction rules are needed to convert 92% of the French Treebank (93% for Sequoia), what we created by ourselves.

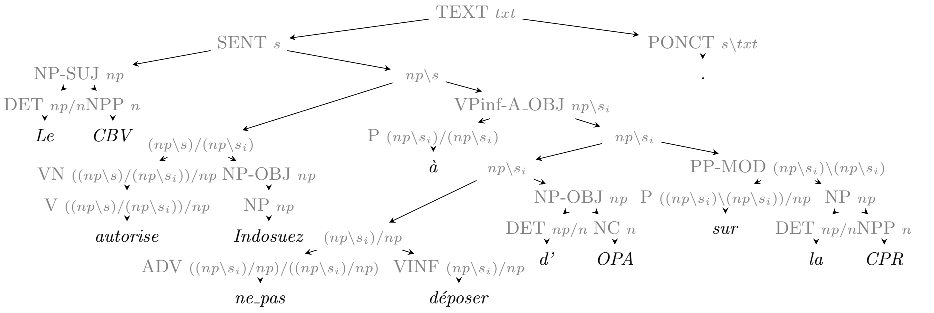


Fig. 2. Output of the transducer. The information from the syntactic tree remains. Regarding the real output, we suppress the type on the leaves, which is inherited from the pre-terminal nodes.

The transducer uses a small number of atomic types for the transduction: *np, n, s, txt, pp, cs,* cl_r, which correspond respectively to: noun phrase, noun, sentence, text (a sentence with an ending punctuation), a prepositional phrase, a subordinate clause and a reflexive clitic. In addition, the types $np\backslash s_p$ and $np\backslash s_i$ are given for past participle and the infinitival groups. However, the transducer is quite modular: one can create a new set of rules with new atomic types and the transducer will apply them so long as the rules create binary trees and the types stay coherent according to the elimination rules of Lambek calculus.

We will use these types to initialize our trees before the unification step; however, the input format will be described in section 3.

3 Grammar induction

A well known grammatical induction algorithm for rigid AB grammars is described by Buszkowski and Penn [BP1, Bu1]. When it comes to learning a rigid grammar, it is quite simple: either the types can be unified to get one type by word, or the algorithm fails. The algorithm created by Kanazawa [Ka1], learns a k-valued grammar. The two algorithms, however, need the same input format. Figure 3 shows an example. For a k-valued grammar, deciding which of the types to unify is more complicated and the best solution for a grammar can generally only be decided globally. As a consequence, grammar inference is know to be NP-Hard [CF1] when $k \geq 2$. Plus, k is generally unknown in advance.

To illustrate the unification problem, we will take two sentences from the French Treebank: *Le gouvernement n'avait ni écrit ni choisi cet accord dont nous avons hérité* (The government did not write nor choose this agreement we inherited) and *Nous avons de plus en plus de monde dans les DOM-TOM* (We have more and more people in the DOM-TOM). By applying the Buszkowski and Penn algorithm, the verb *avons* will get two types which look alike: $(a\backslash b)/c$ and $(d\backslash e)/f$, because in each case it takes two arguments, the first one on its left and the second one on its right. However, we cannot unify *avons*, because the right argument is a noun (*de plus en plus de monde*) or a past participle(*hérité*), and the two must not have the same type to avoid non grammatical sentences. We need for these two sentences at least a 2-valued grammar, and *avons* needs to get the two types: $(np\backslash s)/np$ and $(np\backslash s)/(np\backslash s_p)$.

Between these standard inference algorithms and ours, there are two main differences.

The first one is that we use some of the types from the treebank annotations, as summarized in table 2. The types assigned to nodes have been chosen according to their frequency in the lexicon extracted from transduced trees, with a cutoff corresponding to 75%: indeed, over 75 percent we assume that the type is clearly predominant and can be given when the POS-tag and the word is used as an argument. If a label is not in this table, its type will be set to a variable (if it is an argument). If the node is a functor, its type is instantiated simultaneously with the one of its argument. The input trees are then specified derivation trees where some sub-formulas contain free variables. Grammatical inference consists

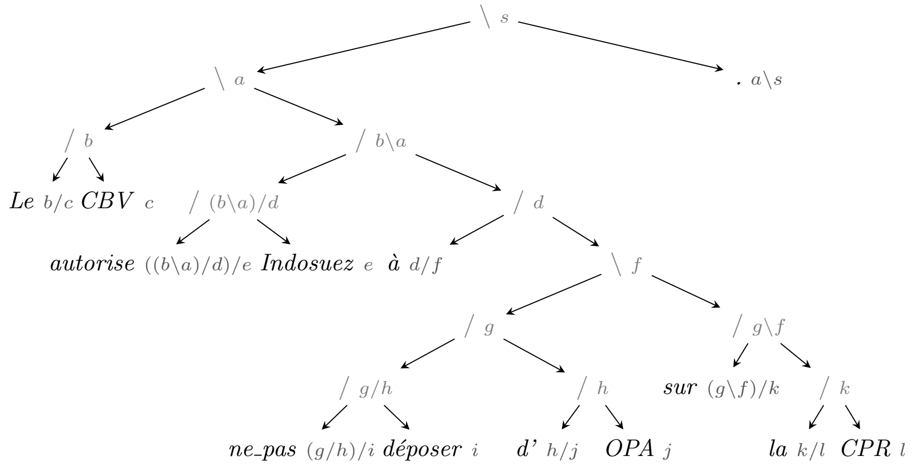


Fig. 3. Example input for the Buszkowski and Penn algorithm

of transforming these partially specified derivation trees into complete derivation trees. An example of our input trees is shown in figure 4. We can note that some words, even if their POS-tags are not in table 2, already have a complex type without variable.

Table 2. List of types assigned to the nodes that have the right label, if and only if they are arguments.

label	**type**	**label**	**type**
TEXT	txt	SENT	s
NP	np	NP-ARG	np
PP	pp	AP-ARG	n\n
CLS	np	CLS-SUJ	np
NC	n	NPP	np
VPP	np\ s_p	VINF	np \ s_i

The other difference is the use of clusters to guide the unification step, as discussed below. A cluster groups words by similarity between their vectors. At each cluster level, we apply a unification step. We choose a priority order[2] according to the clusters, level by level, which can be summed up by :

1. unify the smallest clusters,
2. unify the clusters where there is only one choice per variable,
3. unify with the simplest candidate (the complexity of a candidate is computed with the number of slashes \ and / it has),

[2] The steps are sorted by accuracy for the effectiveness of the unification.

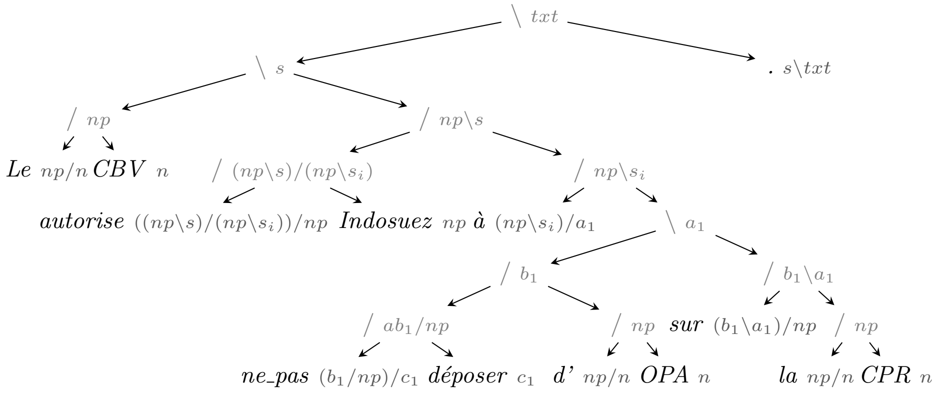


Fig. 4. Input tree of our inference algorithm. Some types remain from the previous tree, others are replaced by variables.

4. select the first found for other variables, but with the possibility to randomize the unification. Unfortunately, after the third step, a variable may have more than one possible unification and we cannot yet define which one is the best one. In this case, we use an arbitrary unification. Two methods can be applied, the "first-found" one, in the list of all possible unification (so it always gives the same result) or we can randomly pick up one possible unification (the results are never the same, but sometime they are better).

If variables remain in the types, because all the words may not be represented at a level, we proceed to a new cluster level. This way of processing first unifies the variables which are the most similar.

Even with variables, the derivation tree remains a valid AB derivation: the two elimination rules are the only ones used to create the trees.

4 Clustering

4.1 Vector extraction

We have decided to assign vectors to words, extracting informations from the treebanks before transduction.

The vectors have six dimensions :

1 POS-tag of the word (mother)
2 Syntactic node (grand mother)
3-4 POS-tag of the left node and the right one.
5-6 distance from the nearest common ancestor with the left neighbor node and the right one.

$$le_1 < \text{DET, NP-SUJ}, NILL, \text{NC}, -5, 2 >$$
$$le_2 < \text{DET, NP-MOD, VPP, ADJ}, 3, 2 >$$

Fig. 5. Two vectors corresponding to the determiner *le* (the).

If there is no left or right neighbor (first or last word of a sentence), the value of corresponding coordinate of the vector is set to $NILL$ or -5, depending on whether it is a label or a number. Two examples of vectors are shown figure 5.

To compare the vectors, we need to transform them into vectors in $\mathbb{Z}^n, n \in \mathbb{N}$. We choose to transform each label in a vector, where one or two dimensions have the value 1, and the others are equal to 0. The POS-tags and the syntactic data are transformed this way, and the distance from the nearest common ancestor stays, like shown in figure 5. The transformation is illustrated in table 3 with only a part of all the data. Of course, there is a "dimension" for almost all POS-tags (some exceptions are the ones we want to see unified together, like *ET* for the foreign words and *NPP* for the proper nouns); for the syntactic information, we only make a distinction between an argument (represented by the *-SUJ, -OBJ, -ATS...* at the end of the label) and a modifier *-MOD*. The *P+D* POS-tag correspond to a preposition which gathers a preposition and a determiner (”du” or ”au” instead of ”de le” or ”à le”). That's why the coordinates corresponding to determiner and preposition are set to 1.

Table 3. Example of vector transformations.

POS-tag	**NC**	**DET**	**P**	**...**
NC	1	0	0	0...0
DET	0	1	0	0...0
P+D	0	1	1	0...0
Other	**NP**	**...**	**-ARG**	**-MOD**
NP	1	0...0		
NP-SUJ	1	0...0	1	0
NP-MOD	1	0...0	0	1

The transformed vectors can be used with R [IG1] without further transformations.

4.2 Creation of clusters

To compute the hierarchical cluster, we use the software R [IG1], the Manhattan distance metric and for the clustering itself Ward's minimum variance method [Wa1]. For a better overview on the whole cluster, we use Tulip [AM1], which gives some graphs like the figure 6. The details of the cluster graph will be shown in the next section.

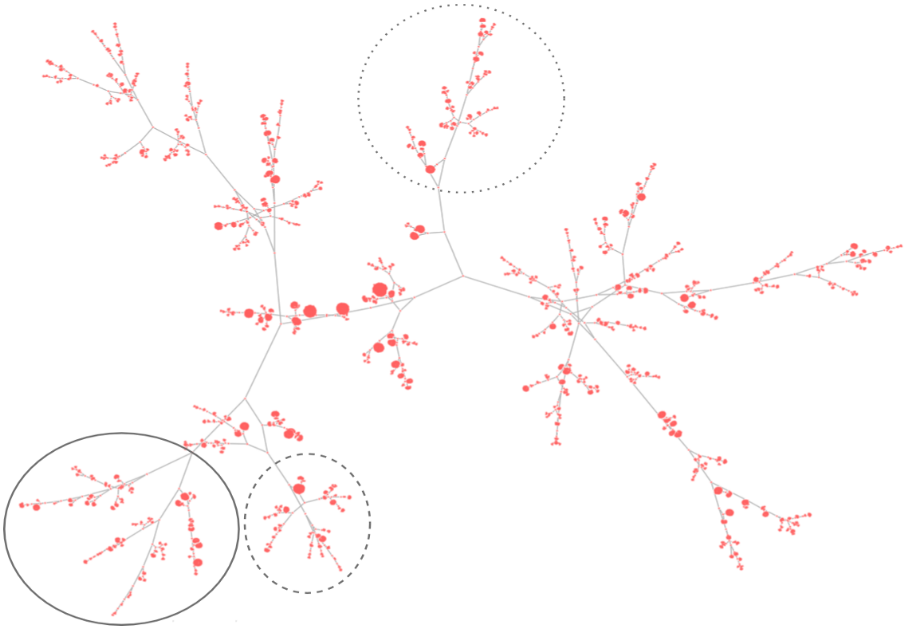


Fig. 6. Cluster corresponding to our test-set sentences. The part of the cluster circled by dots is where most of the verbs are; the one dashed corresponds to the determiners and the simply circled to the adjectives. We notice that the adjectives and the determiners are close to each other.

5 Evaluation

Before evaluating our method, we want to give some baselines for it :

– If we unify the types only when there is one solution, it leaves a lot of pairs word-type with variables: around 80%. However, there is a lower number of variables, due to the unification (a ratio of 47 %).
– If we only use the first found method or the all random method to unify clusters, no variable remains, but the comparison with the gold-standard lexicon is really bad (less than 50% of identical types).
– If we try to unify without the clustering step, the method does not take into account the context of words, and becomes equivalent to trying to reduce the lexicon of each word, which is not the point of our work: we would rather have the same type for different words which occur in same context.

We ran our method over 754 sentences of Sequoia, and a set of 553 sentences from the French Treebank. We use a random method to chose de sentences. The table 4 shows the effectiveness of the method, computed on the percentage of variables remaining after unification.

Table 4. Remaining variables after the various unifications. The two first baselines have been tested with the 553 sentences of the French Treebank.

Treebank	754 Sequoia	553 FTB	One-solution unification	First-found method
not unified	3	0	364	0
number of var.	1.429	686	686	686
rate	99,8%	100%	46,9%	100%

Nevertheless, we cannot judge the effectiveness of our method only over the remaining variables. Indeed, if the variables are unified but the result types are overly complex, we cannot say that our method performs well, even though all sentences are derivable. On the lexicons extracted from the transduced trees and the unified words over the 754 sentences of Sequoia, we can note that the lexicon similarity is 82,7%: this means that only 2.967 word-type pairs are different over the 17.110 pairs of the lexicons. It corresponds to 1.012 words forms.

We focused our study on the 553 sentences of the French Treebank. We compare the lexicon extracted from the unified trees and from the transducer, which corresponds to a lexicon of 2.076 different words, with 4.808 associated types for 5.731 total occurrences.

The differences between the two lexicons are about 329 words, and it corresponds to only 859 word-type pairs, that is 14,9% of the lexicon. It means that 85,1% of the lexicons are identical. We count as identical the modifications like np which become a n or the inversion between a pp and pp_{de} or pp_a (the three correspond to prepositional phrases, but the last two add the information that the preposition is a or de), but these cases are really rare (less than 2% of the lexicon).

Table 5 sorts the differences in three categories: the errors, the similar pairs and the identical pairs.

We can give two examples of these categories :

The past participle *accumulé* (accumulated) gets the type $np\backslash s_p$ from unification and $n\backslash n$ from the transducer. The type given by the unification corresponds to a VPP used as a past participle and not as an adjective, but in the CCG Bank [Ho1] a special rule permits the translation of $np\backslash s_p$ into $n\backslash n$, so treating the two as similar for the purpose of this evaluation seems justified.

The type of *change* (change) gets the type $np\backslash s$ instead of $(np\backslash s)/np$. It is a real error: instead of being treated as a transitive verb (which takes two arguments), it is treated as an intransitive verb. The error comes from the clustering step, where *change* is near an intransitive verb.

However, it is important to remember that even with the inconsistencies, the sentences can still be parsed with the types given by unification.

Figure 7 shows two level 0 clusters. The left one is a cluster of adverbs. The variable ab_{331} will be unified with np. The right cluster only has variables that will be unified together. This cluster gathers adjectives and participles used as an adjective.

Table 5. Ratio between the differences, count in words. Even for the baselines, the ratio is computed over the whole lexicon.

Unification + clustering	identical	4 832	85,1%
	similar	5 168	91,3%
One-solution unification	identical	728	12,7%
	similar	1 164	20,3%
First-found method	identical	2 745	47,9%
	similar	3 576	62,4%

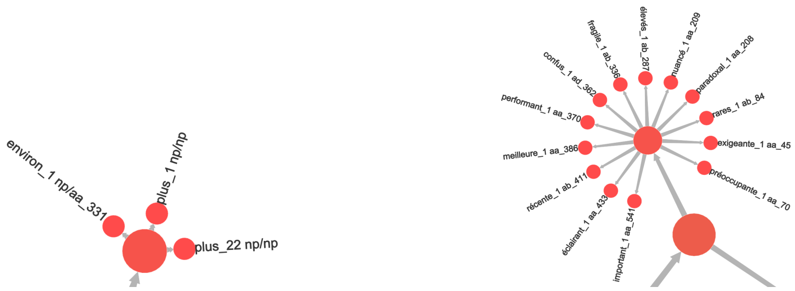


Fig. 7. Focus on two clusters of level 0.

6 Discussion

The method we use is designed to run with derivation trees: we use unsupervised learning, but with very informative input structures including some syntactic information. However, it could be extended to all sentences of the french language with some modifications, using together the raw sentences and the derivation trees.

The problem is to get vectors from sentences that do not have a syntactic tree. We could use vectors in a subspace, because some informations, like the POS-tags of words, can easily be retrieved, from a tagger like the one of Moot [Mo1]. Another solution is to retrieve the structures using the Stanford Parser [GM1] and then apply our algorithm without modifying anything.

Then, we can cluster these vectors with other ones, making a projection to lower the number of dimensions. This way, the words can have the most used type of a level 0 cluster. It enables us to have a better overview on the words than if we just gave them the most used type in the lexicon, corresponding to their POS-tag. Given that our vectors are high-dimensional and sparse, we could also apply the method described by Kailing et al. [KK1] to manage them.

6.1 Scaling up to Larger Treebanks

We plan to apply the current method to larger datasets, but we will have to deal with much larger clusters for Sequoia (more than 63.000 words) or the French Treebank (around 300.000 words). Though the complexity remains polynomial,

$O(n^3)$ would still be problematic for these larger datasets. Note, however that there is a significant improvement over other learning algorithms since k-valued grammars have an exponential complexity.

6.2 Optimized type unification

For the moment, we use the “first found” criterion to unify variables when there is no other choice criterion. A better solution would be to look at all the variables, to assign them a list of possible unifications and to use the Hungarian Algorithm [Ku1, Mu1] to choose the unification which produces the set of formulas which is the smallest in terms of the total sum of the connectives after unification.

7 Conclusion

In this article, we have shown a new method to extract an AB grammar by unification, using clusters to guide us. Some proposition of improvement for this work, focused on the treatment of coordinations and modifiers are discussed in the article [Sa3]. As explained in section 6, it opens up many work directions.

We decided to use hierarchical clustering which enabled us to unify the lexicon step by step, either until convergence or, very exceptionally, until a conflict blocks further unification. However, it would be interesting to test our method with other types of clustering and distance metrics, like the *k-means* method or the one called “Clustering By Committee”, described in [Pa1]. This method searches for the best centroids of clusters, which are supposed to be representative of their clusters; but it cannot be applied here in its present state, because we want to make a unification after the clustering, and the types of the centroids are not defined.

The results we have are promising, since in spite of the fact that the input is less detailed (and therefore requires less annotator hours) they are near our gold standard: we have 91,3% of the lexicon similar, and 85,1% identical.

References

[AC1] Abeillé, A., Clément, L., Toussenel, F.: Building a treebank for french. Treebanks : Building and Using Parsed Corpora p.165–188 (2003)

[AC2] Abeillé, A., Clément, L.: Annotation morpho-syntaxique (2003)

[Ad1] Adriaans, P.W.: Learning shallow Context-Free languages under simple distributions. Algebras, Diagrams and Decisions in Language, Logic and Computation, CSLI/CUP (2001)

[Aj1] Ajdukiewicz, K.: Die syntaktische konnexität. Studia Philosophica (1935)

[AM1] Auber, D., Mary, P.: Tulip: Better visualization through research (2007)

[Ba1] Bar-Hillel, Y.: Language and information: selected essays on their theory and application. Addison-Wesley Pub. Co. (1964)

[Bu1] Buszkowski, W.: Solvable problems for classical categorial grammars. Bull. Pol. Aca and Scle. Math. p.373–382 (1987)

[BP1] Buszkowski, W., Penn, G.: Categorial grammars determined from linguistic data by unification. Studia Logica, p.431–454 (1990)

[CS1] Candito, M., Seddah, D.: Le corpus Sequoia : annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical TALN'2012 proceedings p.321–334 (2012)

[Co1] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (2007)

[CF1] Costa-Florêncio, C.: Consistent identification in the limit of any of the classes k-valued is np-hard. LACL 2001 proceedings p.125–138 (2001)

[GM1] Green, S. and Marneffe, M.C. and Bauer, J. and Manning, C.D. : Multiword Expression Identification with Tree Substitution Grammars: A Parsing tour de force with French. EMNLP proceedings p.725–735 (2011)

[Go1] Gold, E.: Language identification in the limit. Information and Control (1967)

[Ho2] Hockenmaier, J.: Data and models for statistical parsing with combinatory categorial grammar PhD Thesis (2003)

[Ho1] Hockenmaier, J., Steedman, M.: CCGbank: a corpus of CCG derivations and dependency structures extracted from the penn treebank. Computational Linguistics p.355–396 (2007)

[IG1] Ihaka, R., Gentleman, R.: R project (1993)

[KK1] Kailing, K., Kriegel, H., Kröger, P.: Density-connected subspace clustering for high-dimensional data. Proceedings of the Fourth SIAM International Conference on Data Mining p.246–257 (2004)

[Ka1] Kanazawa, M.: Learnable Classes of Categorial Grammars. Center for the Study of Language and Information, Stanford University (1994)

[Ku1] Kuhn, H.: The hungarian method for the assignment problem. Naval Research Logistics Quarterly p.83–97 (1955)

[La1] Lambek, J.: The mathematics of sentence structure. The American Mathematical Monthly **65** p.154–170 (1958)

[Mo1] Moot, R.: Wide-coverage semantics for spatio-temporal reasoning. Traitement Automatique des Langues **52** p.1–28 (2012)

[Mo2] Moot, R.: Semi-automated Extraction of a Wide-Coverage Type-Logical Grammar for French Traitement Automatique des Langues proceedings (2010)

[Mu1] Munkres, J.: Algorithms for the assignment and transportation problems. Journal of the Society for Industrial and Applied Mathematics p.32-38 (1957)

[Pa1] Pantel, P.: Clustering by committee. PhD thesis (2003)

[SM1] Sandillon-Rezer, N.F., Moot, R.: Using tree tranducers for grammatical inference. Proceedings of Logical Aspects of Computational Linguistics (2011)

[Sa1] Sandillon-Rezer, N.F.: Learning categorial grammar with tree transducers. ESSLLI Student Session Proceedings p.175–181 (2011)

[Sa2] Sandillon-Rezer, N.F.: Ygg, parsing French text using AB grammars. LACL demo session p.17–20 (2012)

[Sa3] Sandillon-Rezer, N.F.: Clustering pour grammaires catégorielles du second ordre. Mixeur proceedings p.88–91 (2013)

[TA1] Trautwein, H., Adriaans, P., Vervoort, M.: Towards high speed grammar induction on large text corpora. SOFSEM '00 proceedings p.173–186 (2000)

[Wa1] Ward, J.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association p.236–244 (1963)

Constructing a Phenomenal Corpus: Towards Detecting Linguistic Phenomena in Precision Grammars

Ned Letcher and Timothy Baldwin

Department of Computing and Information Systems,
The University of Melbourne
ned@nedletcher.net, tb@ldwin.net

Abstract. Precision grammars and their accompanying treebanks are valuable linguistic resources, but due to their inherent complexity as well as differences in underlying theoretical approaches and differing styles of implementation between grammar engineers, the discoverability of linguistic phenomena is poor. In this paper, we describe the methodology used for the manual annotation of a small syntactic phenomenon corpus of English. When scaled up to a larger corpus, this resource will be used to link structures in parser output — and thus grammar internal components — to the linguistic phenomena whose analysis they pertain to. This resource will also provide an opportunity for phenomenon-specific parser evaluation.

Keywords: Precision grammar, grammar engineering, linguistic phenomenon detection, discoverability

1 Introduction

Precision grammars are machine readable grammars of natural language which aim to accurately model the linguistic constraints of specific languages. The linguistic fidelity of precision grammars, combined with their ability to process large amounts of naturally occurring language, means that these grammars and their respective treebanks constitute a significant opportunity for the exploration of linguistic phenomena (Bender, 2008). However, identifying components of grammars responsible for implementing specific phenomena can be problematic due to the interaction between linguistic phenomena and difficulties in their documentation. The problem this paper is concerned with is the discoverability of linguistic phenomena within precision grammars. By *linguistic phenomena*, we mean the components of language that linguists concern themselves with when describing language and that are amenable to structural analysis by a formal grammar.

In this paper, we argue that precision grammars would be more valuable linguistic resources if the linguistic phenomena they cover were more discoverable, and additionally, that it would be desirable to have automated approaches for achieving this. We present a proof-of-concept linguistic phenomenon corpus of English, annotated independently of existing grammar formalisms, and outline the methodology used in its construction. By comparing parser output of items from this corpus with phenomenon annotations, this corpus will provide a means by which grammar internal components can be linked with the phenomena they are associated with. Additionally, this resource will also facilitate fine-grained phenomenon-specific parser evaluation.

2 Background

Since precision grammars are both machine readable and designed with the express purpose of modelling the linguistic constraints of language, prima facie it might seem that the identification of linguistic phenomena within precision grammars would be straightforward. The reality is far from this. In the rest of this section, we outline the difficulties involved in pinpointing components of precision grammars which pertain to specific phenomena and the factors which contribute towards this.

2.1 One Does Not Simply `grep` for Linguistic Phenomena

In the context of someone wishing to determine whether a precision grammar covers a particular linguistic phenomenon, and if it does so, which components of the grammar constrain the analysis of the phenomenon, a number of potential strategies can be employed.

One approach is to use a string-based search tool such as `grep` to search grammar source files and derivation trees produced by the grammar. This strategy is premised upon the assumption that names assigned to components of the grammar and node labels in derivation trees are likely to correspond to the phenomena whose analyses they constrain. While this assumption may be reasonable, there are a number of reasons why this approach will fail. This first is due to pragmatic constraints — it is often simply not possible to encode the complete names of phenomena as this would result in unwieldy source code and derivation trees. For instance, the LinGO English Resource Grammar (Flickinger, 2002) uses the name `v_cp_le` as the lexical type which is a verb that takes a complementizer phrase as an argument. Without knowing this abbreviation in advance however, the type would not be readily retrievable. This clearly motivates the maintaining of rigorous documentation, however, even in the presence of this, there are still significant barriers to phenomenon discovery.

Another problem with string-based searches is that it is not always clear what to search for. In descriptive linguistics, there are many different terms that have come to be used to describe different linguistic phenomena, many of which have near or sometimes exact synonyms. For instance, the terms *subordinate clause*, *dependent clause* and *embedded clause* are used throughout the literature. Adopting a standard for linguistic description — such as GOLD (Farrar and Lewis, 2005) — would likely ameliorate this difficulty. A limitation of GOLD, however, is that in aiming to be applicable across different languages and different formalisms, it focuses on capturing high-level phenomena such as sentential force and main versus subordinate clauses, whereas more nuanced phenomena such as raising and control constructions and ellipsis are not handled.

Phenomena may also not be identifiable via keyword search due to the analysis found in the grammar differing from the expected one. A cursory glance through the linguistic typology literature will reveal that linguistic phenomena are often subject to multiple competing analyses. When implementing an analysis of a phenomenon in a given language, a grammar engineer might need to select between a number of such analyses. For example, the passive voice can be analyzed in a number of different ways, including as a lexical rule, a raising-type construction or a gapping-type construction. Once a particular analysis has been selected, documentation and grammar components will refer to terminology pertaining to this analysis, and thus plausible queries for the phenomenon could fail. These two intertwined issues of competing nomenclature and analyses are further exacerbated when dealing with cross-formalism investigation. Precision grammar implementations exist for multiple grammar frameworks, such as LFG (Maxwell and Kaplan, 1996), HPSG (Copestake and Flickinger, 2000), and CCG (Baldridge and Kruijff, 2003). Each of these bring their own theoretical presuppositions to the table, increasing the potential for failed queries.

Even in cases where string-based search approaches do succeed in identifying relevant grammar components, there is a good chance that the results will not be exhaustive. This arises in part from the inherent complexity of precision grammars, with the analysis of many phenomena involving constraints across various grammar subsystems. Additionally, linguistic phenomena and their respective analyses within grammars interact with each other (Bender, 2008; Fokkens, 2011). When extending a grammar, it is possible that the analysis of a new phenomenon will interact poorly with analyses of existing phenomena, requiring additional constraints on superficially unrelated components. Tracking which phenomena these constraints were intended to pertain to becomes difficult. An example of this situation is discussed in Bender (2008), where a V2 analysis of word order in Wambaya interacts with the analysis of secondary predicates.

An alternative to string-based search approaches is to use a treebank search tool such as `TGrep2` (Rohde, 2005) or Fangorn (Ghodke and Bird, 2010). The problem with this approach is that it assumes some knowledge of how the phenomenon is likely to be represented in a treebank — and thus also of how it is analyzed within the grammar — which introduces a circularity into the discovery process.

2.2 Other Pitfalls when Searching for Linguistic Phenomena

An additional problem with the use of precision grammar derived-treebanks is not knowing in advance whether the treebank contains items that exhibit the phenomenon. Another possibility, then, is to manually parse items known to exhibit the phenomenon and inspect the contents of the derivation trees. Aside from being laborious, this approach runs into some significant problems.

Firstly, there may be other phenomena in the sentence that are unhandled by the grammar, thus blocking a successful parse and resulting in a false negative. The grammar might also overgenerate for this sentence, resulting in a false positive for the phenomenon. The grammar could also have an analysis of the phenomenon, but fail to parse the sentence due to the analysis not being general enough. There is also the problem of identifying the correct parse from the parse forest, given that the nature of the analysis in the grammar (if there is one) is unknown.

In addition to all of the aforementioned issues, there is also significant heterogeneity to be found within the architecture of precision grammars. Even within two grammars built using the same formalism and grammar engineering toolchain, it is possible to see dramatic differences in implementation arising from differing theoretical assumptions, engineering decisions or just general differences in grammar engineering styles, potentially making the navigation of previously unseen grammar source code difficult, even for grammar engineers experienced in the formalism. This can be seen in the Norsyg Norwegian Grammar,[1] which uses the same formalism and grammar development environment used within the DELPH-IN consortium, but differs markedly from other DELPH-IN grammars in that the argument structure of predicates is determined through syntax rules, rather than being specified in the lexicon (Haugereid, 2004).

The result of all this is that for most precision grammars, without prior experience with their internal components, it is usually not possible to quickly ascertain whether a precision grammar covers a particular phenomenon, nor to determine which components are involved in its handling.

3 A Phenomenal Vision

There has been significant work in the development of phenomenon-centric grammar engineering practices, such as the LinGO Grammar Matrix (Bender et al., 2010), CLIMB (Fokkens et al., 2012), and the ParGram project (King et al., 2005). These different approaches all try to build into the grammar engineering process strategies for identifying linguistic phenomena.

The ParGram project is distinct from the other cited projects, in that its focus on linguistic phenomena occurs towards the end of the grammar engineering pipeline, by harmonizing parse results from grammars of different languages so that phenomena are represented similarly. The LinGO Grammar Matrix involves the creation of a core module containing analyses of linguistic phenomena purported to be universal to all languages, and the curation of a set of libraries of linguistic phenomena that languages may or may not make use of. The Customization System provided by the Grammar Matrix enables users to select from these libraries with appropriate parameters to generate fully-functioning starter grammars, removing much of the initial overhead involved in grammar creation. CLIMB extends the Grammar Matrix's Customization System to provide a metagrammar engineering methodology for the ongoing development of

[1] `http://moin.delph-in.net/NorsygTop`

grammars which is analysis-centric, allowing constraints not normally contiguous in the grammar to be grouped together.

Rather than altering the process by which grammars are developed, our proposed approach treats existing grammars as fixed resources from which we try to automate the detection and location of constraints that correspond to specific phenomena. We propose to use parser output known to exhibit specific linguistic phenomena to link those phenomena with the corresponding grammar internal components which constrain them. We plan to do this by learning corresponding "signatures" from within data structures produced by the parser that are characteristic of specific phenomena. These signatures could take the form of flat clusters of node labels or commonly-occurring subtrees from the derivation trees produced by the parser, as well as patterns within other data structures produced by the parser. In order for this to work, however, what is first required is a corpus annotated with linguistic phenomena.

Bender et al. (2011) describe a method for semi-automated detection of phenomena within a treebank. This was performed by manually identifying lexical types and construction rules found in derivation trees considered characteristic of the phenomena under investigation. These were used to generate candidate items which were then vetted to verify the occurrence of the phenomenon. This approach has the advantage of being able to rapidly identify many candidates, but is limited by the fact that it can only be used to locate phenomena covered by the grammar, and biases results towards the analysis found in the grammar. Our approach differs in that we perform manual annotation of phenomena independently of any grammar, ensuring that we do not bias phenomena towards analyses found in specific grammars, nor exclude instances of phenomena not yet implemented.

As far as we are aware, there are no large-scale treebanks specifically annotated with linguistic phenomena. In the remainder of this section we outline some specific motivations for developing such a resource.

3.1 Incentives for Building a Phenomenon Corpus

Enhancing Descriptive Grammars Bender et al. (2012) argue that electronic descriptive grammars can be enhanced through augmentation with treebanks produced by precision grammars. These treebanks provide a ready source of exemplar items exhibiting the phenomena described in the reference grammar, which can be recalled via the use of a treebank search tool. Reference grammars then become more dynamic, with the number of potential exemplars increasing as the treebank grows in size. This approach requires authors to provide canned queries to retrieve appropriate exemplars. Automated phenomena detection could potentially relieve this burden; furthermore, as suggested by Bender et al., this could also allow users to navigate from arbitrary structures within exemplar items back to the relevant parts of the descriptive grammar.

Grammar Re-use The grammar engineering process involves a considerable amount of time. When making a contribution to a precision grammar, not only is confirmation required that changes have had the intended effect, but also that changes did not introduce regressions such a drop in coverage, loss of accuracy of existing analyses, or the introduction of spurious ambiguity. In the context of implementing a new phenomenon in a grammar, it would be desirable for a grammar engineer to be able use existing grammars which contain tried-and-tested implementations of the relevant construction as inspiration. With a means to detect the handling of a phenomenon by a grammar, and identifying the components involved in its implementation, it would be possible to develop a search tool that would allow grammar engineers to much better exploit the knowledge contained within precision grammars.

Phenomenon-based Parser Evaluation In addition to the long-term goals of phenomenon detection, this corpus would also have more immediate applications. One in particular is that of phenomenon-based parser evaluation, in the style of Bender et al. (2011). This more nuanced form of parser evaluation is able to pinpoint specific gaps in parser coverage that would otherwise be hidden by coarser, more commonly-used parser evaluation approaches. Bender et al. found that all the parsers they examined struggled to correctly identify many frequently-occurring non-local dependencies. One notable difference between the approach proposed in this paper and that of Bender et al. is that we identify occurrences of different phenomena via lexical spans in a corpus, where Bender et al. represent constructions via their component "dependencies". While there are certainly merits to a dependency-based representation, they tend to require noisy mappings from the native outputs of individual parsers (Clark and Curran, 2007); lexical spans, on the other hand, are a natural representation for all grammar formalisms.

Bootstrapping the Phenomenon Corpus Another application of the proposed signature extraction approach is the automatic annotation of linguistic phenomena, inspired by the approach of Bender et al. (2011). We plan to investigate this as a means of rapidly scaling up the size of the phenomenon corpus. We suspect that this approach has the potential to yield accurate phenomenon annotation, but will be limited in that it will likely result in non-exhaustive annotation of phenomena, as particular manifestations of phenomena not captured by the signatures will not be identified.

4 Methodology

In this section we describe the methodology used for the construction of a proof-of-concept phenomenon corpus. The initial stage involved identifying the desiderata for the properties of such a resource. This is outlined below.

1. **Grammar engineering framework independent:** Annotations need to be represented independently of specific grammars and frameworks. This

ensures we do not bias phenomena toward analyses found within existing grammars and that the corpus can be applied to grammars from different frameworks.

2. **Exhaustively annotated:** Each item in the corpus needs to be exhaustively annotated so that the non-occurrence of a phenomenon can be used as negative data. An item not containing a phenomenon under investigation is a useful data point insofar as it suggests that structures involved in the parse are less likely to be associated with the phenomenon.
3. **Token-level:** Each phenomenon occurrence in a sentence should be annotated at the word token level to yield tighter associations between phenomena and substructures produced by the parser.
4. **Disambiguated:** By basing the corpus on a manually disambiguated treebank produced by a precision grammar, the parse for each item will reflect the best parse the grammar can yield, increasing the likelihood of phenomena being associated with relevant components of the grammar. To be in keeping with the first desideratum, there would ideally be treebanks of the same corpus produced in multiple grammar engineering frameworks.

4.1 Phenomena

The notion of *linguistic phenomena* is hard to pin down. Rather than attempt to rigorously define it, we have opted for an operational characterization that is compatible with the use of the term in the motivating contexts described in Section 3.1. Since we want to leave the door open for cross-linguistic comparison, we looked for phenomena that are found across many of the world's languages. Instead of looking for simple morphosyntactic phenomena such as case, tense and gender, we chose more complex phenomena that would likely involve the interaction of various simple phenomena, and thus be more likely for their implementation to involve constraints across different parts of the grammar.

As a part of the process of selecting the five different phenomena, we consulted the typological literature to determine the range of languages the phenomena are known to be found in, as well as to develop a set of criteria for annotators to use to identify instances of the phenomenon. Rather than trying to account for the full variability of each phenomenon across all the languages of the world, we attempted to find a simple set of criteria that is sufficiently broad to capture a range of different manifestations of the phenomenon, while also providing a simple-to-use and clear guide to prevent annotators from becoming quagmired in analysis. For this preliminary version of the corpus, we chose phenomena that take the form of simple constructions spanning either matrix or subordinate clauses. The five phenomena selected were: (1) the passive voice, (2) interrogative clauses, (3) imperative clauses, (4) relative clauses, and (5) complement clauses. For further details regarding the chosen phenomena along with the criteria developed for their identification, please see the annotator guidelines document.[2]

[2] http://repository.unimelb.edu.au/10187/17611

These phenomena are also relatively broad, encompassing other finer-grained categories of phenomena such as subject raising relative clauses and impersonal passives. It is likely that users of precision grammars will be more interested in these more fine-grained categories. The problem is knowing how fine-grained to go — how narrowly to delineate each phenomenon and carve-up the linguistic phenomenon space? A particular concern here is that, as outlined in Section 2, different grammars will divide the space in different ways due to typological variation and differing theoretical approaches. It is for this reason we believe it makes more sense to prioritize cross-grammar applicability in the corpus construction stage. This does not necessarily preclude the identification of finer-grained phenomena however, as it is foreseeable that this could be achieved in the phenomenon detection stage through the use of clustering techniques. Furthermore, finer-grained distinctions will likely involve increased annotation time costs and, if successful, this approach has the advantage that it automates this more costly analysis.

4.2 Constructing the Corpus

The purpose of this preliminary proof-of-concept corpus was to test out and fine-tune the criteria for phenomenon identification in the annotation guidelines and produce an exportable format for distribution. The corpus was composed of 200 lines of *The Speckled Band*, a Sherlock Holmes novel by Arthur Conan Doyle. This text was chosen for a number of reasons: (1) it is out of copyright, and can thus be shared freely; and (2) it is syntactically highly rich and varied, and more likely to shed light on potential problems with the annotation guidelines. As discussed in Section 5, we plan to use existing treebanks produced by precision grammars developed within the DELPH-IN consortium[3] as a basis for the large-scale phenomenon corpus; thus, our methodology targets the format used for storing treebanks in the DELPH-IN grammar engineering toolchain.

For the annotation process, we used `brat`,[4] a browser-based rapid annotation tool (Stenetorp et al., 2012). Annotators were instructed to exhaustively annotate each sentence with occurrences of the five phenomena by labelling character spans they deemed to display the phenomena. The annotations were then converted into the format used by `[incr tsdb()]`, the grammar profiling and treebanking tool used in the DELPH-IN consortium (Oepen and Flickinger, 1998). As a part of its relational database used to store test suite profiles, `[incr tsdb()]` conveniently supports the recording of multiple phenomenon occurrences for a given item and it is trivial to augment existing `[incr tsdb()]` profiles with phenomenon records.

After an initial round of annotation of 50 sentences by two different annotators with training in syntactic theory, the results were used to fine-tune the annotator guidelines by making the instructions clearer and eliminating identified ambiguities. A second round of 50 lines was completed by both annotators

[3] `http://www.delph-in.net`
[4] `http://brat.nlplab.org`

Phenomenon	Frequency	Agreement
Passive voice	25	0.85
Interrogative clause	16	0.94
Imperative clause	5	0.78
Relative clause	62	0.91
Complement clause	54	0.71

Table 1. Attested phenomenon occurrences within the 200 lines of Sherlock Holmes text and inter-annotator agreement for the 50 line subset.

using the updated guidelines and inter-annotator agreement was calculated using Fleiss' coefficient of agreement (Artstein and Poesio, 2008). Agreement was calculated on a binary basis for each phenomenon using word tokens, such that tokens occurring within an annotation span received a 1 if the annotation was of the type of phenomenon being calculated, and 0 otherwise. After the results were analyzed, the final 100 lines of the text were then completed by a single annotator.

4.3 Results

Table 1 shows the results of the annotation of the Sherlock Holmes text. The relatively low frequency of imperative clauses in the corpus highlights the sensitivity of some linguistic phenomena to specific domains. We would, for example, expect there to be few imperatives in a corpus of newswire text, but likely many in a corpus comprised of spoken language. While it will be important to select phenomena that are well represented in the final corpus, it will still be advantageous to include infrequently occurring phenomena, as this will provide an opportunity to determine how well our phenomenon detection techniques perform in the presence of data scarcity.

The inter-annotator agreement results show good reliability[5] for all phenomena with the exception of complement clauses. Investigation of disagreements for this category showed that there were a number of cases where one of the annotators had simply missed a complement clause, suggesting that this frequently occurring phenomenon is easier to overlook — a problem which should improve with increased vigilance and further practice.

5 Next Steps

This initial prototype corpus was developed as a proof-of-concept, to help refine the methodology of its construction. Now that it has been developed, we

[5] There is a general consensus in the computational linguistics community (Artstein and Poesio, 2008) that values of coefficients in the family of Fleiss' coefficient greater than 0.8 are said to show "good reliability" while greater than 0.67 allows for "highly tentative and cautious conclusions."

can look ahead to the next steps. Firstly, we will turn our attention to building a large-scale phenomenon corpus based on the DeepBank resource (Flickinger et al., 2012a). DeepBank is a dynamically annotated large-scale treebank of text from The Wall Street Journal sections of the Penn Treebank (Marcus et al., 1994). The annotations are produced by The LingGO English Resource Grammar (ERG), a broad-coverage precision grammar of English (Flickinger, 2002) developed within the DELPH-IN consortium. The treebank contains manually disambiguated parse results and is available in a number of representation formats. The existence of parallel treebanks developed with the same methodology for both Portuguese and Bulgarian via the ParDeepBank project (Flickinger et al., 2012b) leaves open the possibility for cross-linguistic phenomena exploration.

The linguistic phenomena selected for annotation in this corpus were intentionally chosen so as to be well known and frequently occurring constructions that are relatively uncontroversial in their characterization. In the large-scale phenomenon corpus, we will introduce additional phenomena with more subtle characteristics whose identification will likely involve more difficulties. Each phenomenon added will be subject to the same methodology as outlined in this paper to ensure reliability in the identification of phenomena.

After the creation of a larger phenomenon corpus, we will then be in a position to develop a phenomenon signature extraction technique which uses parser output from the ERG to link linguistic phenomena found in DeepBank to corresponding components of the ERG. While we initially plan to focus on the derivation trees produced by the parser, this approach could also be extended to use semantic output, as well as the contents of the parse chart — the data structure built up by the parser at parse time. This last source of data is significant in that it is available even for sentences that do not receive a spanning parse, potentially providing a means of detecting the presence of unimplemented phenomena. We also foresee the possibility of using the signatures to perform automatic phenomenon annotation — potentially providing a means of rapidly scaling-up the phenomenon corpus, albeit with “silver” rather than gold standard annotations. These automatically extracted annotations would be biased towards instances of phenomena handled by the ERG, however the spans themselves remain formalism independent, as they would be just flat character spans.

6 Conclusion

In this paper we have identified various incentives for developing techniques for detecting linguistic phenomena within precision grammars. We have argued that a necessary — but so far non-existent — resource for this goal is corpora annotated with linguistic phenomena independently of existing precision grammars. We also presented a proof-of-concept English phenomenon corpus — along with a methodology for its development — that aims to fill this gap. While currently only a small offering, we plan to produce a much larger version of the corpus with additional phenomena based on the DeepBank treebank. This resource will then be used to explore techniques for automatically extracting characteristic signatures of linguistic phenomena within parser output, providing a means of associating specific linguistic phenomena with relevant grammar components. In addition, this resource will also have more immediate applications, such as enabling fine-grained phenomenon-based parser evaluation.

We hope this work will ultimately lead towards improving the discoverability of linguistic phenomena within precision grammars by helping them wear their linguistic phenomena on their respective sleeves, thus increasing their utility as linguistic resources.

References

Artstein, R. and Poesio, M. (2008). Inter-coder agreement for computational linguistics. *Computational Linguistics.*, 34(4):555–596.

Baldridge, J. and Kruijff, G.-J. M. (2003). Multi-modal combinatory categorial grammar. In *Proceedings of The 11th Conference of the European Chapter of the Association for Computational Linguistics*, Budapest, Hungary.

Bender, E. M. (2008). Grammar engineering for linguistic hypothesis testing. In *Proceedings of the Texas Linguistics Society X Conference: Computational Linguistics for Less-Studied Languages*, pages 16–36, Austin, USA.

Bender, E. M., Drellishak, S., Fokkens, A., Poulson, L., and Saleem, S. (2010). Grammar customization. *Research on Language & Computation*, 8(1):23–72.

Bender, E. M., Flickinger, D., Oepen, S., and Zhang, Y. (2011). Parser evaluation over local and non-local deep dependencies in a large corpus. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 397–408, Edinburgh, Scotland, UK.

Bender, E. M., Ghodke, S., Baldwin, T., and Dridan, R. (2012). From database to treebank: On enhancing hypertext grammars with grammar engineering and treebank search. In Nordhoff, S., editor, *Electronic Grammaticography.* University of Hawai'i Press.

Clark, S. and Curran, J. (2007). Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 248–255, Prague, Czech Republic.

Copestake, A. and Flickinger, D. (2000). An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the 2nd International Conference on Language Resources and Evaluation (LREC 2000)*, Athens, Greece.

Farrar, S. and Lewis, W. D. (2005). The GOLD community of practice - an infrastructure for linguistic data on the web. In *In Proceedings of the EMELD 2005 Workshop on Digital Language Documentation: Linguistic Ontologies and Data Categories for Language Resources*, Cambridge, USA.

Flickinger, D. (2002). On building a more efficient grammar by exploiting types. In Oepen, S., Flickinger, D., Tsujii, J., and Uszkoreit, H., editors, *Collaborative Language Engineering*, pages 1–17. Stanford: CSLI Publications.

Flickinger, D., Kordoni, V., and Zhang, Y. (2012a). DeepBank: A dynamically annotated treebank of the wall street journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, Lisbon, Portugal.

Flickinger, D., Kordoni, V., Zhang, Y., Branco, A., Simov, K., Osenova, P., Carvalheiro, C., Costa, F., and Castro, S. (2012b). ParDeepBank: Multiple parallel deep treebanking. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, pages 97–108, Lisbon, Portugal.

Fokkens, A. (2011). Metagrammar engineering: Towards systematic exploration of implemented grammars. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, USA.

Fokkens, A., Avgustinova, T., and Zhang, Y. (2012). Climb grammars: three projects using metagrammar engineering. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey.

Ghodke, S. and Bird, S. (2010). Fast query for large treebanks. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 267–275, Los Angeles, USA.

Haugereid, P. (2004). Linking in constructions. In Müller, S., editor, *Proceedings of the HPSG-2004 Conference, Center for Computational Linguistics, Katholieke Universiteit Leuven*, pages 414–422. CSLI Publications, Stanford.

King, T. H., Forst, M., Kuhn, J., and Butt, M. (2005). The feature space in parallel grammar writing. *Research on Language and Computation*, 3:139–163.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119, Plainsboro, USA.

Maxwell, J. T. and Kaplan, R. M. (1996). An efficient parser for LFG. In Butt, M. and King, T. H., editors, *The Proceedings of the LFG '96 Conference*, Rank Xerox, Grenoble.

Oepen, S. and Flickinger, D. P. (1998). Towards systematic grammar profiling test suite technology ten years after. *Journal of Computer Speech and Language* Special Issue on Evaluation, 12:411–436.

Rohde, D. L. T. (2005). Tgrep2 user manual version 1.15. `http://tedlab.mit.edu/~dr/Tgrep2/tgrep2.pdf`.

Stenetorp, P., Pyysalo, S., Topić, G., Ohta, T., Ananiadou, S., and Tsujii, J. (2012). brat: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations Session at EACL 2012*, Avignon, France.

Coupling Trees, Words and Frames through XMG*

Timm Lichte[1], Alexander Diez[1], and Simon Petitjean[2]

[1] University of Düsseldorf, Germany
[2] University of Orléans, France

Abstract. This work presents first results on the integration of frame-based representations into the lexical framework of eXtensible MetaGrammar (XMG). Originally XMG supported tree-based syntactic structures and underspecified representations of predicate-logical formulae, but the representation of frames as a sort of typed feature structures failed due to reasons of usability and completeness. Therefore, after having explored strategies to simulate frames within the original XMG, we introduce an extension that is capable of handling frames directly by means of a novel `<frame>`-dimension. The `<frame>`-dimension can be also applied to recent accounts of morphological decomposition, as we show using a refined version of the `<morph>`-dimension from [4]. The presented extensions to XMG are fully operational in a new prototype.

1 Introduction

Recent work [9, 10, 16] has shown increasing interest in coupling a frame-based semantics with a tree-based syntax such as Tree Adjoining Grammar (TAG, [7]). While having lead to promising results on the theoretic side, it is unclear how to implement these ideas with existing grammar engineering tools, let alone how to bring them alive in natural language parsing. In this paper, we present first results on the integration of frame-based representations into the lexical framework of eXtensible MetaGrammar (XMG, [3]). XMG originally supported tree-based syntactic structures and underspecified representations of predicate-logical formulae, but the representation of frames as a sort of typed feature structures failed due to reasons of usability and completeness. Therefore we extend XMG by a novel `<frame>`-dimension that makes it capable of handling frames directly. This feature also paves the way for implementing recent accounts to morphological decomposition, such as in [16], where morphemes are linked to a frame-semantic representation.

The paper proceeds as follows. The next section briefly illustrates the lexical objects that we are concerned with, and Section 3 then shows the proposed

* This paper has greatly benefited from discussions with Laura Kallmeyer, Rainer Osswald and Yulia Zinova. We also thank the reviewers of HMGE 2013 for valuable comments. The work presented in this paper was financed by the Deutsche Forschungsgemeinschaft (DFG) within the CRC 991.

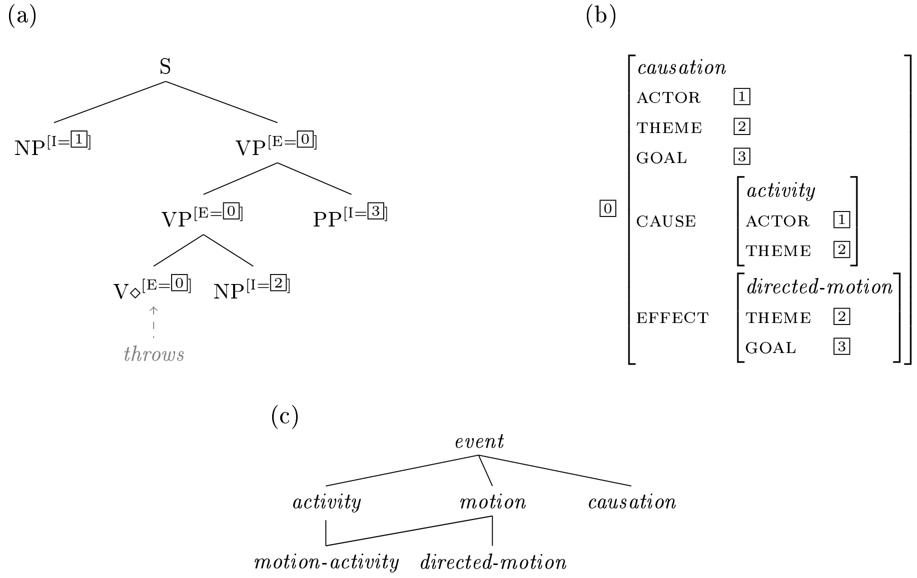


Fig. 1. A tree template, its frame-semantic counterpart and the associated type hierarchy.

factorization which crucially guides the implementation with XMG. After having explained the basics of XMG in Section 4, and after having explored strategies to simulate frames within the `<sem>`-dimension in Section 5, we present the `<frame>`-dimension in Section 6. Finally, Section 7 covers the application to a frame-based analysis of morphological decomposition.

2 A frame-based semantics for LTAG

We are concerned with lexical objects such as in Fig. 1, namely a (partial) phrase structure tree, a typed feature structure, and the associated type hierarchy, all of which are taken from [10, 11].

Phrase structure trees such as (a) make up the lexicon of a TAG, which is why they are also called *elementary trees*. TAG is a grammar formalism based on tree-rewriting, meaning that elementary trees can be combined (by means of two basic operations, substitution and adjunction) to generate larger trees.[3] A lexicalized TAG (LTAG) adds the constraint that every elementary tree includes at least one nonterminal leaf, the *lexical anchor*. Note however that throughout this paper we are rather concerned with so-called tree templates, which lack a

[3] Since in the present paper we are mainly concerned with the organization of the lexicon and thus focus on single elementary trees, we skip most details of the formalism here. See [7] or [1] for comprehensive presentations.

lexical anchor and from which elementary trees are lexically derived. The site of lexical insertion, here of *throws*, is marked by means of the ◇-symbol. Finally the nodes of both elementary trees and tree templates carry (non-recursive) feature structures, as shown at least in some nodes of (b) in Fig. 1. This is relevant for the interface between tree and frame, since following [9, 10, 16] the interface is indicated by means of co-occurring boxed numbers. For example, this way the subject NP-leaf is linked with the ACTOR role(s) of the frame, eventually causing the unification of the ACTOR role and the frame of the incoming NP.

Typed feature structures such as (b), on the other side, are a common representation of so-called frames (see [14]), which, according to Frame Theory [5], are considered a proper representation of mental concepts. As can be seen from the example in Fig. 1, features describe semantic participants and components, while feature structures correspond to conceptual objects, restricted by the type (*causation*, *activity*, . . .) that they are associated with. Moreover types are part of a type hierarchy, which determines the set of appropriate semantic features and moreover the unifiability of feature structures. Finally, boxed numbers indicate reentrancies, i. e. structure identity, which may cause a features structure to correspond to a graph rather than to a tree. This also holds for the feature structure in Fig. 1.

3 Factorization of tree templates and frames

Richly structured lexical objects like those in Fig. 1 make necessary some kind of metagrammatical factorization, once a large coverage grammar gets compiled and maintained [15]. Metagrammatical factorization roughly means to define recurring subcomponents of lexical objects, which can then be combined in several ways, for example in a purely constraint-based fashion as is the case in XMG. In addition to the benefit in terms of grammar engineering, however, [9–11] claim that metagrammar factorization can be also used to reflect constructional analyses in the spirit of Construction Grammar [12, 6]. Under this perspective the lexical material as well as the used “constructions” contribute meaning.

Taking these two aspects into account, [11] propose to factorize the tree template and the frame in Fig. 1 along the lines of Fig. 2, where boxes stand for the resulting factors or classes (i. e. classes in the sense of XMG), consisting of a tree and a frame fragment.[4] It illustrates that the tree-frame couple in Fig. 1 results from instantiating the class POConstr, which combines the classes n0Vn1 and DirPrepObj-to. Note that Fig. 2 shows a part of the proposed factorization only, as for example the class n0Vn1 would result from combining three other classes (Subj, VSpine, DirObj). Combining two classes essentially means that all associated information is unified, from which a minimal model is computed (see next section). Finally, one constructional facet can be found in the class POConstr in that it only contributes a frame fragment, but no tree fragment.

[4] Furthermore double edges indicate identity constraints, while within trees dashed edges represent non-strict dominance and $\prec^*$ non-strict precedence.

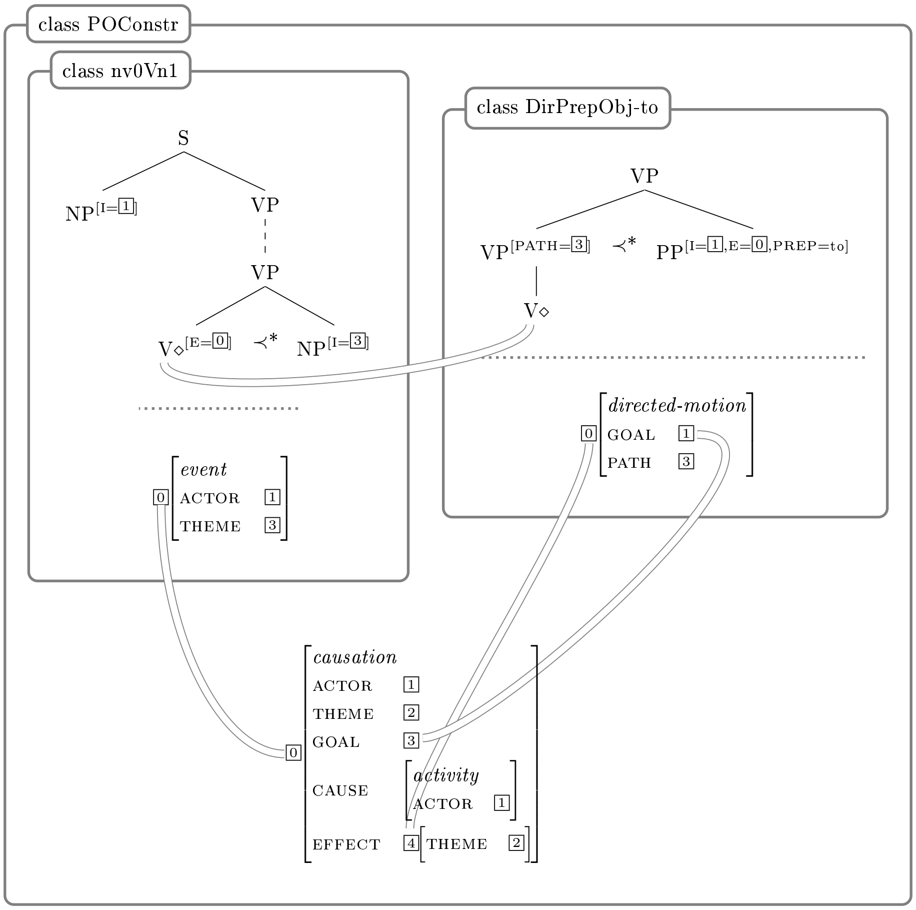


Fig. 2. Metagrammatical factorization of the tree template and the frame in Fig. 1.

The graphic representation of the metagrammatical factorization in Fig. 2 remains at a rather informal level and the question arises, how this could be translated into XMG code. We will see in Section 5 that the original XMG did not give a straightforward answer due to reasons of usability and completeness – other than the new `<frame>`-dimension, which is presented in Section 6.

4 A brief introduction to XMG

XMG (eXtensible MetaGrammar, [3]) stands both for a metagrammatical description language and the compiler for this language. Descriptions are organized into classes, that can be reused (i. e. "imported" or instantiated) by other classes. Borrowing from object oriented programming, classes are encapsulated, which

```
class n0Vn1
...
<syn>{
  node ?S [cat=s];  node ?VP1 [cat=vp]; node ?VP2 [cat=vp];
  node ?SUBJ [cat=np];  node ?OBJ [cat=np]; node ?V [cat=v];
  ?S->?SUBJ; ?S->?VP1; ?VP1->*?VP2; ?VP2->?OBJ; ?VP2->?V;
  ?V>>*?OBJ
}
...
```

Fig. 3. The <syn>-dimension of class n0Vn1.

means that each class can handle the scopes of their variables explicitly, by declaring variables and choosing which ones to make accessible for other instantiating classes (i. e. to "export"). The namespace of a class is then composed of the declared variables and all the variables exported by the imported classes.

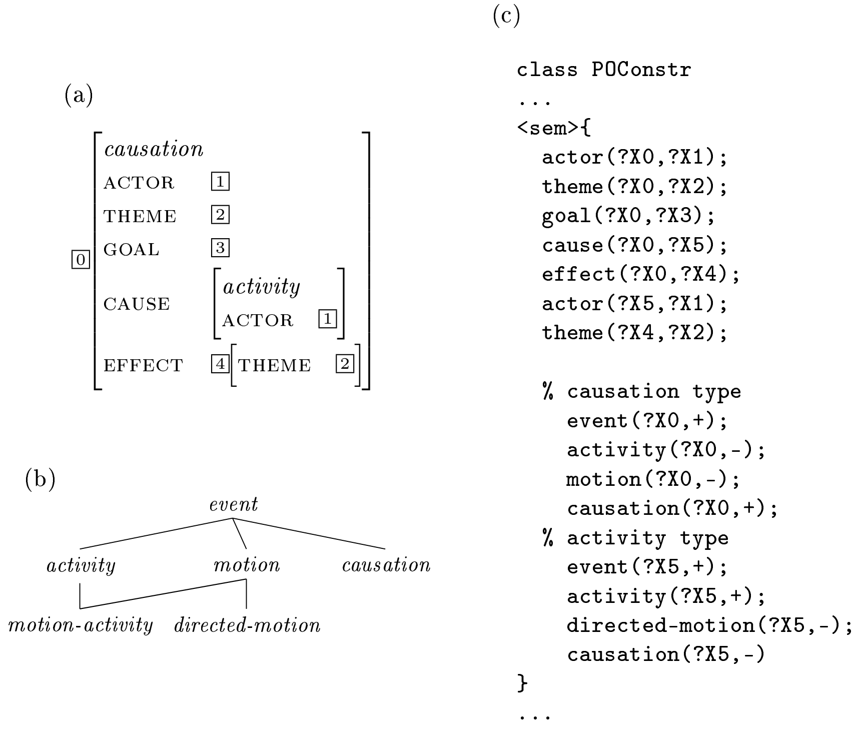
Crucial elements of a class are the so-called dimensions. Dimensions can be equipped with specific description languages and are compiled independently, therefore enabling the grammar writer to treat the levels of linguistic information separately. The <syn>-dimension, for example, allows to describe TAG tree templates (or fragments thereof). To give a concrete example, Fig. 3 shows the <syn>-dimension of class n0Vn1 from Fig. 2. It shows two sorts of statements, namely those like '`node ?S [cat=s];`' that instantiate nodes of the trees, and those like '`?S->?SUBJ;`' which determine the relative position of two nodes in the trees by referring to dominance and linear precedence.[5] In contrast, the <sem>-dimension includes descriptions of a different language as we will see in the following section.

When the metagrammar is compiled, first a set of descriptions for each class under evaluation is accumulated, and then the accumulated descriptions are resolved to yield minimal models. In the case of <syn>, the solver computes tree templates as minimal models, which is to say that only those nodes are included that are mentioned in the description. The final result can be explored with a viewer, or exported as XML file to feed a parser (e. g. the TuLiPA parser [8]).

5 Implementation within the <sem>-dimension

As mentioned before, the <sem>-dimension is designed to contain underspecified, flat formulae of predicate logic (borrowing from [2]). It is possible, however, to simulate frames by using binary predicates, such that they represent single semantic features. For example, a frame such as $\boxed{0}$[ACTOR $\boxed{1}$] is translated into the binary predicate `actor(?X0,?X1)`. A more detailed example for the class POConstr is shown in Fig. 4. The implementation of the syntax-semantics interface is straightforward, since the same variables can be used in both dimensions

[5] In XMG, variable names are prefixed with a question mark ('?').



```
class POConstr
...
<sem>{
  actor(?X0,?X1);
  theme(?X0,?X2);
  goal(?X0,?X3);
  cause(?X0,?X5);
  effect(?X0,?X4);
  actor(?X5,?X1);
  theme(?X4,?X2);

  % causation type
    event(?X0,+);
    activity(?X0,-);
    motion(?X0,-);
    causation(?X0,+);
  % activity type
    event(?X5,+);
    activity(?X5,+);
    directed-motion(?X5,-);
    causation(?X5,-)
}
...
```

Fig. 4. The feature structure of POConstr (repeated from Fig. 2), the corresponding type hierarchy (repeated from Fig. 1), and its simulation inside the `<sem>`-dimension.

`<syn>` and `<sem>`. Taking the class n0Vn1 from Fig, 2 as an example, the variable `?X1` would also appear in the XMG description of the subject-NP leaf, and the variable `?X2` in the description of the object-NP leaf (later identified with `?X3` in the class POConstr). We skip a code example due to lack of space.

While the simulation of the frame via binary predicates is straightforward, it is far less obvious how to simulate types and the type hierarchy. However, as can be seen again from Fig. 4, types can be also simulated by sets of binary predicates with a boolean second argument, or *type simulating sets* (TSS) as we call them, in the following way:[6] given a type hierarchy T such as the one in Fig. 4, we say that t in T is simulated by the minimal set of predicates $P_t(?X)$ for some variable $?X$, if $P_t(?X)$ is assembled in the following way: for every t' in T, if t' reflexively and transitively dominates t in T, then $t'(?X,+)$ is in $P_t(?X)$; else if t and t' have no common subtype, then $t'(?X,-)$ is in $P_t(?X)$. To give an example, $P_{directed\text{-}motion}(?X)$ for the type *directed-motion* in the type hierarchy of Fig. 4 would be the set $\{$*activity*$(?X,-)$, *motion*$(?X,+)$, *causation*$(?X,-)$,

[6] Thanks to Laura Kallmeyer and Yulia Zinova for pointing this out.

motion-activity$(?X, -)$, *directed-motion*$(?X, +)\}$.[7] It is easily seen that the size of some $P_t(?X)$ crucially depends on the position of t in T, and on the size of T.

One basic problem of this approach is that so far XMG does not interpret the predicates of the `<sem>`-dimension, but merely accumulates them for later use. Hence XMG allows for the coexistence of predicates `activity(?X,-)` and `activity(?X,+)`, which should be ruled out when simulating types as constraints on unification. But even if XMG was enhanced to verify the functionality of predicates, at least three disadvantages would remain: (i) TSS have to be provided by the grammar writer, (ii) they have to be included in the XMG descriptions as a whole, and (iii) unifying sister types with a common subtype will yield a TSS that does not immediately reveal the type of the common subtype. The latter disadvantage might be more of an aesthetic kind, but the first and the second one clearly have an impact on usability. Modifying the type hierarchy in the context of a large grammar would make necessary a meta-Metagrammar, that would automatically recompute the TSS and adapt the parts of the XMG descriptions, where TSS were used. Therefore we present a novel `<frame>`-dimension in the next section, which is adjusted to the peculiarities of frame structures and frame composition.

6 A new `<frame>`-dimension

The description language employed in the `<frame>`-dimension of the extended XMG follows the one of the `<syn>`-dimension in many respects.[8] Basically, a `<frame>`-dimension contains descriptions of nodes and edges, where nodes can be assigned a variable (with `var`) and a type, and edges can carry a semantic label. The example in Fig. 5 illustrates this.[9] Note that the `var`-equations correspond to the boxed numbers found in the AVM notation of frames (see Fig. 1 and 2). But comparing `<frame>`-descriptions with `<syn>`-descriptions also reveals several crucial distinctions: neither do `<frame>`-descriptions employ non-strict dominance, nor do they refer to linear precedence, as the daughters of a mother are generally unordered. Furthermore the edges in the `<frame>`-dimension can carry a semantic label, other than those in the `<syn>`-dimension.

The most essential difference, however, is found in the solvers, since the solver of the `<frame>`-dimension takes into account the type hierarchy, which is specified globally within the header of the code example in Fig. 5. It also computes and inserts the highest common subtype. Apart from that the solving of the `<frame>`-descriptions is relatively cheap, since nodes and edges are completely specified, and therefore the solving only consists of finding the root and traverse the edges top-down. In contrast, the solvers of the `<syn>`-dimension (and of the `<sem>`-dimension) rest upon the unification of untyped partial descriptions,

[7] Of course $P_{directed\text{-}motion}(?X)$ could be further minimized, since *motion-activity*$(?X, -)$ already follows from *activity*$(?X, -)$.

[8] To be exact, we extended XMG2 (http://wikilligramme.loria.fr/doku.php?id=xmg2).

[9] A bracket notation is also available, similarly to the one in the `<syn>`-dimension.

```
class POConstr
...
hierarchy TYPE = {(event,activity),(event,motion),(event,causation),
          (activity,motion-activity),
          (motion,motion-activity),(motion,directed-motion)}
...
<frame> {
  node (type=causation,var=?X0) ?ROOT;
  node (var=?X1) ?ACTOR;
  node (var=?X2) ?THEME;
  node (var=?X3) ?GOAL;
  node (type=activity) ?CAUSE;
  node (var=?X4) ?EFFECT;
  node (var=?X1) ?CAUSEACTOR;
  node (var=?X2) ?EFFECTTHEME;

  edge (label=actor) ?ROOT ?ACTOR;
  edge (label=theme) ?ROOT ?THEME;
  edge (label=goal) ?ROOT ?GOAL;
  edge (label=cause) ?ROOT ?CAUSE;
  edge (label=effect) ?ROOT ?EFFECT;
  edge (label=actor) ?CAUSE ?CAUSEACTOR;
  edge (label=theme) ?EFFECT ?EFFECTTHEME
}
...
```

Fig. 5. The `<frame>`-dimension of class POConstr.

which means they only take into account the values when unifying features or predicates.

The solvers of `<frame>` and `<syn>` do not differ, however, with respect to one crucial aspect: both resolve only tree structures. This might come as a surprise given that frames correspond to directed graphs [14], where nodes can be immediately dominated by more than one other node. Of course, in restricting itself to tree structures, the `<frame>`-dimension can model structures like the latter one only in an implicit way, namely by identifying separate nodes based on identical values in their respective `var`-property. Going back to the example in Fig. 5, the identification of nodes `?ACTOR` and `?CAUSEACTOR` is expressed through the `var`-value `?X1`. This account obviously borrows from the use of boxed-number variables in common AVM notation.

It remains to say that single connected frames with a unique root are resolved based on the descriptions within the `<frame>`-dimension. We do not see that solutions with more than one root could become necessary on the level of the lexicon.[10]

[10] Otherwise the solver of the `<frame>`-dimension might be modified analogously to the shift from TAG to multi-component TAG [13].

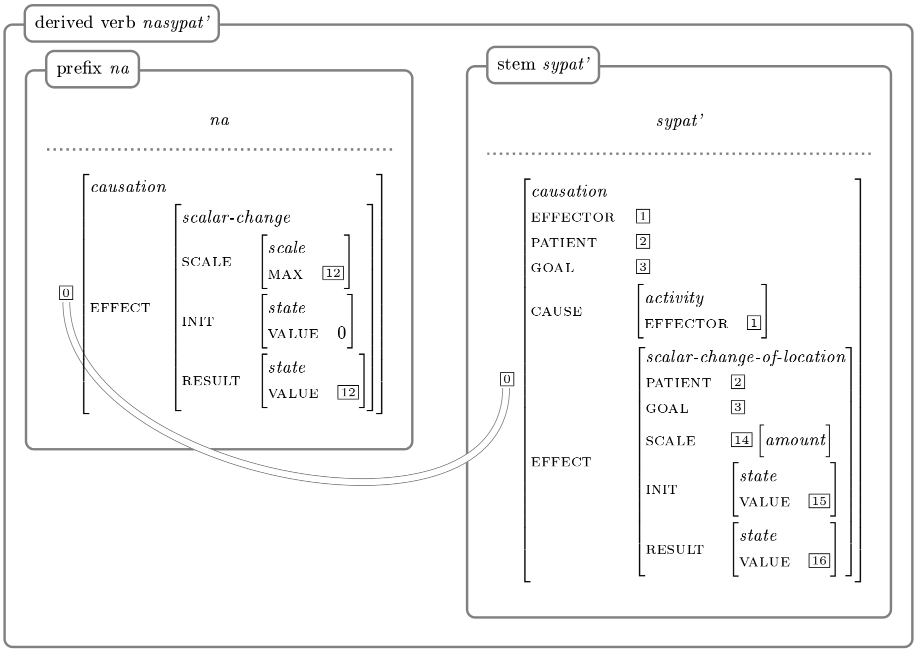


Fig. 6. Morphological decomposition of the Russian verb *nasypat'* ('to put') following [16]. The aspectual prefix *na* adds a perfective reading to the verbal stem *sypat'*. The perfective reading is encoded in the frame of *na* in terms of a specific scalar change.

7 An application to morphological decomposition

[16] not only present a metagrammatical factorization of elementary tree templates and their frame-based semantics, but they also briefly show that the same idea can be applied below the word level, namely to verbal prefixation in Russian: just as tree templates can be factorized (or decomposed) into tree fragments, complex morphological units can be decomposed into morphemes and their specific frame-semantic contribution. As an example, [16] decompose the perfective verb *nasypat'* ('to put') along the lines of Fig. 6. Again, we won't go into the linguistic details of the proposed analysis, but rather try to answer the question how this could be implemented by means of XMG.

Until recently, the morphological level of linguistic descriptions had not been attracting much attention within the framework of XMG. In general, morphology was (and still is) seen to lie outside its main focus, and that the word lexicon should be rather treated using other means. To our knowledge, the first work to deviate from this general picture is [4] in treating the system of agglutinative affixes in Ikota (a Bantu language). For this, [4] presents an extension of XMG which lets the grammar writer define linearly ordered "fields" for each type of affix. The specification and field assignment of affixes then takes place in a new dimension, called `<morph>`.

```
class nasypat
declare ?M1 ?M2 ?S1 ?S2
{
  ?M1 = na[];
  ?M2 = sypat[];
  ?S1 = ?M1.?S;
  ?S2 = ?M2.?S;
  <morph>{
        ?S1 >> ?S2
  }
}
```

```
class na                      class sypat
export ?S                     export ?S
declare ?S                    declare ?S
{ <morph>{                    { <morph>{
        morpheme ?S;                  morpheme ?S;
        ?S <- "na"                    ?S <- "sypat'"
  }                             }
}                             }
```

Fig. 7. The <morph>-dimension of the classes corresponding to the morphological decomposition of *nasypat'* in Fig. 6.

Considering the case of morphological decomposition shown in Fig. 6, it would be possible to adopt the fields-and-<morph> approach of [4] in a straightforward way: first two fields, say F1 and F2, would be globally specified, and then *na* would be assigned to F1 and *sypat'* to F2 within their respective morph-dimension. Given, however, that prefixation in Russian verbs is more flexible, allowing for, e. g., the stacking of several aspectual prefixes, we have chosen a more general approach which underlies the implementation shown in Fig. 7.[11] Instead of specifying a fixed number and order of fields, the linear order of *na* and *sypat'* is constrained locally inside the instantiating class nasypat using the common LP-operator ('>>'). Note that the operator <- assigns a surface string to a morpheme object. No matter what approach to morphology is chosen, the respective <frame>-dimension remains the same, along the lines of what has been presented in the last section. It is therefore omitted in Fig. 7.

The solver for the <morph>-dimension is rather simple compared to the one of <syn>, since the order of morphemes is constrained by immediate precedence only and the accumulated descriptions are supposed to be complete, meaning that no precedence between morphemes has to be inferred. After accumulating the morpheme objects and the precedence rules between them, the solver therefore just searches for the first morpheme (with no morpheme preceding it), and then

[11] A more flexible <morph>-dimension could be also advantageous in other cases, such as nominal compounds in German.

follows the line(s) of precedence rules. Finally it checks that no morpheme is left behind.

Of course the presented <morph>-dimension and its solver are very preliminary and designed specifically for the kind of analysis in Fig. 6. It needs to be clarified in subsequent research whether this is also applicable on a larger scale.

8 Conclusion

In this paper, we presented ongoing efforts to extend the grammar engineering framework XMG in order to deal with typed feature structures, respectively frames. We showed that the simulation of frames within the <sem>-dimension is doable, however there are disadvantages concerning the implementation of type hierarchies. Therefore a novel <frame>-dimension was developed which is adjusted to the peculiarities of frame structure and frame composition, and which should eventually reduce the burden for the grammar writer. We then showed that the <frame>-dimension can not only be combined with trees and tree fragments, but it can also be useful for the implementation of recent frame-based accounts to morphological decomposition, thereby considerably widening the scope of XMG.

The presented extensions to XMG are fully operational in a recent prototype. We further plan to make available a bracket notation for <frame>-descriptions that is closer to the common AVM notation, and to also include the <frame>-dimension in the XMG viewer.

References

1. Abeillé, A., Rambow, O.: Tree Adjoining Grammar: An overview. pp. 1–68 (2000)
2. Bos, J.: Predicate logic unplugged. In: Dekker, P., Stokhof, M. (eds.) Proceedings of the Tenth Amsterdam Colloquium. pp. 133–143. Amsterdam, Netherlands (1996)
3. Crabbé, B., Duchier, D., Gardent, C., Le Roux, J., Parmentier, Y.: XMG : eXtensible MetaGrammar. Computational Linguistics 39(3), 1–66 (2013), http://hal.archives-ouvertes.fr/hal-00768224/en/
4. Duchier, D., Ekoukou, B.M., Parmentier, Y., Petitjean, S., Schang, E.: Describing morphologically rich languages using metagrammars: a look at verbs in Ikota. In: Workshop on Language Technology for Normalisation of Less-Resourced Languages (SALTMIL 8 - AfLaT 2012). pp. 55–59 (2012), http://www.tshwanedje.com/publications/SaLTMiL8-AfLaT2012.pdfpage=67
5. Fillmore, C.J.: Frame semantics. In: The Linguistic Society of Korea (ed.) Linguistics in the Morning Calm, pp. 111–137. Hanshin Publishing (1982)
6. Goldberg, A.: Constructions at Work. The Nature of Generalizations in Language. Oxford Univ. Press, Oxford (2006)
7. Joshi, A.K., Schabes, Y.: Tree-Adjoining Grammars. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 69–124. Springer, Berlin, New York (1997)
8. Kallmeyer, L., Lichte, T., Maier, W., Parmentier, Y., Dellert, J., Evang, K.: Tulipa: Towards a multi-formalism parsing environment for grammar engineering. In: Coling 2008: Proceedings of the workshop on Grammar Engineering Across Frameworks. pp. 1–8. Manchester, England (August 2008)

9. Kallmeyer, L., Osswald, R.: An analysis of directed motion expressions with Lexicalized Tree Adjoining Grammars and frame semantics. In: Ong, L., de Queiroz, R. (eds.) Proceedings of WoLLIC. pp. 34–55. No. 7456 in Lecture Notes in Computer Science LNCS, Springer (September 2012)
10. Kallmeyer, L., Osswald, R.: A frame-based semantics of the dative alternation in Lexicalized Tree Adjoining Grammars. In: Piñón, C. (ed.) Empirical Issues in Syntax and Semantics 9. pp. 167–184 (2012), iSSN 1769-7158
11. Kallmeyer, L., Osswald, R.: Syntax-driven semantic frame composition in Lexicalized Tree Adjoining Grammar (2013), unpublished manuscript
12. Kay, P.: An informal sketch of a formal architecture for Construction Grammar. Grammars 5, 1–19 (2002)
13. Parmentier, Y., Kallmeyer, L., Lichte, T., Maier, W.: XMG: eXtending MetaGrammars to MCTAG. In: Actes de l'atelier sur les formalismes syntaxiques de haut niveau, Conférence sur le Traitement Automatique des Langues Naturelles, TALN 2007. Toulouse, France (June 2007)
14. Petersen, W.: Representation of concepts as frames. The Baltic International Yearbook of Cognition, Logic and Communication 2, 151–170 (2007)
15. Xia, F., Palmer, M., Vijay-Shanker, K.: Developing tree-adjoining grammars with lexical descriptions. In: Bangalore, S., Joshi, A.K. (eds.) Using Complex Lexical Descriptions in Natural Language Processing, pp. 73–110. MIT Press, Cambridge (2010)
16. Zinova, Y., Kallmeyer, L.: A frame-based semantics of locative alternation in LTAG. In: Proceedings of the 11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+11). pp. 28–36. Paris, France (September 2012), http://www.aclweb.org/anthology-new/W/W12/W12-4604

Extended Projections in a Guadeloupean TAG Grammar

Emmanuel Schang

Univ. Orléans, LLL UMR 7270, F-45065 Orléans, France

emmanuel.schang@univ-orleans.fr

1 Introduction

This paper presents a TAG (meta-)grammar of Guadeloupean Creole (GC). Guadeloupean is a French-based creole language spoken on the island of Guadeloupe (French West Indies). While sharing most of its lexicon with French, GC differs from French in its grammar (see Bernabé (1983) for a comprehensive description, and Damoiseau (2012) for a comparative approach). In particular, GC has preverbal Tense-Aspect Markers while French has inflectional suffixes for Tense. I will show that these TMAs can be correctly described as extended projections of verbs. They are generated as inflected forms of a verbal lexeme (i.e. in morphology rather than in syntax).

First, I present succinctly the key concepts of a TAG grammar to the reader. Next, section 2 describes the Tense-Aspect markers (TMAs) in GC and section 3 presents the way they are incorporated in the GC metagrammar [1]. Section 4 briefly presents how the same approach can be applied to the nominal domain.

Before addressing the question of Tense and Aspect markers, let me briefly present the key notions of TAG.

Tree-Adjoining Grammar (TAG) allows two operations (for a complete description, see Joshi and Schabes (1997), from which I take the following definitions):

- " *Substitution* takes only place on non-terminal nodes of the frontier of a tree. [...] By convention, the nodes on which substitution is allowed are marked by a down arrow (↓). When substitution occurs on a node *n*, the node is replaced by the tree to be substituted. When a node is marked for substitution, only trees derived from initial trees can be substituted for it." (Joshi and Schabes, 1997, p.4)
- " Adjoining[2] builds a new tree from an auxiliary tree β and a tree α (α is any tree, initial, auxiliary or derived). Let α be a tree containing a non-substitution node *n* labeled by X and let β be an auxiliary tree whose root node is also labeled by X. The resulting tree γ, obtained by adjoining β to α at node *n* is built as follow:

[1] I would like to thank Simon Petitjean, Yannick Parmentier and Denys Duchier for their precious help. I would also like to express my gratitude to the anonymous reviewers for their precious comments.

[2] I will prefer the term *adjunction* in the remainder of the paper.

- the sub-tree of α dominated by n, call it t, is excised, leaving a copy of n behind.
- the auxiliary tree β is attached at the copy of n and its root node is identified with the copy of n.
- the sub-tree t is attached to the foot node of β and the root node of t (i.e. n) is identified with the foot node of β. "

(Joshi and Schabes, 1997, p.4)

As proposed in Frank (2002), Substitution and Adjunction are supposed to be universal operations. The differences between languages can only reside in the shape of the elementary trees. The way elementary trees are built is thus a crucial matter, and yet the locus of divergences between several TAG grammars. It is thus necessary to explain the principles that govern the building of the elementary trees.

The major reference for a TAG French grammar is Abeillé (2002)[3] which presents the following (linguistic) principles of elementary trees well-formedness:

Lexical Anchoring: An elementary tree must have (at least) one non-empty lexical head.

Predicate-Argument Co-occurrence: A predicate elementary tree must have a node for each of its arguments.

Semantic Anchoring: A syntactic elementary tree must correspond to a (non-empty) semantic element.

Compositionality Principle: An elementary tree corresponds to one and only one semantic unit.

Furthermore, I adopt the Conditions on Elementary Tree Minimality (CETM) (Frank, 2002, 54) :

CETM: The syntactic heads in an elementary tree and their projections must form an extended projection of a single lexical head.

This leads me to adapt Grimshaw (2000)'s definition of head and projection to the TAG framework and say that an elementary tree is a maximal projection of one (or several) lexical head(s) within which the categorial features are shared. This opens discussions about what is functional and what is lexical. In the case of prepositions, this is a difficult matter for which the debate remains open (see Cinque (2010)).

2 Tense-Aspect Markers

2.1 A Brief Description of Tense-Aspect Markers in GC

Creole languages are known to make use of independent[4] markers to express Tense and Aspect (see Winford (2012) for a synthesis) usually gathered under the label *TMA markers* (Tense, Mood and Aspect markers) or TAM (Tense and Aspect Markers). Guadeloupean behaves like other creoles w.r.t. TMAs. The inflectional morphology found in

[3] For English, see the XTAG project (XTAG Research Group (2001))

[4] In a sense that will be clarified later.

French verbs is replaced in GC by independent morphemes. For comprehensive analyses, the reader may refer to McCrindle (1999) and Pfänder (2000). In Table 1, I will use the description of Vaillant (2008a), taking as a paradigmatic example the verb *dansé* 'to dance'[5] [6].

Table 1. Tense and Aspect markers.

Value	Form
Accomplished/Aoristic	dansé
Unaccomplished / Present	ka dansé
Frequentative	ka dansé
Progressive	ka dansé
Future	ké dansé
Unaccomplished Future (seldom)	ké ka dansé
Accomplished past (pluperfect)	té dansé
Unaccomplished past	té ka dansé
Irrealis (Past)	té ké dansé
Irrealis unaccomplished (extremely rare)	té ké ka dansé
Conditional / Optative	té dansé

Vaillant (2008b) also notes that *ké ka* and *té ké ka* are attested, although rare[7]. The precise semantic value and uses of these TMAs are beyond the scope of this paper and the reader may refer to Pfänder (2000) for a complete description. The following lines are just intended to present the useful key points for a non-specialist reader.

Bare Verbs: As in many languages, bare verbs in GC are used to express the (past) perfective (or preterite) with dynamic processes (as in (1-a)) and express the present tense with stative verbs[8], as in (1-b).

(1) a. Jan rivé.
Jean come
'Jean came.'
b. Jan enmé Sofi.
Jean love Sophie
'Jean loves Sophie.' (and not: * Jean loved Sophie)

Tense: The anterior marker of GC is *té*. When combined with non-stative verbs, *té* (ANT) provides a perfective interpretation:

[5] There is another marker *kay*, which is sometimes analysed as part of this system, however, as I will show later, it can't be included among the TMAs.

[6] This table shows the main uses of the TMA markers to give the reader a quick overview of the TMAs, but, as it will be claimed below, the interpretation of a TMA sequence is highly dependent on the context.

[7] I leave aside the marker *laka* which can plausibly be analysed as *la* (locative) and *ka*.

[8] And more generally with non-stative predicates, such as adjectival predicates.

(2) Sofi té palé ba Jan
Sophie ANT speak to Jean
'Sophie had spoken to Jean.'

And a past imperfective reading with stative verbs:

(3) Jan té enmé Sofi.
Jean ANT love Sophie
Litt.:'(At this time,) Jean was loving Sophie'

Aspect: GC Aspect markers are *ka* and *ké* for Imperfective and Prospective respectively. When *ka* is combined with stative verbs, the reading must be Iterative, as in:

(4) I ka tini onlo lajan a fen a chak mwa.
3sg ASP have a-lot-of money at end of each month
'He has a lot of money at the end of each month.' (example from (Delumeau, 2006, 117))

With non-stative predicates, the favored interpretation is Imperfective:

(5) Jan ka manjé.
Jean ASP eat
'Jean is eating'

ké triggers a future reading:

(6) Jan ké manjé.
Jean ASP eat
'Jean will eat'

2.2 Projecting TMAs

TMAs in GC have already been described in the TAG framework by Vaillant (2008a). This analysis differs from the one I present here on several aspects: First, it does not rely on a metagrammar, a point which matters since the implementation I present here relies heavily on the concept of metagrammar. Second, it calls upon adjunction to describe the TMA markers, an approach I will question here.

In this section, I will provide arguments in support of integrating TMA markers into the elementary trees as extended projection of a verbal head[9], instead of using adjunction as a way to integrate TMAs (as auxiliary trees) into the structure. Naturally, it is well known that every grammar using substitution can be rewritten using adjunction only. Formal considerations are therefore of little help in this discussion as far as the derived tree is concerned, but if derivation trees as intended to reflect the meaning of the sentence, the way the trees combine is a crucial matter. I base my argumentation on linguistic arguments and propose below a series of tests which show that TMAs behave differently from verbs.

[9] This can be extended to predicates in general, since there are non-verbal predicates in GC.

Several tests have been proposed for Romance languages (in Abeillé and Godard (2003) that cannot be used for GC. These are based on clitic-climbing and infinitival constructions which do not exist as such in GC and are therefore of little help. Instead, I will base my arguments on coordination and cleft structures.

Coordination: While coordination can apply to lexical items almost without restriction (see (Bernabé, 1983, 1396ff.) for a review of coordination in GC), TMA markers cannot be coordinated on the same position:

(7) *Jan ka é ké manjé.
Jean Imperf and Prosp eat
'Jean is and will be eating'

But lexical items can be coordinated:

(8) sèvolan la ka monté, monté, (é) monté !
kite def imperf go-up go-up (and) go-up
'The kite goes up, up, up!' (from (Bernabé, 1983, 545))

(9) shows that the repetition of the predicate is the norm, since coordination of TMA is blocked, and (10) shows that verb coordination can appear below TMAs.

(9) an fouté y, an ka fouté y, an ké fouté y !
1sg win 3sg 1sg imperf win 3sg 1sg prosp win 3sg
'I won (against him in the past), I won (this time) and I will win!'

(10) Jan ka dansé é chanté
Jean imperf dance and sing
'Jean is dancing and singing'

Predicate Cleft: Predicate cleft is a frequent construction in GC (while impossible in French). In (11), the lexical verb is clefted without the TMA marker. The same sentence with clefted TMA is ungrammatical, as in (12).

(11) sé monté nou ka monté pou nou rivé la nou ka alé.
it-is climb 1pl imperf climb for 1pl arrive there 1pl imperf go
'We are climbing to arrive where we are going' (intensive meaning)

(12) a. *sé **ka** monté nou ka monté... [Aspectual marker prohibited]
b. *sé **té** monté nou té monté...[Tense marker prohibited]

This test highlights the difference between (still) real periphrastic elements and TMAs. For instance, it is usually accepted that the form *kay (ka+ay)* 'Imperfective+go' (example (13)) is one of the TMA (see Vaillant (2008a) and (Bernabé, 1983, 1035)).

(13) Jan kay vini
Jean imperf+go come
'Jean is going to come'

But *ay* 'go' differs from the TMAs w.r.t this test, as shown in (14):

(14) sé ay Jan kay vini
it-is go Jean IMPERF+go come
'Jean is GOING to come.' (stress on the movement)

Then, the verb *ay* 'to go' will be inserted using adjunction, as in Abeillé (2002) for aspectual verbs and auxiliaries in French. It anchors its own tree and is stored in the Lexicon, contrarily to the TMAs.

The negative marker is also excluded from the clefted position (15), lending weight to the hypothesis that the clefted element is below any verbal functional projections [$_{VP}$ V (NP)].

(15) Sé (*pa) monté, nou pa monté
it-is NEG climb 1pl IMPERF climb
'We DIDN'T climb up.'

A noteworthy exception seems to be (16):

(16) Sé vlé pa, i vlé pa !
It-is will NEG 3SG will NEG
'He really doesn't want !'

But the behavior of *vlé*, here, is atypical, since the negative marker follows the verb[10].

From these tests, I conclude that TMA markers are functional elements, just like inflectional affixes in French are. The only – but significant– difference is that these markers are not expressed via synthesis but via a particular form of periphrasis (a situation where two or more words express the meaning expected for a single word). Syntactic intermediate projections are available between Tense and Aspect and allow some (rare) adverbs to be inserted between the TMA markers, as in (17):

(17) Pyè té ja ka vin.
Pierre ANTERIOR already IMPERF come
'Pierre was already coming.' (from (Bernabé, 1983, 1060))

Thus, I treat TMA markers as functional elements that form an extended projection of the lexical head, in conformity with the CETM.

This work is in accordance with many recent works attempting to reconsider the traditional division between morphology and syntax. For instance, Brown et al. (2012) try to avoid the binary division of labor between morphology and syntax and claim that periphrasis are both. I will follow here some key assumptions presented in the Paradigm Function Morphology (PFM) framework (Stump (2001); Bonami and Stump (prep)) which are highly compatible with a lexicalist approach of syntax. This separates clearly the TAG grammar I propose here from the standard GB analyses (in the wake of Chomsky (1981)) which are non-lexicalist. In particular, I adopt the idea that "some syntactic constructions express the pairing of a lexeme with a morphosyntactic set, and should thus be incorporated into inflectional paradigms as inflectional periphrases" (Bonami and Stump, prep, p.20).

[10] It might illustrate a case of lexical integration of *pa* into the verbal stem.

In the cases of the GC's TMAs, it is reasonable to claim that the TMAs and the lexical verb realize different forms of the same lexeme. Bonami and Webelhuth (2013) note that most of the morphological descriptions of functional periphrasis are rudimentary in their description of the periphrasis syntactic structures and that their syntactic behavior differs from one language to another (see also Abeillé and Godard (2003). I suggest that TMAs are in part similar to proclitics, an analysis which has also been proposed in the HPSG framework by Henri and Kihm (2013), but still allow syntactic nodes for adjunction. The present study – as well as Henri and Kihm (2013) and Schang et al. (2012) for other creole languages – suggest a finer-grained characterization of the TMAs from a typological point of view.

3 TMAs in the TAG Metagrammar

3.1 eXtensible Meta-Grammar

XMG[11] is a declarative language for specifying tree-based grammars at a meta-level (Crabbé and Duchier (2004); Crabbé et al. (2013)). XMG allows the linguist to capture generalizations on his grammar by defining tree fragments (*Classes*) that can combine via feature unification (conjunctive / disjunctive combinations of fragments). Once defined, Classes can be reused in distinct contexts, allowing elegant generalizations. A *core* grammar is described by fragments of elementary trees and these fragments combine to form the *expanded* grammar which is made of elementary trees[12]. I will use these concepts to describe the TMA part of the *core* grammar of GC (Crabbé and Duchier, 2004, p.1).

3.2 Fragments of Functional Projections

From the preceding section, I take for granted that TMAs are not stored in the Lexicon (they don't anchor any tree properly) but are co-anchors of the elementary tree associated with verbs[13].

For instance, (19) illustrates the structure of (18).

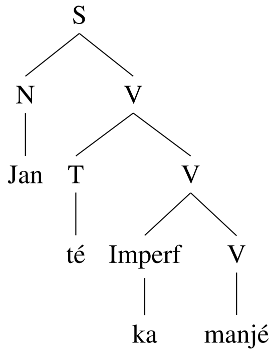
(18) Jan té ka manjé
Jean ANT IMPERF *eat*
'Jean was eating.'

[11] I am using XMG-2 version.

[12] See Crabbé (2005) for a large metagrammar of French using XMG.

[13] For a similar approach on Santomense's Tense and Aspect markers, see Schang et al. (2012) .

(19)

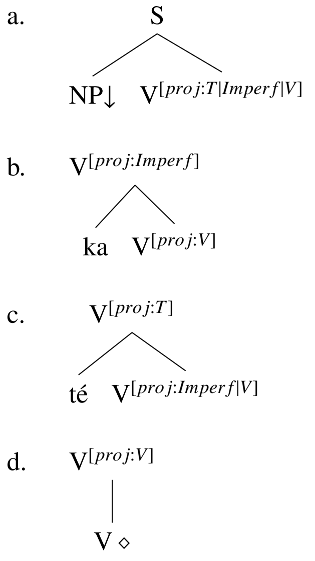


In (19), S is a projection of V, the maximal functional stretching of the verb.

In XMG's framework, the structure (19) is broken down into four pieces (*i.e.* classes) each containing minimal information. These Classes are listed below.

- *CanSubject*: for the External Argument of the verb. It is described in (20-a) .
- *Imperf*: as a projection of the Imperfective (Aspect) marker. It is described in (20-b).
- *Tensed*: as a projection of Tense.
- *Intransitive* verb: the minimal projection of V. It is described in (20-d).

(20)



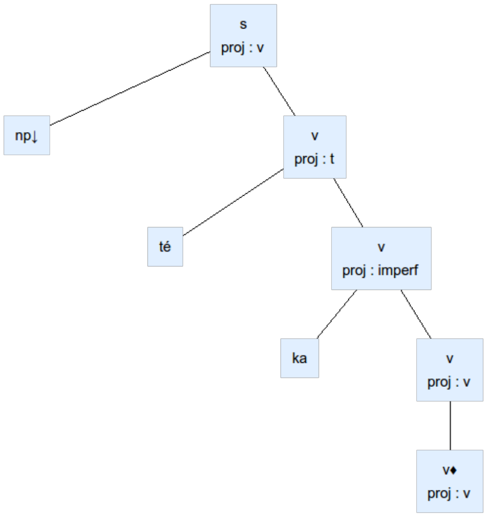
Thus, (19) is built up from the following conjunction of Classes:

$$CanSubject \wedge Intransitive \wedge Imperf \wedge Tensed$$

As in Schang et al. (2012) for Santomense, the feature *proj(ection)* is used to rule out invalid combinations in the output elementary tree.[14]

[14] In the values associated with feature proj, "|" refers to disjunction.

From the conjunction of classes given above, the result of the metagrammar compilation are elementary trees for intransitive verbs, inflected with Tense and Aspect, as shown in (21)[15].



(21)

The same mechanism is used to describe verb families. For instance, the class of Intransitive verbs inflected for TMAs can be described as a class *IntransV* gathering the classes of the inflected forms for intransitive verbs[16]:

$$BareV \mid ImperfV \mid ProspImperfV \mid TensProspImperfV$$

$$\mid TensedV \mid ProspV \mid TensImperfV \mid TensProspV$$

As expected from the morphological approach I defend here, the TMAs do not appear as adjuncts but are co-anchors of the verb. (22) shows the derivation tree[17] for the sentence *Jan té ka manjé* 'Jean was eating'.

(22) α1-manjé$[Ant; Imperf]$

$$\begin{tikzcd}\alpha\text{1-manjé}[Ant; Imperf] \arrow[d, no head] \\ \alpha\text{2-Jan}\end{tikzcd}$$

The benefit we have here with regards to adjunction is that the semantic interpretation of the sequence *N0 té ka V* is directly derivable from the features on the verb (see Table

[15] Which is a picture of the elementary tree as displayed by XMG.

[16] The abbreviations are: Imperf(ective), Prosp(ective), Tens(ed).

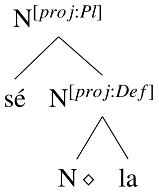
[17] Where α is the conventional label for an elementary tree.

1). There is no need to postulate ambiguous TMA markers triggering various interpretations or different syntactic positions for the same marker, as is commonly proposed in creole languages descriptions (see Winford (2012)), i.e. postulating several distinct *ka* or Ø markers to derive the correct interpretation. On the contrary, it is compatible with the PFM idea that words (or multiword expressions) realize the morphosyntactic property set of the lexeme they are member of. Then, *té ka manjé* is the realization of the morphosyntactic set $\{Anterior, imperfective\}$ for the lexeme MANJÉ.

4 Extended Projections in the Nominal Domain

The same methodology developed for the TMAs is adopted in the Nominal domain, incorporating Definite, Demonstrative and Plural markers as co-anchors of a lexical Noun, as in (23), where the Plural marker *sé* precedes the N and the Definite marker follows the N, as illustrated in (24). Note that in GC, the Plural requires the Definite, **sé timoun* being agrammatical.

(23)



(24) [sé timoun la] vin
PL child DEF come
'The children came.'

(23) is the conjunction of the classes:

$$Plural \mid Definite \mid Noun$$

where *Plural* can only appear if *Definite* is present.

The genitive preposition a[18] 'of' in (25), which only has a functional role akin to a Genitive case marker is also treated as a co-anchor.

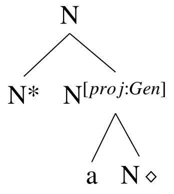
In (25), the NP [a Lelette] is adjoined to the head noun *kaz*.

(25) [kaz [a Lelette]]
house GENITIVE Lelette
'Lelette's house'

The genitival form of the noun (as head of the genitive NP) is an auxiliary tree (for adjunction) containing the co-achor *a*, as shown in (26).

[18] And its allomorph *an* when preceding a nasal.

(26)



The comparison of GC with a Matiniké Creole (spoken on the island of Martinique, closely related to GC and diachronically linked to GC) lends weight to this approach, since in Matiniké, the same NP as in (25) would be *kaz Lelette*, without preposition. The difference between these creoles relies only in the fact that Matiniké has a covert (silent) preposition for the same structure as (26).

To summarize, the inflected forms of a noun can be represented as a conjunction of classes:

$$Demonstrative - N \mid Plural - N \mid Definite - N \mid Bare - N \mid Genitive - N$$

Note that while Plural, Demonstrative, Definite and Genitive appear as syntactic projections in GC, the same functional elements are concatenated at the word level in other languages.

5 Conclusion

I have shown that Tense and Aspect markers of Guadeloupean Creole are functional projections that can be described as co-anchors of a lexical head (extended projections). Their response to the tests of coordination and cleft structures shows that TMAs are not as free as other periphrastic elements (such as the verb *ay* 'to go'). Their place is somewhere between concatenative morphology at the word level and 'free' periphrastic elements, which I take to be adjoined elements. In the nominal domain, I have suggested that the definite, demonstrative, plural and genitive markers are also functional elements of the same sort. These functional elements are integrated in the elementary trees that form the GC grammar as co-anchors of the lexical item. I have demonstrated how these elements form fragments of (elementary) trees and how they combine to form the expanded grammar. To do this, I have used the XMG formalism for metagrammar. Since the combination of the fragments constitute inflected forms of a lexeme (as the compound tenses are still members of the verbal paradigm, see Ackerman and Stump (2004)), the building of Elementary trees is as much a morphological as a syntactic operation. It thus casts a new light on Creole languages which are commonly thought to have little (or even no) morphology.

Bibliography

Abeillé, A. (2002). *Une Grammaire électronique du Français*. CNRS Editions, Paris.

Abeillé, A. and Godard, D. (2003). Les prédicats complexes dans les langues romanes. *Les langues romanes. Paris: CNRS Editions*, pages 125–184.

Ackerman, F. and Stump, G. (2004). Paradigms and periphrastic expression: a study in realization-based lexicalism. *Projecting Morphology. CSLI.*, pages 111–58.

Bernabé, J. (1983). *Fondal-natal*. l'Harmattan Paris.

Bonami, O. and Stump, G. (in prep). Paradigm Function Morphology. In Spencer, A., editor, *The Handbook of Morphology, 2nd ed.* Wiley-Blackwell.

Bonami, O. and Webelhuth, G. (2013). The phrase-structural diversity of periphrasis: a lexicalist account. In Chumakina, M. C. G. G., editor, *Periphrasis: The role of syntax and morphology in paradigms*. Oxford University Press.

Brown, D., Chumakina, M., Corbett, G., Popova, G., and Spencer, A. (2012). Defining 'periphrasis': key notions. *Morphology*, 22(2):233–275.

Chomsky, N. (1981). Lectures on the Theory of Government and Binding. *Dordrecht: Foris*.

Cinque, G. (2010). The syntax of adjectives: a comparative study.

Crabbé, B. (2005). *Représentation informatique de grammaires d'arbres fortement lexicalisées : le cas de la grammaire d'arbres adjoints*. PhD thesis, Université Nancy 2.

Crabbé, B. and Duchier, D. (2004). Metagrammar Redux. In *International Workshop on Constraint Solving and Language Processing*, Copenhagen.

Crabbé, B., Duchier, D., Gardent, C., Le Roux, J., and Parmentier, Y. (2013). XMG : eXtensible MetaGrammar. *Computational Linguistics*, 39(3):1–66.

Damoiseau, R. (2012). *Syntaxe créole comparée*. Karthala et cndp-crdp edition.

Delumeau, F. (2006). *Une description linguistique du créole guadeloupéen dans la perspective de la génération automatique d'énoncés*. PhD thesis, Université Paris X.

Frank, R. (2002). *Phrase Structure Composition and Syntactic Dependencies*. MIT Press, Cambridge, Mass.

Grimshaw, J. (2000). Locality and extended projection. *Amsterdam Studies in the Theory and History of Linguistic Science Series 4*, pages 115–134.

Henri, F. and Kihm, A. (2013). The Morphology of TMA in Mauritian and Creole.

Joshi, A. K. and Schabes, Y. (1997). Tree-Adjoining Grammars. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York.

McCrindle, K. L. (1999). Temps, mode et aspect, les creoles des Caraibes a base lexicale française.

Pfänder, S. (2000). *Aspekt und Tempus im Frankokreol*. G. Narr.

Schang, E., Duchier, D., Ekoukou, B. M., Parmentier, Y., and Petitjean, S. (2012). Describing São Tomense Using a Tree-Adjoining Meta-Grammar. In *11th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 11)*.

Stump, G. (2001). *Inflectional morphology: a theory of paradigm structure*, volume 93. Cambridge Univ Pr.

Vaillant, P. (2008a). A layered grammar model: Using tree-adjoining grammars to build a common syntactic kernel for related dialects. *arXiv preprint arXiv:0810.1207*.

Vaillant, P. (2008b). Une grammaire formelle du créole martiniquais pour la génération automatique. *arXiv preprint arXiv:0810.1199*.

Winford, D. (2012). Creole Languages. *The Oxford Handbook of Tense and Aspect*.

XTAG Research Group (2001). A Lexicalized Tree Adjoining Grammar for English. Technical report, Institute for Research in Cognitive Science, Philadelphia. Available from `ftp://ftp.cis.upenn.edu/pub/xtag/release-2.24.2001/tech-report.pdf`.

Frigram: a French Interaction Grammar

Guy Perrier and Bruno Guillaume

LORIA, Université de Lorraine, Nancy, France

Abstract. We present Frigram, a French grammar with a large coverage, written in the formalism of Interaction Grammars. The originality of the formalism lies in its system of polarities, which expresses the resource sensitivity of natural languages and which is used to guide syntactic composition. We focus the presentation on the principles of the grammar, its modular architecture, the link with a lexicon independent of the formalism and the companion property, which helps to guarantee the consistency of the whole grammar.

Keywords: Formal Grammar, Model Theoretic Syntax, Polarity, Interaction Grammar.

1 Introduction

The aim of our work is to show that it is possible to build a realistic computational grammar of French, which integrates fine linguistic knowledge with a large coverage. As a framework, we have chosen the formalism of Interaction Grammar (IG) [7]. IG combines a flexible view of grammars as constraint systems with the use of a polarity system to control syntactic composition. The system of polarities expresses the saturation state of partial syntactic structures and their ability to combine together.

The main challenge is to guarantee and to maintain the consistency of the grammar while aiming at the largest coverage. We resort to several means:

- a modular organization of the grammar in a hierarchy of classes, which is able to capture the generalizations of the language,
- principles of well-formedness for the elementary structures of the grammar,
- a separation of the grammar itself from the lexicon, which is independent of any grammatical formalism,
- the use of the *companion property* to help the checking of the grammar consistency.

Starting with a brief presentation of IG, we continue with an explanation of the different points mentioned above and with a comparison with other French grammars and a discussion about the evaluation of the grammar.

2 Interaction Grammars

IG is a grammatical formalism which is devoted to the syntax of natural languages using two notions: *tree description* and *polarity*. For a complete presentation of the formalism, the reader can refer to [7].

2.1 Tree Descriptions

The notion of a tree description [11] is related to a model theoretic view of the syntax of natural languages [10]. In this view, the basic objects of the grammar are not trees but properties that are used to describe them, in other words *tree descriptions*. This approach is very flexible allowing the expression of elementary properties in a totally independent way, and their combination in a free manner. A tree description can be viewed either as an underspecified tree, or as the specification of a tree family, each tree being a model of the specification.

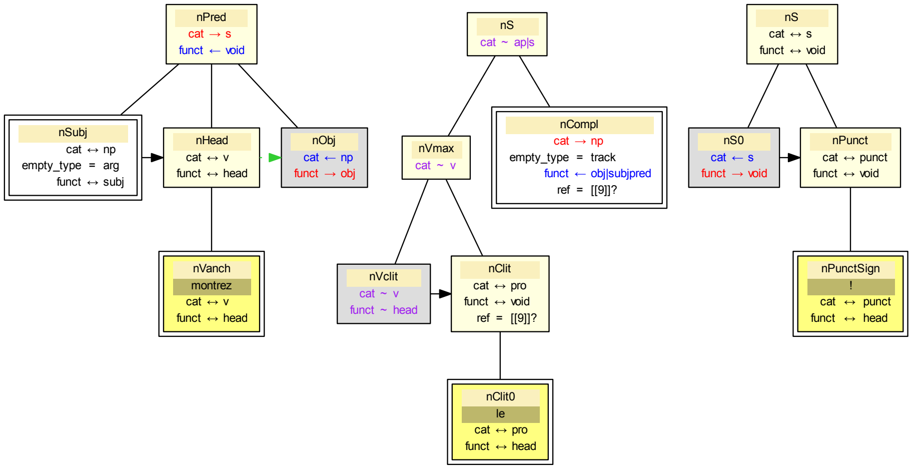


Fig. 1. PTD associated with the sentence *"montrez-le !"* by the grammar FRIGRAM.

Figure 1 gives an example of the tree description, which is associated with the sentence *"montrez-le !"* [*"show it !"*]. Even if the description is composed of three parts associated with the three words of the sentence (punctuation signs are considered as words), it must be considered as a unique tree description.

A tree description is a finite set of nodes structured by two kinds of relations: *dominance* and *precedence*. Dominance relations can be immediate or large. In the example, there are only immediate dominance relations represented with solid lines. Precedence relations can also be immediate or large. They are represented with arrows in Figure 1; these arrow are solid and black or dashed and green, depending on whether the dependencies are immediate or large.

Nodes, which represent constituents, are labelled with features describing their morpho-syntactic properties. Feature values are atoms or atom disjunctions. When a feature value is the disjunction of all elements of a domain, this value is denoted with "?". A mechanism of co-indexation between feature values (a common index [[n]] is put before their values) allows for sharing.

It is possible to add constraints on the phonological form and on the daughters of nodes: a node is declared to be *empty* if its phonological form is empty and it is graphically represented with a white rectangle; at the opposite, it is declared to be *full*, and it is represented with a light-yellow rectangle; it is declared to be *closed*, if the set of its daughters is fixed and it is represented with a double rectangle; finally, a node is declared to be an *anchor*, if it is a full leaf, and it is used to anchor a word of the language. An anchor is represented with a canary yellow rectangle.

IG uses three kinds of empty nodes:

- when an argument has moved from its canonical position, this position is marked with a *trace*, an empty node with the feature `empty_type = track`; this covers all cases of extraction, subject inversion and cliticization of arguments; in Figure 1, node *nCompl* is the empty trace of the object represented with the clitic pronoun *"le"*;
- when an argument is not expressed with a phonological form, it is represented with an empty node carrying the feature `empty_type = arg`; this is the case for subjects of adjectives, infinitives and imperatives, as well as some objects of infinitives (tough movement); in Figure 1, node *nSubj* represents the non-expressed subject of the imperative verb *"montrez"*;
- in presence of an ellipsis, the head of the elided expression may be represented with an empty node carrying the feature `empty_type = ellipsis`.

2.2 Polarities

Polarities are used to express the saturation state of syntactic trees. They are attached to features that label description nodes with the following meaning:

- a *positive* feature `f` → `v` expresses an available resource, which must be consumed;
- a *negative* feature `f` ← `v` expresses an expected resource, which must be provided; it is the dual of a positive feature; one negative feature must match exactly one corresponding positive feature to be saturated and conversely;
- a *saturated* feature `f` ↔ `v` expresses a linguistic property that needs no combination to be saturated;
- a *virtual* feature `f` ∼ `v` expresses a linguistic property that needs to be realized by combining with an actual feature (an actual feature is a positive or saturated feature).

In Figure 1, node *nObj* carries a negative feature `cat` ← `np` and a positive feature `funct` → `obj`, which represents the expected object noun phrase for the transitive verb *"montrez"*.

The virtual features of the second part of the tree description represent the syntactic context required by the clitic pronoun *"le"*: a verb *nVclit* put immediately before the pronoun to build the node *nVmax* with it.

Only resource sensitive features are polarized. Other features are called *neutral* features and denoted `f = v`. For instance, agreement properties are expressed with neutral features.

The descriptions labelled with polarized features are called *polarized tree descriptions (PTDs)* in the rest of the article.

2.3 Grammars as constraint systems

An interaction grammar is defined by a finite set of Elementary PTDs, named EPTDs in the following, and it generates a tree language. A tree belongs to the language if it is a model of a finite set of EPTDs in the sense given by [7]. Each node of the EPTDs is mapped to a node of the model through an interpretation function. Properties of models include:

- A model is *saturated*: every positive feature $\mathtt{f} \rightarrow \mathtt{v}$ is matched with its dual feature $\mathtt{f} \leftarrow \mathtt{v}$ in the model and vice versa. Moreover, every virtual feature has to find an actual corresponding feature in the model.
- A model is *minimal*: it has to add a minimum of information to the initial descriptions (it cannot add immediate dominance relations or features that do not exist in the initial descriptions).

Parsing a sentence with a grammar G consists first in selecting an appropriate set of EPTDs from G. The selection step is facilitated if G is lexicalized: each EPTD has an anchor associated with a word of the language. It strongly reduces the search space for the EPTDs. Then, the parsing process itself reduces to the resolution of a constraint system. It consists in building all models of the selected set of EPTDs.

Figure 1 represented a possible selection of EPTDs from FRIGRAM to parse the sentence *"montrez-le !"*. The selection includes three EPTDs[1], which are gathered in a unique PTD. Figure 2 shows the unique minimal and saturated model of the PTD. It is an ordered tree where nodes are labelled with non polarized features in the form `f : v`, where `v` is an atomic value. In the head of each node, a list gives the nodes of the PTD that are interpreted in the node of the model.

In an operational view of parsing, the building of a saturated and minimal model is performed step by step by refining the initial PTD with a merging operation between nodes, guided by one of the following constraints:

- neutralise a positive feature with a negative feature having the same name and carrying a value unifiable with the value of the first feature;
- realize a virtual feature by combining it with an actual feature (a positive or saturated feature) having the same name and carrying a value unifiable with the value of the first feature.

[1] The EPTDs are labelled by the name of the class of the grammar generating them followed by a number. In the order of the sentence, we have NP0_V_NP1_134, PROCLIT_COMPL_IMPERS_POS_38 and PUNCTSTOP_S1INTER_IMPER_1.

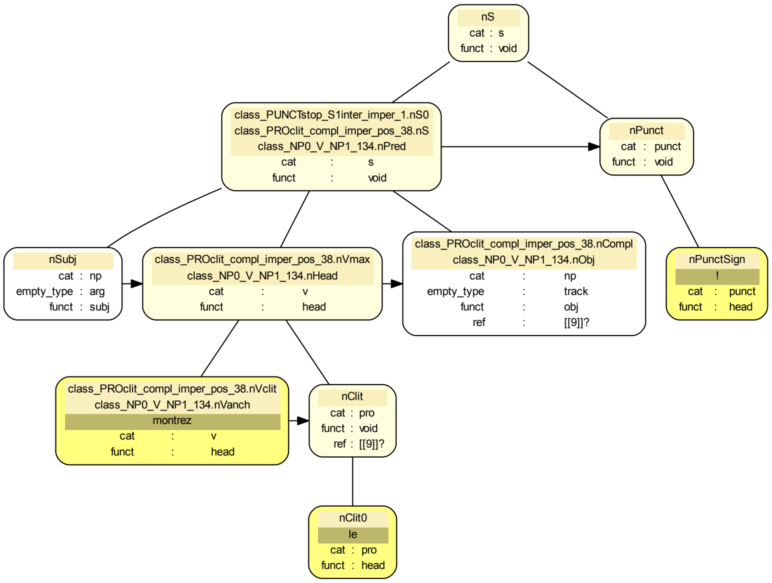


Fig. 2. Model of the PTD shown in Figure 1 representing the syntax of the sentence *“montrez-le !”*.

The constraints of the description interact with node merging to entail a partial superposition of their contexts represented by the tree fragments in which they are situated. So the model of Figure 2 can be obtained from the PTD of Figure 1 with a sequence of three node merging operations: *nVanch* with *nVclit*, *nObj* with *nCompl* and *nPred* with *nS0*.

To summarize, IG combine the strong points of two families of formalisms: the flexibility of *Unification Grammars* and the saturation control of *Categorial Grammars*.

3 The Principles of the Grammar Frigram

Frigram is a IG for the French language; it contains 3 794 EPTD templates. Frigram follows a set of principles which express formally the chosen linguistic modeling. These principles are also used to automatically check the consistency of the grammar and its conformity to linguistic principle. The constituant to dependency transformation used with Frigram also strongly relies on this set of principles.

Definition 1. *A node with a positive or saturated* `cat` *feature is called a* concrete node*.*

Principle 1 *(***cat-funct***) In an EPTD, any node has a* `cat` *feature and if it is concrete, it has also a* `funct` *feature.*

The consequence is that any node of a model has a `cat` feature and a `funct` feature. Another consequence is that any node of a model has a unique concrete antecedent in the original PTD, because two concrete nodes of a PTD cannot merge in the model, according to the composition rules of polarities.

Principle 2 *(***strict lexicalisation***) Any EPTD has exactly one anchor node. This anchor node has a saturated* `cat` *feature with an atomic feature value.*

Definition 2. *A* spine *in an EPTD is a list of nodes $N_1, N_2, \ldots, N_p$ such that:*

- *for any i such that $1 < i \leq p$, node N_i is a daughter node of N_{i-1};*
- *for any i such that $1 < i \leq p$, node N_i has a saturated feature* `cat` *and a feature* `funct` ↔ `head`*;*
- *node N_1 is a concrete node and its feature* `funct` *has a value different from* `head`*; it is called the* maximal projection *of all nodes belonging to the spine;*
- *node N_p is either an anchor or an empty leaf; in the first case, the spine is called a* main spine*; in the second case, it is called an* empty spine*; in both cases, node N_p is called the* lexical head *of all nodes belonging to the spine.*

Principle 3 *(***spine***) Any concrete node of an EPTD belongs to exactly one spine.*

An important corollary of the spine principle is that every node N of a PTD model has exactly one lexical head in this model, denoted $head(N)$ and defined as follows: the concrete antecedent of N in the initial PTD belongs to exactly one spine and $head(N)$ is the interpretation in the model of the leaf ending the spine.

A second important corollary is that every node in a PTD model which is not a leaf has exactly one daughter node with the feature `funct` : `head`. By following all nodes with this feature, we have a more direct way of finding the lexical head of every node in a PTD model.

A third corollary is that each model node with a positive feature `cat` is the maximal projection of some spine.

From the strict lexicalisation and spine principles, we can also deduce that every EPTD has exactly one main spine.

To illustrate the concept of a spine, let us consider the EPTDs of Figure 1. The EPTD associated with the verb *"montrez"* has two spines: the main spine *nPred*, *nHead*, *nVanch* with its lexical head *nVanch*, and an empty spine reduced to a single node *nSubj*. The formalism of IG is situated in the constituency approach to syntax, as opposed to the dependency approach but the principles of Frigram allow for an automatic transformation of any parse made with IG from a constituency setting into a dependency setting. Our purpose here is not to describe the transformation in detail but to give an outline of it.

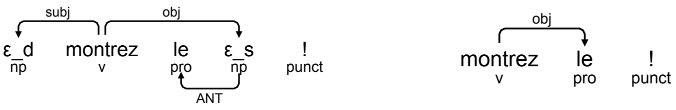


Fig. 3. The dependency graphs representing the syntax of the sentence *"montrez-le !"*.

The dependencies are generated by the interactions between the polarized features `cat` and `funct` of the different nodes of the initial PTD and they are projected on the full and empty words of the sentence through the notion of lexical head.

For the sentence *"montrez-le !"*, from the PTD of Figure 1 and its model from Figure 2, we compute the dependency graph on the left of Figure 3. It has two empty words, the subject of *"montrez"*, named ϵ_a (which corresponds to the node with feature `empty_type = arg`), and its object ϵ_t (which corresponds to the node with feature `empty_type = track`), which is the trace of the clitic pronoun *"le"*. Traces are linked to their antecedent with the relation ANT.

In a second step, the empty words are removed and their incident dependencies are transferred to their full antecedent, when it exists. In our very simple example, the resulting dependency graph reduces to the tree on the right of Figure 3, but in more complex sentences, the dependency graph includes cycles and nodes with several governors. When there is more than one solution, handcrafted rules are used to compute a weight for each solution in order to rank them.

4 The Architecture of the Grammar

4.1 The Modular Organisation of the Grammar

It is unthinkable to build a grammar with about 4000 EPTD templates manually, considering each one individually. Even if it were possible, to maintain the consistency of such a grammar would be intractable.

Now, the EPTD templates of Frigram share a lot of fragments and it is possible to organize the grammar as a class hierarchy. A tool, XMG [4], was specially designed to build such kind of grammars. XMG provides a language to define a grammar as a set of classes. A class can be defined directly but also from other classes by mean of two composition operations: *conjunction* and *disjunction*.

Each class is structured according to several dimensions. Frigram uses two dimensions: the first one is the syntactic dimension, where objects are EPTD templates, and the second one is the dimension of the interface with the lexicon, where objects are feature structures.

The terminal classes of the hierarchy define the EPTD templates of the grammar that are computed by the XMG compiler. Figure 4 gives the example of a

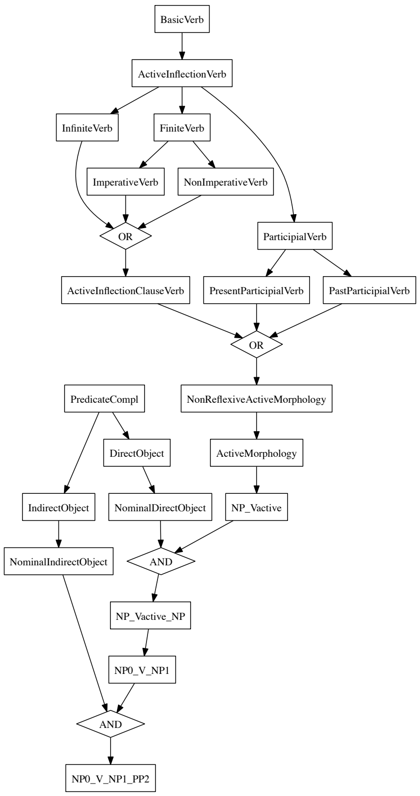


Fig. 4. The hierarchy of classes used to define the NP0_V_NP1_PP2 class of transitive verbs with an indirect complement

terminal class, the NP0_V_NP1_PP2 class of transitive verbs with an indirect complement, with the hierarchy of classes used to define it.

The current grammar Frigram is composed of 428 classes, including 179 terminal ones, which are compiled into 3 794 EPTD templates. Of course, some general classes can be used in several different contexts. For instance, adjectives, nouns and verbs description all inherit from the same subclasses related to complements of predicative structures. The set of classes is organized in a module hierarchy 5.

There is another hierarchy related to the different forms of extraction, in relative, interrogative and cleft clauses. The root module of the hierarchy is the ExtractGramWord module. Three modules depend on it: Complementizer, Interrogative and Relative. Moreover, there are isolated modules related to specific categories.

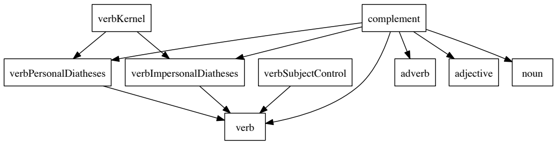


Fig. 5. The main hierarchy of modules

4.2 The link with a lexicon independent of the formalism

The full grammar is produced from the set of EPTD templates and a lexicon. Each EPTD template is associated to a feature structure (called its interface) which describes a syntactic frame corresponding to lexical units able to anchor it; lexicon entries are also described through features structures. Unification between interface of the EPTD template and lexicon feature structure is used to control the combination of a lexical unit description and the template. Thanks to the strict lexicalisation principle, each EPTD of the grammar has a unique anchor node linked with a lexical unit of the language. Since there is a co-indexation between features of the EPTD template and features of the interface, a side effect of anchoring is the instantiation of some feature values in the EPTD.

In our system, the lexicon used is FRILEX[2] which combines morphological information taken in ABU[3] and in Morphalou [12] with syntactical information for verbs from Dicovalence [14]. FRILEX contains 530 000 entries. To avoid size explosion, the full grammar is built on the fly on each input sentence.

5 The Companion Property and the Consistency of the Grammar

Our ambition is to build a grammar with a coverage of all of the most frequent phenomena of French syntax. Even if the hierarchical structure of the grammar makes it more compact ans eases the maintenance of its consistency, the size of the grammar may be important and the grammar offers no global view of its contents.

To verify the consistency of the grammar, it is necessary to check the behavior of each EPTD in the composition process with other EPTDs. A way of doing it is to parse corpora with the grammar but this is a very partial checking. Now, the formalism of IG provides a mechanism to verify the consistency of a grammar in a static way based on the EPTDs of the grammar without using parsing. The mechanism uses the *Companion Property.*

Originally, this property was introduced by [1] to perform lexical disambiguation with IG. Let us consider an interaction grammar.

[2] http://wikilligramme.loria.fr/doku.php?id=frilex

[3] http://abu.cnam.fr/DICO/mots-communs.html

Definition 3. *A companion of a polarized feature in an EPTD E_1 of the grammar is a polarized feature of an EPTD E_2 such that the first feature is saturated by the second feature in a merging of their nodes leading to a consistent PTD.*

What we mean with "a consistent PTD" is that the PTD resulting from the node merging has at least one model, which is a tree but not necessarily minimal and saturated.

For instance, consider the EPTD associated with the verb *"montrez"* in Figure 1. A companion of the positive feature `funct` → `obj` is the negative feature `funct` ← `obj|subjpred` of the EPTD associated with the clitic pronoun *"le"*. The notion of companion can be expressed a the template level: we compute systematically the companions of all polarized features of the EPTDs templates of the grammar. In this way, we limit the number of companions to compute.

So, for the EPTD template E_0 corresponding to the EPTD anchored with *"montrez"* in Figure 1 and for the positive feature `funct` → `obj` , we find 97 companions: 85 are right companions, that is, companions coming from EPTD templates for which the anchor is on the right of the anchor of E_0 after merging, and 12 are companions without order constraints on the anchor of their EPTD.

Among all the information given by the computation of all the companions, a particular part is immediately usable: the polarized features without companions. If they have no companion, their EPTDs cannot enter any parsing, which means that the EPTD template must be removed from the grammar or that there is some mistake in their definition.

6 Comparison with other French Grammars and Evaluation of the Grammar

There is very little work on the construction of French computational grammars from linguistic knowledge using semi-automatic tools. Historically, a very fruitful work was the PhD thesis of Candito [2] about the modular organization of TAGs, with an application to French and Italian. This thesis was a source of inspiration for the development of several French grammars.

A first grammar produced according to this approach and able to parse large corpora was FRMG [15]. FRMG falls within the TAG formalism and its originality lies in the use of specific operators on nodes to factorize trees: disjunction, guards, repetition and shuffling. As a consequence, the grammar is very compact with only 207 trees. Moreover, these trees are not written by hand but they are automatically produced from a multiple inheritance hierarchy of classes.

Another French grammar inspired by [2] is the French TAG developed by [3]. Like FRIGRAM, this grammar was written with XMG. Contrary to FRMG, it is constituted of classical TAG elementary trees, hence its more extensive form: it includes 4200 trees and essentially covers verbs. It was a purely syntactic grammar and then it was extended in the semantic dimension by [5] for generation.

To evaluate the soundness of FRIGRAM and to compare its coverage with other French grammars is problematic. The first difficulty is that there is no robust parser able to deal with IG. The tool developed so far (Leopar [6]) was designed to experiment, to test and to help grammar development. It was latter enriched with filtering algorithms to improve the supertagging stages of the parsing process. Nevertheless, it does not have any robust mechanism to deal with sentences that are not completely covered by the grammar. After filtering steps, deep parsing relies on an exhaustive search of tree description models which is an NP-hard task. As a consequence, Leopar can be used to parse sentence of length up to 15 words. FTB contains 3398 sentences of length lower than 15. Moreover, all linguistic phenomena present in real corpora, like the FTB, cannot be modeled through a lexicalized grammar: dislocation, coordination of non constituents, parenthetical clauses ... These phenomena require an extra-grammatical treatment, which is not yet implemented in Leopar. Thus, we consider the subset of sentence without explicit extra-grammatical phenomenon (parenthesis, reported speech); there are 2166 such sentences. The parser Leopar with the Frigram resource is able to parse 56.4% of the sentences considered.

Another way to evaluate a grammar coverage is to use test suites. Such suites must include not only positive examples but also negative examples to test the overgeneration of the grammar. There exists such a suite for French, the TSNLP[8], but unfortunately, it ignores a lot of phenomena that are very frequent in French. On the set of grammatical sentences of the TSNLP, Leopar and Frigram is able to parse 88% of the sentences. This is equivalent to the number achieved in [9] but the remaining sentences correspond to sentences that should be covered by the robustness of the parser rather than by the detailled grammar (unusual kind of coordination, sentence with incomplete negations, ...)

To try to deal with TSNLP drawbacks, we have designed our own test suite which si complementary to the TSNLP; it contains 874 positive sentences and 180 negative ones. 93% of the grammatical sentences are parsed and the ratio is 21% for ungrammatical sentences. The reader can find the test suite on a web page[4]. For the positive sentences, there is also the result of parsing in the form of a dependency graph. The variety of the examples gives a good idea of the coverage of Frigram and the richness of dependency graphs helps to understand the subtlety of the grammar.

7 Conclusion

The next step to go ahead with Frigram is to solve the bottleneck of the parser Leopar in order to parse raw corpora. We need to improve the efficiency of the parser to contain the possible explosion resulting from the increase of the grammar size in combination with the increased sentence length. It is also necessary to take robustness into account in the parsing algorithm and to add extra-grammatical procedures to deal with phenomena that go beyond the lexicalized grammar. For English, [13] is a first attempt to build a IG grammar

[4] `http://wikilligramme.loria.fr/doku.php?id=hmge_2013`

that should be extended in order to have a coverage equivalent to the one of Frigram.

References

1. G. Bonfante, B. Guillaume, and M. Morey. Polarization and abstraction of grammatical formalisms as methods for lexical disambiguation. In *11th International Conference on Parsing Technology, IWPT'09*, Paris, France, 2009.
2. M.-H. Candito. *Organisation modulaire et paramétrable de grammaires électroniques lexicalisées. Application au français et à l'italien,*. Thèse d'université, Université Paris 7, 1999.
3. B. Crabbé. *Représentation informatique de grammaires fortement lexicalises : application à la grammaire d'arbres adjoints*. thèse de doctorat, université Nancy2, 2005.
4. B. Crabbé, D. Duchier, C. Gardent, J. Le Roux, and Y. Parmentier. XMG : eXtensible MetaGrammar. *Computational Linguistics*, 39(3):1–66, 2013.
5. C. Gardent and Y. Parmentier. SemTAG: a platform for specifying Tree Adjoining Grammars and performing TAG-based Semantic Construction. In *45th Annual Meeting of the Association for Computational Linguistics*, pages 13–16, Prague, Tchèque, République, 2007.
6. B. Guillaume, J. Le Roux, J. Marchand, G. Perrier, K. Fort, and J. Planul. A Toolchain for Grammarians. In *Coling 2008*, pages 9–12, Manchester, Royaume-Uni, 2008.
7. B. Guillaume and G. Perrier. Interaction Grammars. *Research on Language and Computation*, 7:171–208, 2009.
8. S. Lehmann, S. Oepen, S. Regnier-Prost, K. Netter, V. Lux, J. Klein, K. Falkedal, F. Fouvry, D. Estival, E. Dauphin, H. Compagnion, J. Baur, L. Balkan, and D. Arnold. tsnlp — Test Suites for Natural Language Processing. In *Proceedings of COLING 1996, Kopenhagen*, 1996.
9. G. Perrier. A French interaction grammar. In *International Conference on Recent Advances in Natural Language Processing (RANLP 2007)*, pages 463–467, Borovets, Bulgaria, September 27-29 2007.
10. G. K. Pullum and B. C. Scholz. On the Distinction between Model-Theoretic and Generative-Enumerative Syntactic Frameworks. In *LACL 2001, Le Croisic, France*, volume 2099 of *Lecture Notes in Computer Science*, pages 17–43, 2001.
11. J. Rogers and K. Vijay-Shanker. Obtaining trees from their descriptions: an application to tree-adjoining grammars. *Computational Intelligence*, 10(4):401–421, 1994.
12. L. Romary, S. Salmon-Alt, and G. Francopoulo. Standards going concrete: from LMF to Morphalou. In M. Zock, editor, *COLING 2004 Enhancing and using electronic dictionaries*, pages 22–28, Geneva, Switzerland, August 29th 2004. COLING.
13. S. Tabatabayi Seifi. An interaction grammar for English verbs. In R. K. Rendsvig and S. Katrenko, editors, *Proceedings of the ESSLLI 2012 Student Session*, pages 160–169, Opole, Poland, August 6–17 2012.
14. K. Van den Eynde and P. Mertens. La valence : l'approche pronominale et son application au lexique verbal. *French Language Studies*, 13:63–104, 2003.
15. É. Villemonte De La Clergerie. Building factorized TAGs with meta-grammars. In *The 10th International Conference on Tree Adjoining Grammars and Related Formalisms - TAG+10*, pages 111–118, New Haven, CO, États-Unis, 2010.

Kahina: A Hybrid Trace-Based and Chart-Based Debugging System for Grammar Engineering

Johannes Dellert[1], Kilian Evang[2], and Frank Richter[1]

[1] University of Tübingen
[2] University of Groningen

Abstract. This paper provides an overview of the debugging framework Kahina, discussing its architecture as well as its application to debugging in different constraint-based grammar engineering environments. The exposition focuses on and motivates the hybrid nature of the system between source-level debugging by means of a tracer and high-level analysis by means of graphical tools.

1 Introduction

Several decades after their inception, the design, implementation and debugging of symbolic, constraint-based grammars is still a considerable challenge. Despite all efforts to modularize grammars, declarative constraints can have far-reaching unexpected side effects at runtime. The original grammar writers are often linguists with little experience in software development, meaning that in practice their declarative designs must be optimized for the algorithmic environment by iterative refinement. It is a major challenge in this scenario that the execution of even relatively simple grammars tends to result in many thousand computation steps. To make debugging of such grammars feasible at all, innovative and carefully designed debugging tools are needed.

In this paper, we present central aspects of the Kahina debugging framework, which was created to address these issues. We discuss its general architecture as well as its application to grammar debugging in two implementation platforms for constraint-based grammars, TRALE [3] and QType [15]. Unlike other grammar engineering environments, Kahina emphasizes the analysis of the execution model of declaratively designed grammars, with the goal of making the procedural effects of constraint systems more transparent. Developers can spot more easily where constraints have adverse side effects, and where an implementation can be optimized for efficiency. The main challenge in creating such a debugging system is to find appropriate ways to project functionally meaningful intervals from large sequences of computations onto intuitively comprehensible graphical displays. Within the overall display, the close connections between the different perceptual units need to be highlighted, while at the same time avoiding to produce redundant information that could easily overwhelm the user. A highly configurable selection of various perspectives on grammar execution is offered to make its contingencies explorable in a balanced fashion.

Kahina was initially developed as a graphical front end for TRALE's source-level debugger. TRALE is an interesting target, as it is explicitly designed for the implementation of grammatical constraints that stay close to the specification of theoretical HPSG grammars. As Melnik [13] finds in her comparison of the LKB [4] and TRALE for HPSG grammar implementation, the latter system requires fewer initial adjustments of theoretical grammars, at the price of a possibly stronger deviation of the processing model from non-technical user expectations. At the same time, the LKB traditionally provided more graphical debugging support and guidance to users, similar to the facilities featured by XLE [11], an implementation environment for LFG. Kahina aims at graphical debugging support for complex implementation systems such as TRALE, especially to help novice linguist users understand the underlying procedural model. It is designed to bridge the considerable gap between detailed but low-level, command-line based debugging for expert users, and the high-level view of chart-based tools, which (deliberately) hide many potentially relevant procedural details.

Beyond support for grammar implementation platforms, Kahina provides advanced general debugging facilities for logic programming. The system expands on earlier ideas for graphical Prolog debugging presented by e.g. Dewar & Cleary [8] and Eisenstadt [9], and is also inspired by SWI-Prolog's GUI tracer [16], the most mature visualization tool for Prolog processes currently available.[3]

Section 2 gives a bird's-eye view of the Kahina architecture, also outlining the process of implementing a debugging system. Sections 3–5 focus on the central features and functions of a debugging system for TRALE. Section 6 discusses chart displays as a central component of many grammar development systems, and provides a case study of how demands of different systems are accommodated. Section 7 critically evaluates Kahina's approach to grammar debugging, before Section 8 summarizes prominent features and points to open issues.

2 The Kahina Architecture

Kahina is written in Java and distributed under the GPL.[4] Core components are a GUI framework based on Swing, a State class for storing large amounts of step data, a message passing system (Controller) for communication among components, and a control agent system for automatization of user interactions.

Kahina is designed as a general framework for implementing debugging systems, by which we mean integrated graphical environments for analyzing computations consisting of hierarchically related steps, where each step may be associated with a source code location and other detail information for visual display. A debugging system is built by implementing specialized components using Kahina's classes and interfaces. The architecture of the TRALE debugger is shown in Fig. 1; a similar architecture has been used to implement graphical debuggers for different client systems, including SICStus and SWI Prolog.

[3] An earlier version of Kahina for TRALE was previously presented in a poster session at the HPSG conference 2010 [7].

[4] http://www.kahina.org

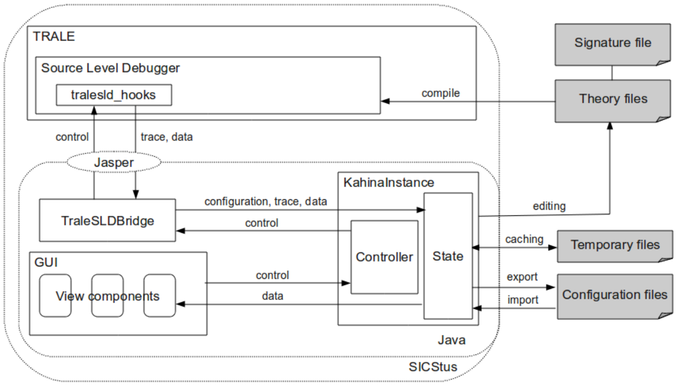


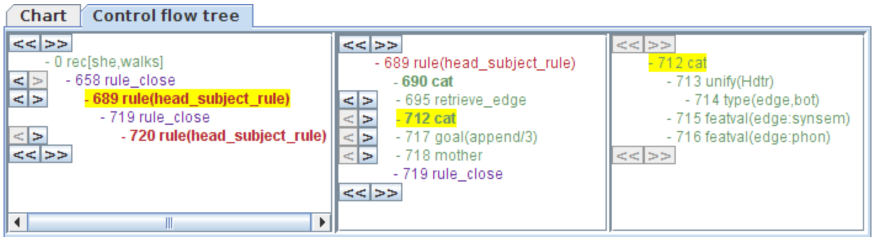
Fig. 1: Architecture of the Kahina-based TRALE debugger

On the Prolog side, the existing source-level debugger was extended by a control loop that interleaves with TRALE's execution process and communicates with Kahina to transmit step details and obtain tracing commands for controlling execution. Communication with Kahina is done via SICStus Prolog's Jasper library, which uses the Java Native Interface. On the Java side it is handled by a specialized **bridge** which translates low-level TRALE tracing information into Kahina's data model for storage and visualization. Another important application-specific Java component is a **step** class used by the State component, defining the data attributes associated with each step. Finally, a customizable configuration of **view components** defines how the step data are visualized.

3 Visualizing Parsing Processes

In this and the next two sections, we focus on the TRALE debugger and the features it offers to grammar engineers. In TRALE, a chart parser steps through the input from right to left, building all possible chart edges starting at each position before proceeding. Its principal operations or *steps* are:

– `rule_close`: A failure-driven loop over all phrase-structure rules in the grammar, takes a chart edge as input and recursively builds all edges that can be built with the selected edge as the leftmost child.
– `rule`: Tries to apply a phrase-structure rule with the input edge as leftmost daughter. Existing chart edges are used for covering the other daughters. Success leads to a new edge on which `rule_close` is called recursively.
– `retrieve_edge`: Retrieves a passive edge from the chart for use as a non-leftmost daughter in a `rule` application.
– `cat`: Applies a daughter description as part of a `rule` application.
– `goal`: Executes a procedural attachment as part of a `rule` application.
– `mother`: Applies a mother description as part of a `rule` application.



```
- 0 rec[she,walks]
  - 658 rule_close
    - 689 rule(head_subject_rule)
      - 719 rule_close
        - 720 rule(head_subject_rule)
```

```
- 689 rule(head_subject_rule)
  - 690 cat
  - 695 retrieve_edge
  - 712 cat
  - 717 goal(append/3)
  - 718 mother
  - 719 rule_close
```

```
- 712 cat
  - 713 unify(Hdtr)
    - 714 type(edge,bot)
  - 715 featval(edge:synsem)
  - 716 featval(edge:phon)
```

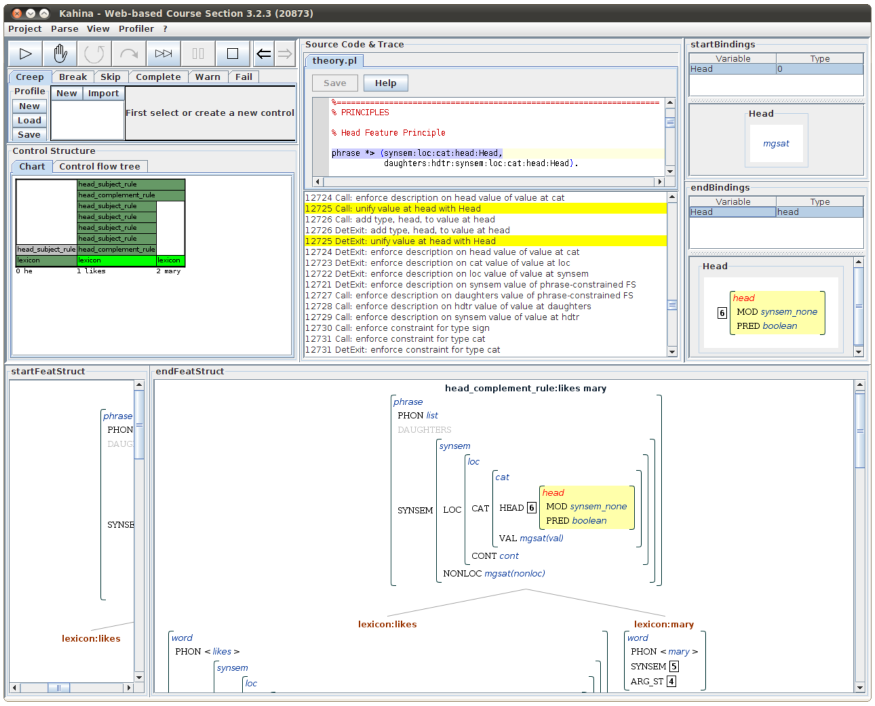
Fig. 2: Control flow tree of a parsing process with three levels of detail

TRALE comes with a command-line based tracer which treats these steps as *procedure boxes* following the Prolog tracing model of [1]. Kahina builds upon this notion and visualizes parsing processes as trees with goals as nodes and subgoals as child nodes. The resulting **control flow tree** is a central element of Kahina's GUI and is used to retrieve details on individual steps (see Section 4) by mouseclick. Each node is labeled with a numeric ID and a short description of the step, and is color-coded for status (Call, Exit, DetExit, Redo or Fail).

Since a single huge tree with thousands of parse steps cannot be navigated, the tree is split into three subviews that show different levels of detail (Fig. 2). The first subview shows a thinned-out version of the tree in which only the **cornerstones** are displayed, i.e. the `rule_close` and `rule` steps. Selection of cornerstone nodes controls which part of the tree occupies the second subview: it contains the descendants of the selected cornerstone down to the next cornerstones, displayed as leaves. Descendants of `cat`, `goal` and `mother` nodes are in turn displayed in the third subview when the respective step is selected.

Apart from their size, TRALE parsing processes are challenging to visualize due to their complex structure. Steps can be arranged in at least two meaningful tree structures. The **call tree**, in which goals have their subgoals as children, is visualized in the control flow tree through indentation. This tree roughly corresponds to the structure of clauses and subclauses in a logic program. The **search tree** is used in backtracking. Without backtracking, a search tree would be a long unary branch in which each step is the child of the step invoked before it, visualized by the top-to-bottom arrangement of steps in the control flow tree. When Prolog backtracks, the step that is the last active choicepoint is copied to represent the new invocation. The copy becomes a sibling of the original, starting a new branch. Kahina shows only one branch of the search tree at a time, focusing on visualizing the call tree, but at each choicepoint two arrow buttons permit browsing through siblings in the search tree. Access to earlier, "failed" branches is important because TRALE makes extensive use of failure-driven loops for exhausting the search space of possible edges. The user can flip through the different rules that were applied to any particular edge.

One way to inspect parsing processes is to trace them interactively, watching the tree grow step by step or in larger chunks. The available tracing commands are similar to those of a classical Prolog tracer (e.g. SICStus Prolog [2]). They



```
%==========================================================================
% PRINCIPLES

% Head Feature Principle

phrase *> (synsem:loc:cat:head:Head,
           daughters:hdtr:synsem:loc:cat:head:Head).
```

```
12724 Call: enforce description on head value of value at cat
12725 Call: unify value at head with Head
12726 Call: add type, head, to value at head
12726 DetExit: add type, head, to value at head
12725 DetExit: unify value at head with Head
12724 DetExit: enforce description on head value of value at cat
12723 DetExit: enforce description on cat value of value at loc
12722 DetExit: enforce description on loc value of value at synsem
12721 DetExit: enforce description on synsem value of phrase-constrained FS
12727 Call: enforce description on daughters value of phrase-constrained FS
12728 Call: enforce description on hdtr value of value at daughters
12729 Call: enforce description on synsem value of value at hdtr
12730 Call: enforce constraint for type sign
12731 Call: enforce constraint for type cat
12731 DetExit: enforce constraint for type cat
```

Fig. 3: Kahina's GUI showing tracing buttons, the chart, and all step detail views

are exposed as clickable buttons, shown in Fig. 3: `creep` advances to the next step. `fail` forces a not-yet-completed step to fail, making it possible to explore otherwise inaccessible parts of the search space. `auto-complete` completes the current step without interaction at substeps, but saves all information about the substeps for later inspection. `skip` does the same without collecting any tracing information. In effect it prunes parts of the process currently not of interest. `leap` proceeds without interaction up to the next breakpoint (see Section 5) or the end of the parsing process. Additional buttons pause or stop leap and auto-complete actions, or step through a history of recently selected steps. Since Kahina keeps old backtracking branches available – a feature that sets it apart from other graphical Prolog tracers such as that of SWI-Prolog [16] – it also supports full post-mortem analysis of a completed parsing process.

The **message console** complements the control flow tree as a navigation tool by displaying a timeline of tracing events. Events include step ports just like in a console-based Prolog tracer, but also user-generated and automatized tracing events: forced fails, auto-completes, skips, and leaps. All events associated with the currently selected step are highlighted, such as the Call port and the DetExit port of the selected `unify` step in Fig. 3.

4 Step Detail Views

Kahina's GUI contains different views for displaying details of the currently selected step. Steps are selected dynamically according to the context, in most cases through direct user interaction with either the control flow tree or the **chart display**, a graphical representation of the parse chart (Section 6).

The **source code editor** reacts to the selection of a step that corresponds to an element in the grammar (a rule application, a description, a constraint or a procedure) by marking the corresponding source code line. Computation steps are thus connected to the underlying grammar code. Fig. 3 shows the step details of a unification step within a constraint application (HPSG's Head Feature Principle) that fires during the application of a description of a rule to the mother feature structure. The defining source code line is colored. Syntax highlighting and basic editing are also supported.

While the source code view shows the reasons for what is happening, the **feature structure view** shows the structures that are being built as a result. For this view, Kahina draws on the Gralej[5] library. For steps that belong to rule applications, it shows the corresponding local tree, with feature structures for all daughters that have been or are being processed, and also the mother structure once it is being processed. The substructure that is being modified in each step is highlighted in yellow. For steps that belong to procedural attachments, the active goal is displayed with embedded graphical representations for feature structure arguments. A separate **bindings view** analogously shows feature structures associated with *variables* in the TRALE description language at every point during the application of a rule, constraint, or procedure.

In Fig. 3, the head complement rule is being applied to the input substring "likes mary", as shown by the local tree in the feature structure view. The current constraint binds the mother's SYNSEM:LOC:CAT:HEAD to a variable. The effect is seen in the bindings view, which compares the state before and after execution of the step. Before, the Head variable is still unbound, defaulting to *mgsat* (most general satisfier). Afterwards, it contains the value of the head feature.

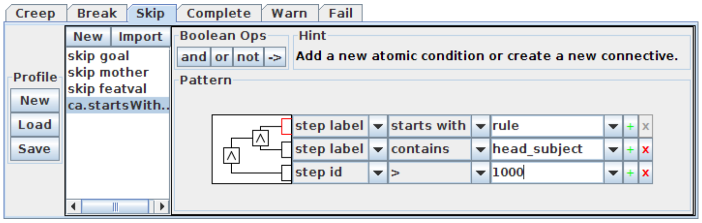
5 Automatization via Control Agents

Stepping through a parsing process manually and skipping or auto-completing irrelevant sections is error-prone and tiring. The `leap` command completes a parse without interaction for later post-mortem analysis, but may preserve too much information if only a certain part of the grammar should be debugged. Classic Prolog tracers, and also TRALE's console-based debugger, offer automatization via **leashing** and **breakpoints**. The user determines that certain types of steps should be **unleashed**, i.e. the tracer will simply proceed (`creep`) at them without asking what to do. Thus, the user can step through the parse faster, without having to issue a `creep` command at every step. Breakpoints single out a class

[5] http://code.google.com/p/gralej/

of steps of interest and then use the `leap` command to immediately jump to the next occurrence of that kind of step in the trace without further interaction.

Kahina generalizes leashing and breakpoints to the more powerful concept of **control agents**. A control agent is best seen as a simple autonomous agent consisting of a **sensor** which detects a pattern in individual steps or in a step tree, and an **actuator** which reacts to pattern matches by issuing tracing commands. Control agents are a method of describing and implementing intelligent behavior with the purpose of automatizing parts of the tracing and thereby of the grammar debugging process.



skip goal
skip mother
skip featval
ca.startsWith..

Add a new atomic condition or create a new connective.

step label | starts with | rule
step label | contains | head_subject
step id | > | 1000

Fig. 4: The control agent editor

The control agent editor (Fig. 4) serves to define, activate and deactivate control agents. Control agents are grouped according to their tracing command, and are accordingly called creep agents, skip agents, etc. While creep agents typically effect unleashing behavior, skip agents and complete agents complete steps without interaction, including all descendant steps, making these steps atomic for the purposes of tracing. Break agents act like breakpoints by exiting leap mode when their sensors fire. Fail agents let certain steps fail immediately, which is useful e.g. for temporarily deactivating some phrase-structure rule. Warn agents have actuators which not only stop a leap but also display a warning. Their sensor counts the pattern matches, and only fires after reaching a predefined threshold. Warn points detect infinite recursion or inefficiencies, e.g. when a procedural attachment predicate is called too often.

Due to the flexibility in their actuators, control agents are more powerful than traditional debugging mechanisms. Since they build on the existing tracing commands rather than introducing completely new concepts[6], we believe they are more intuitive to learn after a short exposure to manual tracing. The central technique for defining basic control agents is the creation of sensors by defining **step patterns**. Usually a simple substring check suffices to identify the relevant step class, but users can also build complex conditions with logical connectives and a range of elementary step tests, as exemplified in Fig. 4. Additionally, **source code sensors** that fire at steps associated with a code location can be created by opening the source code view's context menu for the line of interest

[6] In fact, Kahina sees the manually-tracing user as just another, external control agent with especially complex behavior.

(such as the line containing the name of a phrase structure rule, in order to catch rule application steps) and selecting the desired actuator. In sum, control agents provide an expressive yet accessible tool for automatization, which is crucial for efficient tracing of parse processes.

6 Specialized Charts for TRALE and QType

Apart from storing partial results in parsing algorithms based on dynamic programming, a **chart** summarizes the parsing process by storing successfully parsed constituents, including those which do not become part of a complete parse. The spans covered by constituents are usually symbolized as **edges** over the input string. By default, Kahina's chart display shows currently **active edges** (gray) and successfully built **passive edges** (green). Dependencies between chart edges are optionally displayed by highlighting the edges the selected edge is composed of, and the edges that it is part of. An unexpected parse can often be narrowed down to an unwanted edge for a substructure, while a missing parse is often due to an unrecognized constituent. This has made chart displays a central top-level parse inspection tool in the LKB and XLE. Practical grammar engineering in TRALE has heavily relied on a third-party chart display. While parsing algorithms usually only require storage of positive intermediate results, a grammar engineer often needs to find out why an expected substructure is missing. Kahina recognizes the equal importance of successful and failed edges: A failed attempt to build an edge can be displayed as a **failed edge** which is linked to the step where the respective failure occurred. A chart that does not contain failed edges, such as the LKB chart, does not provide direct access to such information.

As the chart display is fully integrated with the basic tracing functionality, the number of edges on the chart grows as a parse progresses. Every chart edge constitutes a link into the step tree, giving quick access to the step where the respective attempt to establish a constituent failed or succeeded.

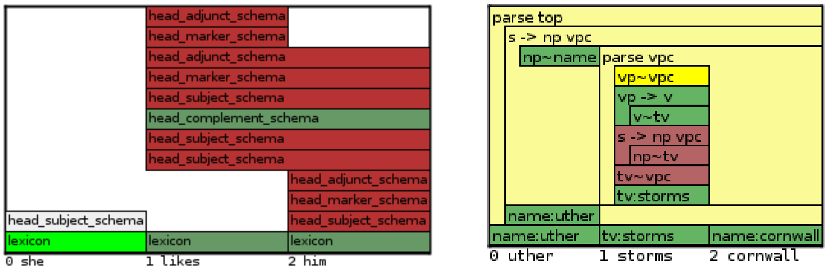


Fig. 5: Examples of the chart displays for TRALE and QType

Kahina's view components are designed for flexible customization to different grammar engineering systems, as demonstrated by two very different chart variants tailored to TRALE and QType, respectively. On the left side of Figure 5, we see a chart from a TRALE implementation of the English grammar by Pollard & Sag [14]. After recognizing all the input tokens, the right-to-left bottom-up parser has established an edge for the constituent “likes him” by means of the head complement schema. The chart also contains failed edges for several other schemas. The red edges below the successful edge represent failed attempts to form constituents by means of the respective lexicon edge. The edges above it represent failed attempts to build larger constituents from the successful edge. The lexicon edge for the input symbol “she” is currently being closed. The active application of the head subject schema will eventually lead to a successful parse.

Unlike TRALE, QType does not use dynamic programming. To still provide the functionality of a chart for analyzing left-corner parses, we defined a bridge that constructs a left-corner (LC) chart from the incoming step information. An intermediate stage of this chart for a parse process is displayed on the right side of Figure 5. The top-down prediction steps of the LC parser are visualized by inverted L-shaped prediction edges which wrap around the edges that were generated while attempting to complete the prediction. In the example, the active prediction edge labeled `s -> np vpc` ranging over the entire sentence indicates that QType is working on a parse based on the corresponding rule at the root level. The required `np` constituent has already been recognized by successful unification with the feature structure derived from the lexical entry for “uther”, and QType is now trying to complete the expected `vpc` constituent. The feature structure for the next token “storms” is of category `tv`. Attempts to interpret it directly as a `vpc`, or to once again apply the rule `s -> np vpc` to integrate it, have failed. The unary rule `vp -> v` was more successful, resulting in a feature structure of type `vp`. Its unification with the `vpc` specification in the rule definition will fail due to the mismatch between the subcat lists of the transitive verb and the postulated `vpc` structure, causing the correct phrase structure rule for verbs with one complement to be predicted next.

7 Discussion

One of the main contributions of Kahina to symbolic grammar engineering consists in the integration of two very different debugging paradigms of previous grammar debugging environments. We first compare Kahina to the old console-based source level debugger (SLD) for TRALE, and then to LKB and XLE as the two most popular environments which rely on a graphical interface and high-level representations.

During unifications and applications of descriptions, TRALE’s SLD displays feature structures only on demand, and for one step at a time. This makes it hard to recognize these operations as processes and to understand them. Kahina’s GUI makes it easy to go back and forth between steps and to quickly compare different points in time. Neither does the old SLD provide explicit information on how and when constraints and procedural attachments are executed. Given the complexities of TRALE’s constraint formalism, this is a severe problem, since goals are often suspended until some preconditions are fulfilled, and are only then executed in a delayed fashion. In larger parses, this behavior makes it virtually impossible to infer the current state of execution from a linear trace. Kahina's two-dimensional step tree with specialized nodes for representing suspended goals makes these aspects much more transparent.

In a classical tracer, decisions are always made locally, without any possibility to correct errors. A single erroneous skip command or a small mistake in a breakpoint definition may force the user to abort and restart a long tracing process. This necessarily leads to defensive behavior to prevent the loss of relevant information. As a result, traces tend to take longer than they would if information on past steps remained accessible. Kahina improves the accessibility of non-local information by its support for post-mortem inspection, but also with the simultaneous graphical display of multiple types of information.

The other major debugging paradigm of grammar engineering is characterized by chart-based high-level debugging (LKB and XLE). The LKB is the most relevant point of comparison for a TRALE debugger, since both systems are primarily designed for HPSG implementations. The LKB excels at detailed error messages for violations of formal conditions, whereas for more complex debugging tasks, especially those involving rule interactions, its tools are a lot less developed. It is standard debugging procedure to find a short sentence that exhibits the relevant problem, and then to inspect the parse chart in a time-consuming process which may require substantial intuition about the grammar. Once the problem is isolated in a small set of phrases, LKB's mechanism for interactive unification checks comes into play. Any two structures in the feature structure visualization can be tested for unifiability. If unification fails, the user receives explicit feedback on the reasons for failure. To trace the interaction between multiple constraints, intermediate results of successful unifications are used to chain together unification checks.

While Kahina also supports high-level parse chart inspection in the spirit of the LKB, interactive unification is only supported experimentally. Kahina compensates for this by its much more direct support for locating sources of error. Since every single unification or retrieval step is fully exposed by the source-level debugger, the inefficient process of narrowing down a problem only by means of the chart and test parses can in most cases be avoided. This reduces the importance of interactive unification, since the relevant failure can already be observed in the full context of the original problematic parse.

The LFG parser of XLE consists of a c-structure parsing component, and a constraint system that subsequently enforces f-structure constraints. A display of legal c-structures for which no valid f-structures could be found provides more fine-grained feedback about the reasons for structure invalidity than in the LKB. The XLE chart shows edges for partial matches of c-structure rules. While this provides some of the desired information on failed edges, compared to Kahina it still lacks information on rules that fail to apply because already the first constituent cannot be established. For the failed edges that are shown, XLE provides advanced analysis tools, also for finding out why no valid f-structure for a given c-structure could be found. All views offer options for extending the displayed information by invalid or incomplete structures, and selecting such a structure will highlight the parts which were missing in a c-structure rule or which violated some f-structure constraint. The exact way in which f-structure constraints are enforced still remains intransparent. This means that XLE lacks Kahina's support for grammar optimization, because the order in which the individual constraints are enforced is not exposed and cannot be manipulated.

To sum up the discussion, Kahina combines desirable properties of both the chart-based and the tracer-based grammar debugging paradigms. The main advantage of its hybrid approach lies in providing support for high-level parse inspection via the chart interface, while still making it possible to find out and to visualize on demand how exactly a parse is computed, effectively giving direct and fine-grained access to sources of undesirable behavior. The procedural orientation also supports the application of profiling techniques for grammar optimization, which is not possible if access is limited to high-level abstractions.

A downside of Kahina's hybrid nature may be that it could take longer for the beginner to develop an efficient grammar debugging workflow than in other environments, mainly because heavy use of control agents is necessary to keep step data extraction efficient and manageable. Moreover, the high-level components do not provide very advanced functionality in the area of interactive error diagnostics by default. Users must dig a little deeper in search of error diagnosis, but the explanations they obtain along the way are very detailed and complete. Finally, Kahina's layered system architecture makes it straightforward to integrate further high-level error diagnosis modules tailored to the user requirements. The prototype of a grammar workbench presented in [5] gives a glimpse of related ongoing developments. Recent extensions are not yet completely integrated, and need to be evaluated by grammar developers before they can become part of a release.

8 Conclusion

Kahina is a framework for building debugging environments for different grammar implementation systems. Its debuggers can take advantage of a modular system architecture, predefined bridges for communication with Prolog systems, a variety of view components, and external software modules. An implementation time of less than 500 person hours for the most recent debugger, compared with an estimated 3,000 person hours for the first, demonstrates the effectiveness of the framework in facilitating the development of interactive graphical debuggers for additional grammar engineering systems. In the present paper, we emphasized the application to TRALE in our discussion of Kahina's strategies for breaking up traces into conceptually meaningful chunks of information. The amount of information presented to the user, and the strategies by which it is gathered (in small steps, forcing shortcuts, leaping with or without recording information) can be customized by means of control agents that offer a very powerful abstraction layer for modifying tracing behavior.

The design of Kahina is tailored to a hybrid approach to grammar debugging which attempts to combine the advantages of a high-level chart-based view of parsing processes with the usefulness of a Prolog tracer for understanding every aspect of computational behavior and system performance. Initial experiences with the new TRALE debugger indicate that its low-level components especially help novice users to gain better insight into controlling parsing behavior within the system. The needs of expert users are currently catered for by a flexible chart display with high interactivity. This aspect of the system will be strengthened in future work through the development of tools that go beyond the analysis of parses.

References

1. Byrd, L.: Understanding the control flow of Prolog programs. In: Tamlund, S.A. (ed.) Proceedings of the 1980 Logic Programming Workshop. pp. 127–138 (1980)
2. Carlsson, M., et al.: SICStus Prolog User's Manual, Release 4.1.1. Tech. rep., Swedish Institute of Computer Science (December 2009)
3. Carpenter, B., Penn, G.: ALE 3.2 User's Guide. Technical Memo CMU-LTI-99-MEMO, Carnegie Mellon Language Technologies Institute (1999)
4. Copestake, A.: Implementing Typed Feature Structure Grammars. CSLI Publications, Stanford, CA (2002)
5. Dellert, J.: Extending the Kahina Debugging Environment by a Feature Workbench for TRALE. Diplomarbeit, Universität Tübingen (February 2012)
6. Dellert, J.: Interactive Extraction of Minimal Unsatisfiable Cores Enhanced By Meta Learning. Diplomarbeit, Universität Tübingen (January 2013)
7. Dellert, J., Evang, K., Richter, F.: Kahina, a Debugging Framework for Logic Programs and TRALE. The 17th International Conference on Head-Driven Phrase Structure Grammar (2010)
8. Dewar, A.D., Cleary, J.G.: Graphical display of complex information within a Prolog debugger. International Journal of Man-Machine Studies 25, 503–521 (1986)
9. Eisenstadt, M., Brayshaw, M., Paine, J.: The Transparent Prolog Machine. Intellect Books (1991)
10. Evang, K., Dellert, J.: Kahina. Web site (2013), access date: 2013-03-04. http://www.kahina.org/
11. Kaplan, R., Maxwell, J.T.: LFG grammar writer's workbench. Technical report, Xerox PARC (1996), ftp://ftp.parc.xerox.com/pub/lfg/lfgmanual.ps
12. Lazarov, M.: Gralej. Web site (2012), access date: 2013-03-05. http://code.google.com/p/gralej/
13. Melnik, N.: From "hand-written" to computationally implemented HPSG theories. Research on Language and Computation 5(2), 199–236 (2007)
14. Pollard, C., Sag, I.A.: Head-Driven Phrase Structure Grammar. The University of Chicago Press, Chicago (1994)
15. Rumpf, C.: Offline-Compilierung relationaler Constraints in QType. Online slides (2002), access date: 2013-03-07. http://user.phil-fak.uni-duesseldorf.de/ rumpf/talks/Offline-Compilierung.pdf
16. Wielemaker, J.: An overview of the SWI-Prolog programming environment. In: Mesnard, F., Serebenik, A. (eds.) Proceedings of the 13th International Workshop on Logic Programming Environments. pp. 1–16. Katholieke Universiteit Leuven, Heverlee, Belgium (2003)

SlaviCLIMB: Combining expertise for Slavic grammar development using a metagrammar

Antske Fokkens[1] and Tania Avgustinova[2]

[1] The Network Institute, VU University Amsterdam,
De Boelaan 1105, 1081 HV Amsterdam, The Netherlands
antske.fokkens@vu.nl,
[2] Department of Computational Linguistics, Saarland University
Campus, 66123 Saarbrcken, Germany
avgustinova@coli.uni-saarland.de

Abstract. We present the initial steps and outlook of SlaviCLIMB, the dynamic component of an effort to develop grammatical resources for Slavic languages around a shared core grammar. The requirements of this component are: (i) it should provide a platform for sharing similar but not identical analyses, (ii) it should be able to contain alternative analyses for the same phenomenon, (iii) it should be easy to use for syntacticians and grammar engineers and (iv) it should provide an interface for Slavicists not trained in grammar engineering allowing them to create grammars. Though the first two properties have been addressed in previous work, this paper is, to our knowledge, the first to address (iii) and (iv) within the context of metagrammars and multilingual grammar development.

Keywords: Metagrammar engineering, alternative analyses management, linguistic hypotheses testing, Slavic languages

1 Introduction

This paper introduces initial ideas on SlaviCLIMB, a metagrammar that is part of an effort to develop HPSG (Pollard and Sag, 1994) grammatical resources for a closed set of languages constituting the traditionally well-studied Slavic language family. The structural design of the applied framework (Avgustinova and Zhang, 2009) is centered on a common Slavic core grammar (SlaviCore) whose components are shared within the language group (Avgustinova, 2007). In the exploration phase of the project, SlaviCore is viewed as an extension of the typologically dedicated components of the LinGO Grammar Matrix (Bender et al., 2002, 2010), with SlaviCLIMB representing a dynamic grammar engineering component that captures language specific variations and facilitates grammar development for individual Slavic languages. We address the main goals of SlaviCLIMB and how we plan to achieve them by creating extensions to existing technology.

We define the following requirements for SlaviCLIMB: It should provide a platform for sharing analyses that are similar, but not identical in different Slavic languages, i.e. it should trigger variations based on parameters of linguistic properties. Second, it should be able to contain alternative analyses for the same phenomenon for more systematic exploration as suggested by Fokkens (2011). Third, it should be easy for syntacticians and grammar engineers to implement analyses. Finally, it should provide a customization interface, so that Slavicists can create grammars for Slavic languages by selecting implemented analyses without being an expert in grammar engineering or even HPSG.

This paper is structured as follows. In §2, we outline the background of this work. §3 illustrates the role of SlaviCLIMB through an example. We describe the currently available software and planned additions for SlaviCLIMB in §4 and conclude by comparing SlaviCLIMB to related work in §5.

2 Background

Limiting the typological coverage to a closed group of related languages provides a special perspective on cross-linguistic modeling. Our long term goal is to encode mutually interoperable analyses of a wide variety of linguistic phenomena, taking into account eminent typological commonalities and systematic differences. The parallel construction of HPSG-based Slavic grammatical resources employs a common core module in combination with language specific extensions and corpus-based grammar elaboration at all stages of the project. Playing a key role in linguistic research, alternative analyses are indispensable in modeling language variation. The Slavic language group offers "laboratory conditions" for experimenting with a novel type of phenomena-oriented alternatives-aware grammar modeling. Therefore, the explicit formulation of alternative analyses needs to be supported from the beginning. As the grammar engineering effort presented here is mainly concerned with enhancing the empirical validation of grammar sharing, the following aspects of multilingual parallel grammar development are of immediate interest to us: (i) what can be included in the common core module and what are the implications of such decisions; (ii) the practicability of phenomena-driven management of alternative analyses; (iii) robust guidelines supporting the grammar engineering activities in (i) and (ii).

The primary purpose of SlaviCLIMB is to support the development of SlaviCore. It follows the CLIMB[3] metagrammar engineering methodology (Fokkens, 2011, Fokkens et al., 2012) which is tightly linked to the Grammar Matrix customization system (Bender et al., 2010). The latter can create starter grammars covering a wide range of typological variation for a limited set of phenomena. Users can specify linguistic properties through a web interface and the customization system automatically generates an implemented grammar. The resulting grammar can then be extended manually. The CLIMB methodology extends the idea of generating code to a general approach for grammar development. The grammar engineer includes new implementations in a metagrammar which uses the original structure of the Grammar Matrix and part of its code generation software. Advantages of the approach include facilitating a phenomenon-based structure of the grammar, supporting alternative analyses for the same phenomenon and increasing modularity (Fokkens et al., 2012). However, we need

[3] Comparative Library Implementations with a Matrix Basis

to introduce a few adjustments to CLIMB in order to accommodate the third and fourth requirement mentioned above, namely: it should be easy to use for grammar engineers, and Slavicists not trained in HPSG should be able to create grammars. These adjustments will be presented in detail in §4. First, we'll illustrate the contribution of SlaviCLIMB through an example in §3.

3 Experimental modeling of the Slavic case system

3.1 The original SlaviCore model

The notion of case, as it is used in linguistic theory, is quite complex and refers to a range of linguistic properties at different levels of description (Avgustinova, 2007, p.25). The main distinction lies in understanding case as morphological marking of forms and as a syntactic notion linked to a particular function. This can be modeled by classifying the type *case* along two dimensions: *f-case* for the level of syntactic functions and *m-case* representing case marking. The functional case is an abstraction over regular case variation and language-specific constraints with respect to case marking. In the HPSG type hierarchy it introduces an additional dimension of case classification. This in particular means that in the lexical entry of a verb, the case value is of type respective f-case type. These abstract case values will be expanded to their concrete instances on the basis of lexical and contextual constraints, taking into consideration the relevant (language-specific) morphological and adpositional case marking.

Currently, SlaviCore contains an elaborate case hierarchy of 117 types that captures the wide variety of case marking found in Slavic languages, as well as (among others) the range of *functional subjective cases* (most subjects in Slavic are nominatives, but Polish also has dative subjects and Russian both datives and genitives, whereas Bulgarian exhibits a morphological base form), as well as *functional objective cases* (mostly accusative, but also genitive and instrumental in Polish and Russian, whereas Bulgarian exhibits a morphological oblique form). None of the Slavic languages uses all 117 types which forms the first motivation for a dynamic component.

3.2 Modeling with SlaviCLIMB

The customization machinery used in CLIMB can add underspecified types to the type hierarchy based on underspecified properties of lexical items or morphological markers.[4] A similar mechanism can be used to create the subtypes that link morphological cases to the right functional cases. As for now, SlaviCLIMB includes an option that triggers the top of the case hierarchy including the main functional cases, while the morphological case can be specified for the language in question. The following addition to SlaviCLIMB will provide a setup where the rest of the type hierarchy is based on direct observations in the language.

[4] This is a Grammar Matrix functionality described in Drellishak (2009).

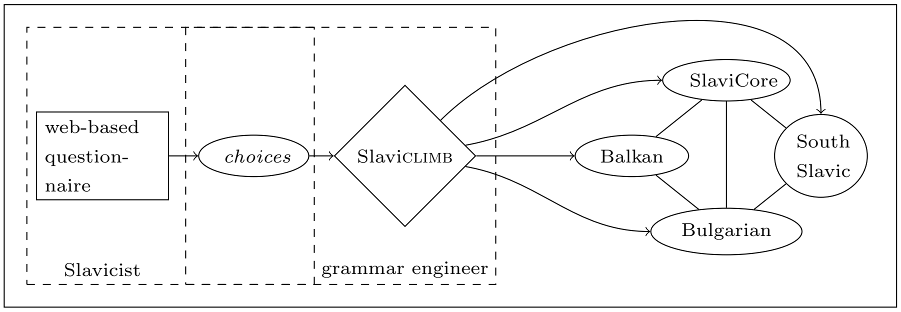


Fig. 1. Overview of development of a Bulgarian grammar with SlaviCLIMB

If a lexical item subcategorizes for a specific functional case, an option indicating the morphological case it subcategorizes for is used to generate the necessary subtypes. In this setup, the case hierarchy depends on the lexical items and morphemes that are defined in *choices*.[5] A Slavicist can define lexical items and morphemes with the relevant linguistic properties and the exact case hierarchy will be derived from these definitions. The case hierarchies thus created for individual languages can be compared to the original proposal in SlaviCore. Modeling the case hierarchy through SlaviCLIMB provides an environment to empirically verify Avgustinova's theoretical work. The original purpose of SlaviCLIMB to support parallel analyses allows grammar writers to either include the functional-morphological distinction or not. For instance, SlaviCLIMB supports Avgustinova's model for Bulgarian placing it in a wider Slavic context, but also the language specific solution adopted in the Bulgarian grammar BURGER (Osenova, 2010).

4 SlaviCLIMB

Figure 1 provides a schematic overview of how a Bulgarian grammar may be created using SlaviCLIMB. The ellipses indicate components that are generated by the system, the rectangle and diamond represent components requiring active interaction with experts. Slavicists fill out a web-based questionnaire, defining phenomena of a particular language and, if applicable, which analysis should be selected. They can indicate whether the phenomenon is a general Slavic property, a property of a subgroup of languages or language specific. This information does not influence the behavior of the ultimate grammar, but is represented in its architecture. The definitions provided by the Slavicist are stored in standardized form in *choices*, which allows us to verify hypotheses about language specific or shared behavior by comparing the *choices* for individual languages. *Choices* is the input to the SlaviCLIMB metagrammar developed by a grammar engineer.

[5] *choices* refers to the file that contains definitions of linguistic properties which trigger implementations in the metagrammar.

The metagrammar creates type definitions based on *choices* and places them in the common core, in a subcore or in the language specific file.

SlaviCLIMB currently covers the process starting with *choices* and can generate the complete SlaviCore and Russian Resource Grammar (RRG, Avgustinova and Zhang, 2010). It supports both the functional-morphological distinction and a simple case hierarchy. As such, it currently fulfills our first two requirements as defined in §1. A declarative version of CLIMB has been developed, which allows grammar engineers to define their analyses in TDL. This revision can serve as a basis to redesign SlaviCLIMB so that it is easier for grammar engineers to work with, which will fulfil the third requirement. It is technically possible for a Slavicist to manually define *choices*, but this requires extensive training. The Grammar Matrix architecture has a web interface that can be filled out by linguists without HPSG expertise, and detailed documentation on how to use the customization system (Fokkens et al., 2012). A Slavic specific version of the interface and documentation will be created to make SlaviCLIMB more accessible to Slavicists (fulfilling the fourth requirement). Designing such an interface is not a trivial task and will be ongoing research requiring both grammar engineering and Slavic expertise. Though the Slavicist will likely need support from grammar engineering experts to fix elements in the grammar or extend it further, a significant amount of work can be done through this method. This is shown by Borisova (2010) who modelled poly-personal agreement in Georgian almost exclusively using the Grammar Matrix customization system.

5 Conclusion

Methods for sharing analyses across grammars have been developed since the early days of grammar engineering. Grammar engineers of individual grammars that are part of the ParGram Project (Butt et al., 2002) meet biannually to compare and discuss their grammars. Kinyon et al. (2006) bring Candito's (1998) MetaGrammar to a cross-linguistic level through a study of verb second word order in several languages. Grammatical Formalism (GF, Ranta, 2009) reveals several similarities to our approach. It uses a metagrammar to divide expertise and share implementations across languages, including family-based cores for Romance and Scandinavian languages. Unlike SlaviCLIMB, it is mainly designed as a practical approach and even though it is based on a well-defined formalism and contains a vast amount of linguistic information, it is not related to a linguistic theory. Both the syntacticians and the engineers with knowledge of the languages in GF need a strong technical background, whereas SlaviCLIMB aims to provide a platform that allows also Slavicists without technical knowledge to create grammars. The CoreGram project (Müller 2013) is particularly interesting in relation to the work presented in this paper. It takes a bottom up approach of sharing analyses between HPSG grammars by means of a common core. If a property is found in several grammars, it is placed in a subcore shared by these grammars. It is in principle possible to gain knowledge about linguistic universals through CoreGram, but no strong claims are made on innateness of such principles. The closed set of languages in the Slavic family allow us to partially adapt a top down approach potentially leading to different insights than a bottom up approach. It should therefore be interesting to compare the insights of the two projects.

The methods mentioned above mainly resemble SlaviCore. To our knowledge, SlaviCLIMB forms the first proposal of a dynamic component that can combine expertise to empirically validate theoretical linguistic models.

References

Avgustinova, T.: Language Family Oriented Perspective in Multilingual Grammar Design. Linguistik International: Band 17. Peter Lang, Frankfurt am Main (2007)

Avgustinova, T., Zhang, Y.: Parallel grammar engineering for Slavic languages. In: Proceedings of GEAF, Singapore (2009)

Avgustinova, T., Zhang, Y.: Conversion of a Russian dependency treebank into HPSG derivations. In: Proceedings of TLT'9. (2010)

Bender, E.M., Flickinger, D., Oepen, S.: The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In: Proceedings of the Workshop on Grammar Engineering and Evaluation, Taipei, Taiwan (2002) 8–14

Bender, E.M., Drellishak, S., Fokkens, A., Poulson, L., Saleem, S.: Grammar customization. Research on Language & Computation **8**(1) (2010) 23–72

Borisova, I.: Implementing Georgian polypersonal agreement through the LinGO Grammar Matrix. Master's thesis, Saarland University (2010)

Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The parallel grammar project. In: Proceedings of the workshop on Grammar Engineering and Evaluation. (2002) 1–7

Candito, M.: Building parallel LTAG for French and Italian. In: Proceedings of ACL and ICCL, Montreal, Canada (1998) 211–217

Drellishak, S.: Widespread but Not Universal: Improving the Typological Coverage of the Grammar Matrix. PhD thesis, University of Washington (2009)

Fokkens, A.: Metagrammar engineering: Towards systematic exploration of implemented grammars. In: Proceedings of ACL, Portland, Oregon, USA (2011) 1066–1076

Fokkens, A., Avgustinova, T., Zhang, Y.: CLIMB grammars: three projects using metagrammar engineering. In: Proceedings of LREC 2012, Instanbul, Turkey (2012)

Fokkens, A., Bender, E.M., Gracheva, V.: LinGO grammar matrix customization system documentation (2012)

Kinyon, A., Rambow, O., Scheffler, T., Yoon, S., Joshi, A.K.: The metagrammar goes multilingual: A cross-linguistic look at the V2-phenomenon. In: Proceedings of the 8th International Workshop on TAG and Related Formalisms, Sydney, Australia (2006) 17–24

Müller, S.: The CoreGram project: Theoretical linguistics, theory development and verification. In: Proceedings of the workshop on HMGE. (2013)

Pollard, C., Sag, I.: Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago, USA (1994)

Osenova, P.: BUlgarian Resource Grammar Efficient and Realistic (BURGER) (2010)

Ranta, A.: The GF resource grammar library. Linguistic Issues in Language Technology **2**(2) (2009)

The CoreGram Project: A Brief Overview and Motivation

Stefan Müller

Freie Universität Berlin

Keywords: Universal Grammar, Head-Driven Phrase Structure Grammar, Multilingual Grammar Engineering, TRALE

This paper gives a brief overview of the CoreGram project. For a more detailed description of the project, further motivation and comparison with similar enterprises see Müller, 2013.

1 Overview and Motivation

The goal of the CoreGram project is the development of large scale computer processable grammar fragments of several languages that share a common core. The theoretical framework is Head-Driven Phrase Structure Grammar (HPSG, Pollard and Sag, 1994; Müller, 2007b). Currently we work on the following languages:

- German (Müller, 2007b, 2009b, 2012; Müller and Ørsnes, 2011a; Müller, To appear a)
- Danish (Ørsnes, 2009; Müller, 2009b, 2012; Müller and Ørsnes, 2011a, In Preparation)
- Persian (Müller, 2010; Müller and Ghayoomi, 2010; Müller, Samvelian and Bonami, In Preparation)
- Maltese (Müller, 2009a)
- Mandarin Chinese (Lipenkova, 2009; Müller and Lipenkova, 2009)
- Yiddish (Müller and Ørsnes, 2011b)
- English (Müller, 2009b, 2012)
- Spanish
- French

For the implementation we use the TRALE system (Meurers, Penn and Richter, 2002; Penn, 2004), which allows for a rather direct encoding of HPSG analyses (Melnik, 2007; Müller, 2013). The grammars of German, Danish, Persian, Maltese, and Mandarin Chinese are of non-trivial size and can be downloaded at http://hpsg.fu-berlin.de/Projects/CoreGram.html. They are also part of the next version of the Grammix CD-ROM (Müller, 2007a). The grammars of Yiddish and English are toy grammars that are used to verify cross-linguistic analyses of special phenomena and the work on Spanish and French is part of work in the Sonderforschungsbereich 632 which just started. See Bildhauer, 2008 for an implemented grammar of Spanish that will be converted into the format of the grammars mentioned above.

I believe that working out large scale computer-implemented grammars is the best way to verify the consistency of linguistic theories (Müller, 1999, Chapter 22; Bender, 2008). Much linguistic work is published in journal articles but the underlying assumptions of articles may be different so that it is difficult to imagine a coherent view that incorporates all insights. Even for articles by the same author it is not guaranteed that basic assumptions are shared between articles since it can take several years for individual papers until they are published. Hence, I believe that books are the right format for describing linguistic theories and ideally such theories are backed up by computer implementations. The larger fragments of the CoreGram project will be documented in a series of book publications. The first book in this series was Müller, 2007b, which describes a fragment of German that is implemented in the grammar BerliGram. Three further books are in preparation and will be submitted to the series Implemented Grammars by Language Science Press: one on the Persian Grammar developed in the PerGram project (Müller, Samvelian and Bonami, In Preparation), the Danish Grammar developed in the DanGram project (Müller and Ørsnes, In Preparation) and the Mandarin Chinese grammar developed in the ChinGram project.

2 The Poverty of the Stimulus and Motivation of Analyses

Huge progress has been made in recent years in the area of language acquisition. Input-based methods with an utterance-final bias have been shown to explain acquisition data better than maturation explanations or principle and parameters models (Freudenthal et al., 2006, 2007, 2009). Bod (2009) showed that English auxiliary inversion can be learned even with the evidence that Chomsky (1971, p. 29–33) claimed to be necessary and unavailable. The cited research shows that quite elaborate structures can be learned from the input alone and hence if there is any innate language-specific knowledge at all it is probably rather general as assumed for instance by Hauser, Chomsky and Fitch (2002). The consequence for linguistic research is that the existence of certain structures in one language does not imply that such structures are part of the grammar of all languages. So, the existence of object agreement in Basque cannot be used as evidence for object agreement projections (AgrO) in German. Neither can the existence of postpositions and agreement in Hungarian be seen as evidence for AgrO projections and hidden movement processes in English. Such complicated analyses cannot be motivated language internally and hence are not acquirable from input alone.

Instead of imposing constraints from one language onto other languages, a bottom-up approach seems to be more appropriate: Grammars for individual languages should be motivated language internally. Grammars that share certain properties can be grouped in classes. This makes it possible to capture generalizations about groups of languages and language as such. Let us consider some examples: German, Dutch, Danish, English and French. If we start developing grammars for German and Dutch, we find that they share a lot of properties: both are SOV and V2 languages, both have a verbal complex. One main difference is the order of elements in the verbal complex. The situation can be depicted as in Figure 1 on the facing page. There are some properties that are shared between German and Dutch (Set 3). For instance, the argument structure, a list containing descriptions of syntactic and semantic properties of arguments, and the linking of these arguments to the meaning of the sign is contained in Set 3. In addition the constraints for SOV languages, the verb position in V2 clauses and the fronting of a

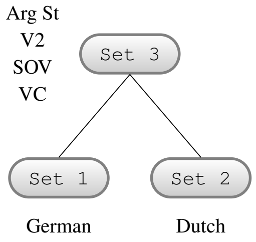


Figure 1. Shared properties of German and Dutch

constituent in V2 clauses are contained in Set 3. The respective constraints are shared between the two grammars. When we add another language, say Danish, we get further differences. While German and Dutch are SOV, Danish is an SVO language. Figure 2 shows the resulting situation: The top-most node represents constraints that hold for all languages (for instance the argument structure constraints, linking and V2) and the node below it contains constraints that hold for German and Dutch only.[1] For instance these constraints contain constraints regarding verbal complexes and SOV order. The union of Set 4 and Set 5 is the Set 3 of Figure 1.

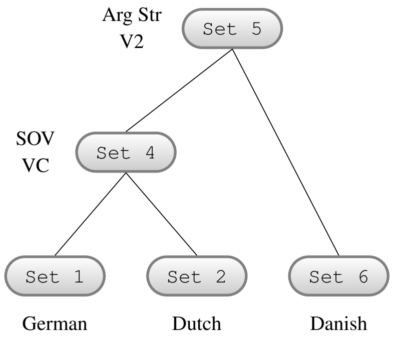


Figure 2. Shared Properties of German, Dutch, and Danish

If we add further languages further constraint sets will be distinguished. Figure 3 on the following page shows the situation that results when we add English and French. Again, the picture is not complete since there are constraints that are shared by Danish

[1] In principle, there could be constraints that hold for Dutch and Danish but not for German and for German and Danish, but not for Dutch. These constraints would be removed from Set 1 and Set 2 respectively and put into another constraint set higher up in the hierarchy. These sets are not illustrated in the figure and we keep the names Set 1 and Set 2 for the constraint sets for German and Dutch.

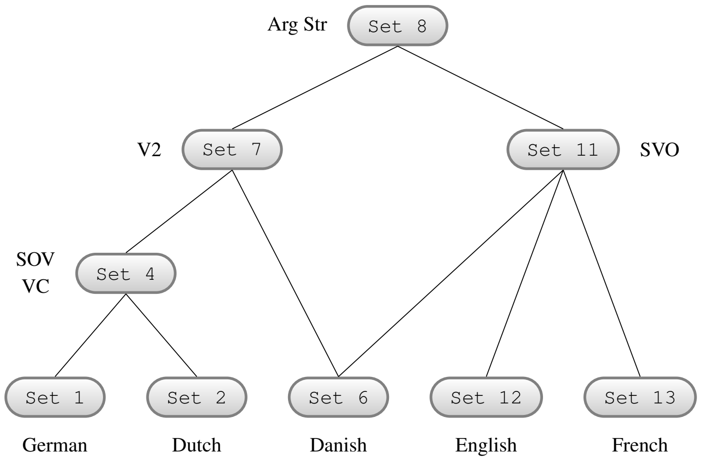


Figure 3. Languages and language classes

and English but not by French, but the general idea should be clear: by consequently working this way, we should arrive at constraint sets that directly correspond to those that were established in the typological literature.

It should be clear from what has been said so far that the goal of every scientist that works this way is to find generalizations and to describe a new language in a way that reuses theoretical constructs that have been found useful for a language that is already covered. However, as was explained above the resulting grammars should be motivated by data of the respective languages and not by facts from other languages. In situations where more than one analysis would be compatible with a given dataset for language X the evidence from language Y with similar constructs is most welcome and can be used as evidence in favor of one of the two analyses for language X. I call this approach the *bottom-up approach with cheating*: unless there is contradicting evidence we can reuse analyses that have been developed for other languages.

3 Coverage and Highlights

The grammars of German, Persian, and Danish are relatively big. The German grammar (BerliGram) was the first one that was implemented. It is an extension of the grammars that were developed for the individual chapters of the HPSG text book (Müller, 2007b). The Situation Semantics was replaced by a Minimal Recursion Semantics (MRS, Copestake et al., 2005). MRS allows for underspecification of scope so that a sentence like (1a) gets one representation from which the several scopings can be derived.

(1)	a.	dass	Max	wieder	alle	Fenster	öffnete
		that	Max	again	all	windows	opened

‘that Max opened all windows again’

b. *again*′(∀(CAUSE(open))); repetitiv
c. *again*′(CAUSE(∀(open))); repetitiv
d. CAUSE(*again*′(∀(open))); restitutiv

The example in (1a) is an example of lexical subscoping: *öffnen* is lexically decomposed into CAUSE(open) and the *wieder* can scope below the CAUSE operator although there is no decomposition in syntax.

Apart from the modification of the semantics component further special phenomena were implemented. For instance an analysis of multiple frontings (Müller, 2003), something that is unique among the existing HPSG implementations. For a discussion of approaches to constituent order that are incompatible with the multiple fronting data see Müller, 2005, In Preparation. Furthermore the analysis of depictives was added (Müller, 2008). Analyses that have been implemented in my earlier grammars of German have not yet been transferred to BerliGram.

The Danish grammar is documented in a 500+ page book, which is not complete yet. The following examples show in a compact way the interaction of several phenomena: passive with promotion of either the direct object or the indirect object (2a,c), passive and pronoun shift (2b,d), partial fronting and object shift in (2b,d):

(2) a. Bjarne bliver ikke anbefalet den.
Bjarne.NOM is not recommended it.ACC
'The book is not recommended to Bjarne.'

b. ? Anbefalet bliver Bjarne den ikke.
recommended is Bjarne.NOM it.ACC not
'The book is not recommended to Bjarne.'

c. Bogen bliver ikke anbefalet ham.
book.DEF.NOM is not recommended him.ACC
'The book is not recommended to him.'

d. ? Anbefalet bliver bogen ham ikke.
recommended is book.DEF.NOM him.ACC not
'The book is not recommended to him.'

The Mandarin Chinese grammar was implemented with the help of Jia Zhongheng. We used the description in Li and Thompson, 1981 as basis of our implementation. Among the things that are special are NPs that contain classifiers (3) and change of part of speech by reduplication as in (4).

(3)
na4 liang4 hong2 de che1 xiu4.le
that CL red DE car rust.ASP
'That red car rusts.'

The adjective gao1xing4 is converted into an adverb by forming the pattern AABB from the original adjective AB, that is gao1 is doubled and xing4 is doubled as well.

(4)
ta1 gao1gao1xing4xing4 you3yong3
he AABB=happily swims
'He swims happily.'

The Persian grammar is a larger fragment, which needs to be documented. The examples in (5) show lightverb constructions, which are an important feature of the language. (5a) shows that the future auxiliary can interrupt the preverb verb sequence of lightverbs. (5b) shows an example with the negation prefix and pro-drop.

(5) a. Man in kr r anjm xham dd.[2]
I this job DOM performance will-1SG gave
'I will do this work.'

b. mard râ dust nadât.
man DOM friend NEG-had
'he/she does not like the man.'

The Maltese grammar is an implementation of the description of Fabri, 1993. Fabri works in the framework of Lexical Decomposition Grammar, which is also a lexical framework and his analysis were translatable into HPSG without great efforts. The examples in (6) show definiteness marking. (6b) shows assimilation and (6c) shows clitics:

(6) a. Il-komunist xejjer lil-l-papa.
DEF-communist winks(3msg) Case-DEF-pope(msg)
'the communist winks the pope.'

b. It-terrorist bagat l-ittr-a lil-l-president.
DEF-terrorist sent DEF-letter-F Case-DEF-president
'The terrorist sent the president the letter.'

c. It-terrorist bagat=hie=l.
DEF-terrorist send.3M.SG=3F.SG=3M.SG
'The terrorist sent it to him.'

(6c) is ambiguous. There is a reading with clitic left dislocation. Both readings are found by the grammar.

4 Basic Assumptions

4.1 Valence

We assume that valence is represented in a uniform way across languages. Arguments of a head are represented in the ARG-ST list (Pollard and Sag, 1994, Chapter 9). They are mapped to the valence features SPR and COMPS in a language dependent fashion. For instance, English and Danish map the subject to the SPR list and the other arguments to COMPS. Danish inserts an expletive in cases in which there is no element that can be mapped to SPR, while English does not do this. German differs from both languages in mapping all arguments of finite verbs to the COMPS list.

The arguments in the ARG-ST list are ordered according to the obliqueness hierarchy of Keenan and Comrie (1977), which plays a role in the analysis of a variety of phenomena. The elements of the ARG-ST list are linked to the semantic roles that a certain head has to fill. Since the traditional role labels like agent and patient are problematic, we adopt Dowty's proto-role approach (Dowty, 1991). We use ARG1, ARG2, and so on as role labels.

[2] Karimi-Doostan, 1997, p. 73.

4.2 Constituent Structure and Constituent Order

Classical HPSG came with very view immediate dominance schemata: Head-Complement Schema, Head-Specifier Schema, Head-Adjunct Schema, the Head-Filler Schema for binding off unbounded dependencies, and the Head-Extra Schema for binding off clause bound nonlocal dependencies. Since Sag, 1997 many HPSG analyses have a more constructional flavor, that is, specific subconstructions of these general schemata are introduced (Sag, 2010). In the CoreGram project we stay within the old tradition of HPSG and keep the rather abstract dominance schemata. However, it is possible to state further constraints on the respective structures. So rather than having several very specific instances of the Head-Filler Schema, we have very few ones, for instance, for verb second clauses and relative clauses and formulate additional implicational constraints that constrain actual instances of head filler phrases further if the antecedent of the implicational constraint is true. Since the schemata are rather general they can be used for all languages under consideration so far. Of course the languages differ in terms of constituent order, but this can be dealt with by using linearization rules that are sensitive to features whose values are language specific. For instance, all heads have a feature INITIAL. The value is '+', if the head has to be serialized before its complements and '–' if it follows its complements. German and Persian verbs are INITIAL −, while English, Danish, Mandarin Chinese and Maltese verbs are INITIAL +.

We assume binary branching structures and hence we get the structures in (7) for English and the corresponding German example:

(7) a. He [[gave the woman] a book].
b. er [der Frau [ein Buch gab]]
he the woman a book gave

The LP rules enforce that *gave* is linearized before *the woman* and *gave the woman* is linearized before *a book*.

The scrambling of arguments is accounted for by ID schemata that allow the combination of a head with any of its arguments independent of the position an element has in the valence list of its head. Non-scrambling languages like English on the other hand combine heads with their complements in a strict order: the least oblique element is combined with the head first and then the more oblique complements follow. Non-scrambling languages with head-final order take the last element from the valence list first.

4.3 Morphology and Lexical Rules

We follow a lexical rule based approach to morphology. Lexical rules are basically unary branching trees that license new lexical items. A lexical rule can add or subtract to the phonology (in implementations the orthography) of an input item. For instance, it is possible to analyze the complex morphological patterns that we observe in Semitic languages by mapping a root consisting of consonants to a full-fledged stem or word that has the appropriate vowels inserted. We follow Bresnan and Mchombo (1995) in assuming the Lexical Integrity Principle. This means that all morphological combinations have to be done by lexical rules, that is, fully inflected forms are part of the lexicon, most of them being licensed by productive lexical rules.

Lexical rules do not have to change the phonology/orthography of the item they apply to. For instance lexical rules can be used to license further lexical items with extended or reduced valence requirements. As was argued in Müller, 2002, 2006 resultative constructions should be treated lexically. So there is a lexical rule that maps the stem *fisch-* of the intransitive version of the verb *fischen* ('to fish') onto a stem *fisch-* that selects for a secondary predicate (adjective or PP) and the subject of this predicate.

(8) Er fischt den Teich leer.
he fishes the pond empty

5 Implementation Details

5.1 TRALE

The grammars are implemented in TRALE. TRALE implements typed feature descriptions. Every grammar consists of a signature (a type hierarchy with feature introduction and appropriateness constraints) and a theory that states constraints on objects of these types. TRALE is implemented in Prolog and comes with an implementation of relational constraints that maps the TRALE relations to Prolog relations. TRALE has two parsers: a standard bottom-up chart parser and a linearization parser. The CoreGram project uses the standard bottom-up parser. Both parsers use a phrase structure backbone.

Compared to other systems like the LKB (Copestake, 2002) the expressive power of the description language is high (see also Melnik, 2007). This allows for the rather direct implementation of analyses that are proposed by theoretical linguists. The following descriptive devices are used in the theory and are provided by TRALE. The references point to papers who argue for such constructs.

- empty elements (Kiss, 1995; Meurers, 1999; Müller, 2007b; Levine and Hukari, 2006; Bender, 2000 und Sag, Wasow and Bender, 2003, p. 464; Borsley, 2004, Section 3.3; Müller, To appear b),
- relational constraints (Pollard and Sag, 1994; Bouma et al., 2001),
- complex antecedents in implicational constraints (Meurers, 2000, p. 207; Koenig and Davis, 2004, p. 145, 149; Müller, 2007b, p. 145, Section 10.3; Bildhauer and Cook 2010, p. 75),
- cyclic structures (Engdahl and Vallduví, 1994, p. 56, Meurers, 1999, Meurers, 2001, p. 176, Samvelian, 2007, p. 638), and
- a morphology component that has the expressive power that is needed to account for nontrivial morphological phenomena.

5.2 Setup of CoreGram

The grammars are organized in one directory for every language. The respective directories contain a subdirectory named Core-Grammar. This directory contains files that are shared between the grammars. For instance the file core-macros.pl contains macros that are or can be used by all languages. For every language there is a load file that loads the files from the core grammar directory that are relevant for the respective language. So, for instance english.pl, french.pl, and danish.pl all load nom-acc.pl since these languages are nominative-accusative languages. These files also contain code for loading macros and constraints for languages that do not form a verbal complex, while german.pl does load the files for cluster-forming languages. These files directly correspond to the constraint sets that were discussed in Section 2.

The possibility to specify type constraints makes it possible to specify constraints that hold for a certain construction cross-linguistically in a file that is loaded by all grammars and restrict structures of this type further in language particular files.

Lexical rules are also described by feature descriptions and organized in type hierarchies (Meurers, 2001). Like other constraints the constraints on lexical rules can be shared.

6 Measuring Progress

Much to the frustration of many linguists the contribution of certain approaches to progress in linguistics is rather unclear. Many proposals do not extend the amount of data that is covered in comparison to analyses that were developed during the 1980s in the framework of GB and other, non-transformational frameworks. In comparison the methodology described in Section 2 leads to grammars with increasing coverage and analyses that are improved by cross-linguistic considerations.

The TRALE system has been combined with [incr tsdb()], a software for systematic grammar profiling (Oepen and Flickinger, 1998). The grammars are accompanied with a set of example phrases that can be analyzed by the grammar. In addition the test suite files contain ungrammatical word sequences from the literature and ungrammatical sequences that were discovered during the grammar development process. Since TRALE has a chart display that makes it possible to inspect the parse chart, it is possible to inspect all linguistic objects that are licensed by the grammar, even if they do not play a role in analyzing the particular sentence under consideration. The result of this careful inspection is a collection of ungrammatical word sequences that no theoretical linguist would have been able to come up with since it is very difficult to find all the side effects that an analysis might have that is not constrained sufficiently. These negative examples are distributed with the grammars and book publications and can help theoretical and computational linguists improve their theories and implementations.

After changing a grammar the sentences of the respective test suite are parsed and the result can be compared to previous results. This way it is ensured that the coverage of grammars is extended. If constraints in files are changed that are shared with other grammars the respective grammars are tested as well. The effects that changes to grammar X cause in grammar Y are often unexpected and hence it is very important to do systematic testing.

7 Conclusions

We argued that linguistic theories have reached a level of complexity that cannot be handled by humans without help by computers. We discussed a method for constructing surface-oriented theories by extending the number of languages that are considered and finally provided a brief description of basic assumptions and the basic setup of the CoreGram project.

Bibliography

Bender, E. M. 2000. *Syntactic Variation and Linguistic Competence: The Case of AAVE Copula Absence*. Ph. D.thesis, Stanford University.

Bender, E. M. 2008. Grammar Engineering for Linguistic Hypothesis Testing. In N. G. et. al. (ed.), *Proceedings of the Texas Linguistics Society X Conference*, pp 16–36, Stanford CA: CSLI Publications ONLINE.

Bildhauer, F. 2008. *Representing Information Structure in an HPSG Grammar of Spanish*. Dissertation, Universität Bremen.

Bildhauer, F. and Cook, P. 2010. German Multiple Fronting and Expected Topic-Hood. In S. Müller (ed.), *Proceedings of the 17th International Conference on Head-Driven Phrase Structure Grammar*, pp 68–79, Stanford: CSLI Publications.

Bod, R. 2009. From Exemplar to Grammar: Integrating Analogy and Probability in Language Learning. *Cognitive Science* 33(4), 752–793.

Borsley, R. D. 2004. An Approach to English Comparative Correlatives. In Müller (2004), pp 70–92.

Bouma, G., Malouf, R. and Sag, I. A. 2001. Satisfying Constraints on Extraction and Adjunction. *Natural Language and Linguistic Theory* 19(1), 1–65.

Bresnan, J. and Mchombo, S. A. 1995. The Lexical Integrity Principle: Evidence from Bantu. *Natural Language and Linguistic Theory* 13, 181–254.

Chomsky, N. 1971. *Problems of Knowledge and Freedom*. London: Fontana.

Copestake, A. 2002. *Implementing Typed Feature Structure Grammars*. Stanford: CSLI Publications.

Copestake, A., Flickinger, D. P., Pollard, C. J. and Sag, I. A. 2005. Minimal Recursion Semantics: an Introduction. *Research on Language and Computation* 4(3), 281–332.

Dowty, D. R. 1991. Thematic Proto-Roles and Argument Selection. *Language* 67(3), 547–619.

Engdahl, E. and Vallduví, E. 1994. Information Packaging and Grammar Architecture: A Constraint-Based Approach. In E. Engdahl (ed.), *Integrating Information Structure into Constraint-Based and Categorial Approaches*, pp 39–79, Amsterdam: ILLC, DYANA-2 Report R.1.3.B.

Fabri, R. 1993. *Kongruenz und die Grammatik des Maltesischen*. Tübingen: Niemeyer.

Freudenthal, D., Pine, J. M., Aguado-Orea, J. and Gobet, F. 2007. Modeling the Developmental Patterning of Finiteness Marking in English, Dutch, German, and Spanish Using MOSAIC. *Cognitive Science* 31(2), 311–341.

Freudenthal, D., Pine, J. M. and Gobet, F. 2006. Modeling the Development of Children's Use of Optional Infinitives in Dutch and English Using MOSAIC. *Cognitive Science* 30(2), 277–310.

Freudenthal, D., Pine, J. M. and Gobet, F. 2009. Simulating the Referential Properties of Dutch, German, and English Root Infinitives in MOSAIC. *Language Learning and Development* 5(1), 1–29.

Hauser, M. D., Chomsky, N. and Fitch, W. T. 2002. The Faculty of Language: What Is It, Who Has It, and How Did It Evolve? *Science* 298, 1569–1579.

Karimi-Doostan, G. 1997. *Light Verb Constructions in Persian*. Ph. D.thesis, Department of Language and Linguistics, University of Essex.

Keenan, E. L. and Comrie, B. 1977. Noun Phrase Accessibility and Universal Grammar. *Linguistic Inquiry* 8(1), 63–99.

Kiss, T. 1995. *Infinite Komplementation. Neue Studien zum deutschen Verbum infinitum*. Tübingen: Niemeyer.

Koenig, J.-P. and Davis, A. R. 2004. Raising Doubts about Russian Impersonals. In Müller (2004).

Levine, R. D. and Hukari, T. E. 2006. *The Unity of Unbounded Dependency Constructions*. Stanford: CSLI Publications.

Li, C. N. and Thompson, S. A. 1981. *Mandarin Chinese. A Functional Reference Grammar*. Berkeley and Los Angeles: University of California Press.

Lipenkova, J. 2009. *Serienverbkonstruktionen im Chinesischen und ihre Analyse im Rahmen von HPSG*. Masters Thesis, Institut für Sinologie, Freie Universität Berlin.

Melnik, N. 2007. From "Hand-Written" to Computationally Implemented HPSG Theories. *Research on Language and Computation* 5(2), 199–236.

Meurers, W. D. 1999. German Partial-VP Fronting Revisited. In G. Webelhuth, J.-P. Koenig and A. Kathol (eds.), *Lexical and Constructional Aspects of Linguistic Explanation*, pp 129–144, Stanford: CSLI Publications.

Meurers, W. D. 2000. Lexical Generalizations in the Syntax of German Non-Finite Constructions. Arbeitspapiere des SFB 340 No. 145, Eberhard-Karls-Universität, Tübingen.

Meurers, W. D. 2001. On Expressing Lexical Generalizations in HPSG. *Nordic Journal of Linguistics* 24(2), 161–217.

Meurers, W. D., Penn, G. and Richter, F. 2002. A Web-Based Instructional Platform for Constraint-Based Grammar Formalisms and Parsing. In D. Radev and C. Brew (eds.), *Effective Tools and Methodologies for Teaching NLP and CL*, pp 18–25.

Müller, S. 1999. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Tübingen: Niemeyer.

Müller, S. 2002. *Complex Predicates: Verbal Complexes, Resultative Constructions, and Particle Verbs in German*. Stanford: CSLI Publications.

Müller, S. 2003. Mehrfache Vorfeldbesetzung. *Deutsche Sprache* 31(1), 29–62.

Müller, S. (ed.). 2004. *Proceedings of the 11th International Conference on Head-Driven Phrase Structure Grammar*, Stanford, CSLI Publications.

Müller, S. 2005. Zur Analyse der deutschen Satzstruktur. *Linguistische Berichte* 201, 3–39.

Müller, S. 2006. Phrasal or Lexical Constructions? *Language* 82(4), 850–883.

Müller, S. 2007a. The Grammix CD Rom. A Software Collection for Developing Typed Feature Structure Grammars. In T. H. King and E. M. Bender (eds.), *Grammar Engineering across Frameworks 2007*, Stanford: CSLI Publications.

Müller, S. 2007b. *Head-Driven Phrase Structure Grammar: Eine Einführung*. Tübingen: Stauffenburg Verlag, first edition.

Müller, S. 2008. Depictive Secondary Predicates in German and English. In C. Schroeder, G. Hentschel and W. Boeder (eds.), *Secondary Predicates in Eastern European Languages and Beyond*, pp 255–273, Oldenburg: BIS-Verlag.

Müller, S. 2009a. A Head-Driven Phrase Structure Grammar for Maltese. In B. Comrie, R. Fabri, B. Hume, M. Mifsud, T. Stolz and M. Vanhove (eds.), *Introducing Maltese Linguistics*, pp 83–112, Amsterdam: John Benjamins Publishing Co.

Müller, S. 2009b. On Predication. In S. Müller (ed.), *Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar*, pp 213–233, Stanford: CSLI Publications.

Müller, S. 2010. Persian Complex Predicates and the Limits of Inheritance-Based Analyses. *Journal of Linguistics* 46(3), 601–655.

Müller, S. 2012. On the Copula, Specificational Constructions and Type Shifting. Ms. Freie Universität Berlin.

Müller, S. 2013. The CoreGram Project: Theoretical Linguistics, Theory Development and Verification. Ms. Freie Universität Berlin.

Müller, S. In Preparation. *German Sentence Structure: An Analysis with Special Consideration of So-Called Multiple Fronting*. Berlin: Language Science Press.

Müller, S. To appear a. Artenvielfalt und Head-Driven Phrase Structure Grammar. In J. Hagemann and S. Staffeldt (eds.), *Syntaktische Analyseperspektiven*, Tübingen: Stauffenburg.

Müller, S. To appear b. Elliptical Constructions, Multiple Frontings, and Surface-Based Syntax. In G. Jäger, P. Monachesi, G. Penn and S. Wintner (eds.), *Proceedings of Formal Grammar 2004, Nancy*, Stanford: CSLI Publications.

Müller, S. and Ghayoomi, M. 2010. PerGram: A TRALE Implementation of an HPSG Fragment of Persian. In *Proceedings of 2010 IEEE International Multiconference on Computer Science and Information Technology – Computational Linguistics Applications (CLA'10). Wisła, Poland, 18–20 October 2010*, pp 461–467, Polnish Information Processing Society.

Müller, S. and Lipenkova, J. 2009. Serial Verb Constructions in Chinese: An HPSG Account. In S. Müller (ed.), *Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar, University of Göttingen, Germany*, pp 234–254, Stanford: CSLI Publications.

Müller, S. and Ørsnes, B. 2011a. Positional Expletives in Danish, German, and Yiddish. In S. Müller (ed.), *Proceedings of the 18th International Conference on Head-Driven Phrase Structure Grammar, University of Washington, U.S.A.*, pp 167–187, Stanford: CSLI Publications.

Müller, S. and Ørsnes, B. 2011b. Positional Expletives in Danish, German, and Yiddish. In S. Müller (ed.), *Proceedings of the 18th International Conference on Head-Driven Phrase Structure Grammar*, pp 167–187, Stanford: CSLI Publications.

Müller, S. and Ørsnes, B. In Preparation. *Danish in Head-Driven Phrase Structure Grammar*. Berlin: Language Science Press.

Müller, S., Samvelian, P. and Bonami, O. In Preparation. *Persian in Head-Driven Phrase Structure Grammar*. Berlin: Language Science Press.

Ørsnes, B. 2009. Preposed Negation in Danish. In S. Müller (ed.), *Proceedings of the 16th International Conference on Head-Driven Phrase Structure Grammar, University of Göttingen, Germany*, pp 255–275, Stanford: CSLI Publications.

Oepen, S. and Flickinger, D. P. 1998. Towards Systematic Grammar Profiling. Test Suite Technology Ten Years After. *Journal of Computer Speech and Language* 12(4), 411–436, (Special Issue on Evaluation).

Penn, G. 2004. Balancing Clarity and Efficiency in Typed Feature Logic Through Delaying. In D. Scott (ed.), *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pp 239–246, Barcelona, Spain.

Pollard, C. J. and Sag, I. A. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, IL and London: University of Chicago Press.

Sag, I. A. 1997. English Relative Clause Constructions. *Journal of Linguistics* 33(2), 431–484.

Sag, I. A. 2010. English Filler-Gap Constructions. *Language* 86(3), 486–545.

Sag, I. A., Wasow, T. and Bender, E. M. 2003. *Syntactic Theory: A Formal Introduction*. Stanford: CSLI Publications, second edition.

Samvelian, P. 2007. A (Phrasal) Affix Analysis of the Persian Ezafe. *Journal of Linguistics* 43, 605–645.

Time Travel in Grammar Engineering: Using a Metagrammar to Broaden the Search Space[⋆]

Antske Fokkens[1] and Emily M. Bender[2]

[1] The Network Institute, VU University Amsterdam,
De Boelaan 1105, 1081 HV Amsterdam, The Netherlands
antske.fokkens@vu.nl,
[2] Department of Linguistics, University Washington,
Box 354340, Seattle WA 98195-4340, USA
ebender@uw.edu

Abstract. In syntactic research, often more than one analysis can be found to account for available data. Interaction with analyses of other phenomena may help choose among them, but only phenomena analyzed in the past are taken into account. As a result, syntactic research is influenced by the order in which phenomena are treated. This paper presents a metagrammar-based methodology for maintaining alternative analyses over time. This way, we can use evidence from phenomena to be analyzed in the future to choose between analyses. We describe CLIMB, a methodology that implements this idea for HPSG grammars in TDL.

Keywords: Linguistic hypothesis testing, metagrammars

1 Introduction

Grammars, both those built by linguists and the natural objects they are intended to model, are complex objects (Bierwisch, 1963, Bender 2008a, Bender et al. 2011). A key benefit of grammar engineering as an approach to syntactic research is that it allows researchers to build and manipulate models of much greater complexity. In fact, it is extremely difficult to verify if all interactions between phenomena in a language model behave correctly without implementing them and checking them with a computer (Bierwisch, 1963, Müller 1999).

In any given grammar engineering project, analyses are implemented in some temporal order. But analyses of phenomena interact and the analytical choices already made influence the relative attractiveness of alternatives at each later choice point and thus the order in which phenomena are added affects the result (Fokkens, 2011). We argue (with Fokkens (2011)) that better grammars can result if grammar engineers can break free of the temporal sequence of implementation, and that metagrammar engineering is an effective way to do so.

Challenges related to deeply embedded analyses are familiar to most grammar engineers working on large scale grammars. Francis Bond (p.c.) reports that

[⋆] The authors thank Ned Letcher and the anonymous reviewers for their feedback which helped improve this paper. All remaining errors are our own.

it is often hard to identify parts of the grammar which relate to obsolete analyses in the Japanese grammar Jacy (Siegel and Bender, 2002). Montserrat Marimon (p.c.) reports that there are analyses in her Spanish grammar (Marimon, 2010) for clitics and word order that need revisions, but it would be an elaborate undertaking to make these changes, due to interactions with other phenomena, and they have been put on hold for now. Tracy King (p.c.) reports an ongoing discussion within ParGram (Butt et al. 2002) on whether adjectives have subjects or not. The English LFG grammar (Riezler et al. 2002) was changed a few times, but this was so time consuming that King decided to call the last change final.

This paper presents the workflow of a metagrammar engineering approach that helps avoid such problems. §2 discusses the theoretical question of what makes an analysis correct. In §3, we discuss the role a metagrammar can play when using grammar engineering for hypothesis testing. §4 presents the workflow of a specific approach for using this idea for building grammars in HPSG (Pollard and Sag, 1994). We compare our approach to related work in §6.

2 Correct analyses

We take it as uncontroversial that grammar engineers, whether working towards practical ends or building grammars to test linguistic hypotheses, strive to develop correct analyses. But what does it mean for a linguistic analysis to be correct? In order to answer that question, we first have to pin down our modeling domain. Many linguists (e.g. Chomsky (1986)) take the modeling domain to be internalized knowledge of language. That is, the rules and principles proposed in the grammatical model should stand in a transparent relation to the rules and principles that speakers encode in 'wet-ware'. However, we do not have direct access to speaker knowledge. Instead, our primary data include attested utterances (written and spoken) as well as speaker intuitions about the meanings of those utterances (entailments, paraphrase relations) and their acceptability.

The first test of a model of grammar, then, is that it should be able to account for the available data: Chomsky's (1965) 'descriptive adequacy' is a prerequisite for any kind of 'explanatory adequacy'. However, the available data typically vastly underdetermine the analysis (Wasow, 1978; Soames, 1984; Bender (2010): Even with the large data sets that implemented grammars can work with, and even fixing the general grammatical framework, there are always multiple ways to approach the analysis of any given phenomenon. Possible sources of information constraining the choice between alternative analyses include psycholinguistic evidence (Sag and Wasow, 2011) and cross-linguistic comparability. That is, if an analysis works for comparable phenomena across different languages, it can be considered a point in favor of the analysis, though the weight given to such evidence varies depending on the theoretical status of universals in a framework.

Here, however, we will focus on evidence which comes from the rest of the grammar. A correct analysis should not only work correctly in isolation, but also work in the larger context of the grammar, making correct predictions when it interacts with other analyses. This is part of descriptive adequacy and therefore

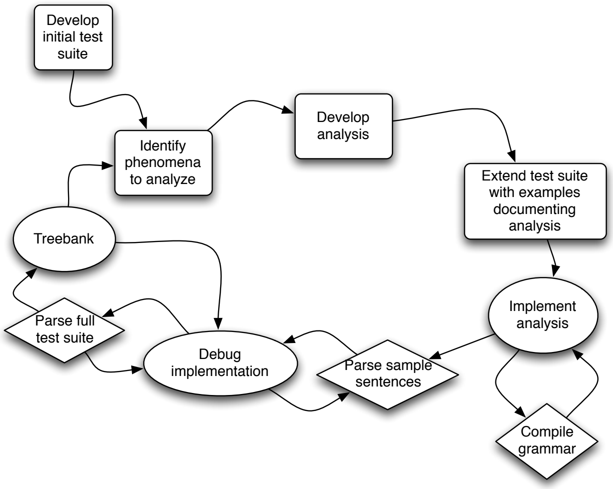


Fig. 1. Grammar engineering workflow, from 2011, p. 10

uncontroversial. Care should be taken however with its application, as we need to distinguish between *phenomena* and their *analyses*. If an analysis leads to incorrect predictions regarding the interaction with another phenomenon, then the analysis can be shown to be in error. However, if the incorrect predictions come from the interaction of the analyses of two phenomena, then it is not clear a priori which of the analyses is incorrect.

3 Metagrammar engineering for hypothesis testing

Because grammars are such complex objects, testing hypotheses about grammatical phenomena generally requires the development of complex models, with many interacting parts. Bender et al. (2011) illustrate how grammar engineering, together with regression testing, can help linguists manage the complexity of their models. This involves both testing the interactions between analyses and testing analyses against much larger collections of data than is feasible by hand. Bender et al. illustrate the process with the diagram in Fig. 1. Grammar engineering is a powerful tool for linguistic hypothesis testing because humans and computers are good at different things: Whereas humans are (still!) better suited to the task of developing linguistic analyses, computers are better at systematic checking and can verify the interaction of analyses on large amounts of data.

3.1 Multiple analyses

The process depicted in Fig. 1 is cyclic, with new phenomena being added on each successive pass. The testsuites document the analyses that have been implemented and allow the linguist to ensure that later work does not break what has gone before. This is key to the ability of grammar engineering to facilitate linguistic hypothesis testing. However, when we view the process of grammar engineering in this light, it also becomes apparent that phenomena considered earlier in the development of a grammar and their analyses have an asymmetrical influence on analyses of phenomena developed later (see also Bender (2008b)).

This asymmetrical influence is unfortunate: It is fairly common for a key phenomenon constraining the choice of analysis of another phenomenon to be only addressed after several further passes through the cycle. In the meantime, whichever analysis was chosen of the phenomenon implemented earlier may become deeply embedded in the growing grammar. This has several unfortunate consequences: First, over time, it is easy to forget what alternative analyses were available. Second, the longer an analysis has been part of a grammar, the more other analyses are likely to depend on it in some way. As noted in the introduction, this leads to scenarios where it becomes cumbersome or impractical to change an analysis, even when it is discovered to be suboptimal. Finally, even if a grammar engineer is inspired to revise a deeply-embedded analysis, it is simply not possible to explore all the paths-not-taken, that is, all the alternative analyses of the various interacting phenomena that might have been just slightly more desirable had the revised analysis been the one chosen in the first place.

In order to escape from this asymmetrical influence problem, what is required is the ability to explore multiple development paths. As described in the next section, this is exactly what metagrammar engineering provides.

3.2 A Many-Model interpretation

We will illustrate how a metagrammar may help improve this situation by drawing a parallel to the many-world interpretation of quantum mechanics (Everett 1957, DeWitt 1972). Quantum mechanics can predict the probability of a location of a photon and this prediction forms a wave function. However, as soon as the photon is observed, the probability function is reduced to one point and, according to the alternative Copenhagen Interpretation, the wavefunction collapses. The many-world interpretation rejects this idea and maintains that the alternative worlds in which the photon is at another location than the one observed are real, implying that alternative histories and futures are real. The analogue we imagine for grammar engineering is a many-model interpretation, where each model considers different analyses to be correct. Each implementation decision we make places us in a given model. While making a decision in grammar engineering, the grammar engineers sets off on a specific road and the route that is taken (the order in which phenomena are considered, the choices made concerning their implementations) influences the final destination (the resulting grammar). However, unlike real life, where we are stuck in a specific world and cannot explore alternatives, we can look at alternative models for grammars. The only problem is that it is not feasible to have a clear picture of all consequences of a specific decision while following the traditional grammar engineering approach. But what if we could monitor more than one model at the same time? What if, instead of making a decision to commit to a specific model,

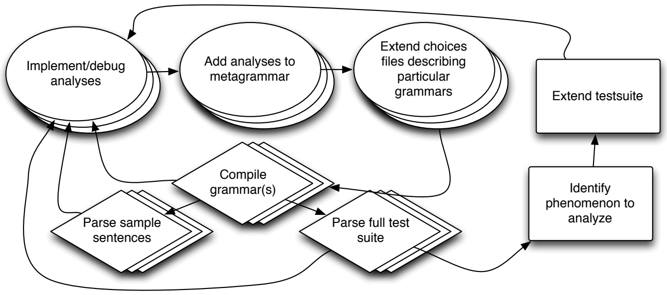

Implement/debug analyses
Add analyses to metagrammar
Extend choices files describing particular grammars
Extend testsuite
Compile grammar(s)
Identify phenomenon to analyze
Parse sample sentences
Parse full test suite

Fig. 2. Metagrammar engineering workflow

we follow a couple of models for a while, and test them with new analyses until we have gathered enough evidence to feel comfortable about a decision?

The methodology of metagrammar engineering can achieve this. The metagrammar can contain alternative analyses, including the alterations needed to make them interact correctly with analyses for other phenomena. Alternative models can be generated by selecting different options in the metagrammar. In §4, we describe a specific approach that applies this idea to HPSG grammars (Pollard and Sag, 1994) written in the TDL (Copestake, 2000).[3] It should be noted, however, that the idea of using a metagrammar and code generation to monitor alternative grammar models is both theory- and software-independent.

Fig. 2 schematizes the general workflow with a metagrammar approach. The stacked ovals represent manual processes that manipulate sets of alternative analyses, and the stacked diamonds automatic processes involving alternative grammars compiled out of those resources. The grammar engineer has the option of adding multiple alternative analyses for any phenomenon and, maintains a set of so-called ‘choices’ files specifying grammars built out of particular sets of choices among the analyses. The testing cycle involves compiling all of these grammars and parsing the testsuite with all of them. Using the many worlds metaphor, the resources encoding the alternative analyses and grammars are what define the space of models that the grammar engineer can monitor, the testsuite is the lens through which the monitoring is done, and the automated processing steps constitute the observations of each of the many ‘worlds’.

4 The CLIMB approach

This section describes the CLIMB[4] approach for metagrammar engineering. We first describe how CLIMB emerged from the Grammar Matrix (Bender et al. 2010), and then describe the three variations of CLIMB that currently exist: the traditional procedural CLIMB, declarative CLIMB and SHORT-CLIMB.

[3] TDL is the DELPH-IN (www.delph-in.net) joint reference formalism.

[4] Comparative Libraries of Implementations with a Matrix Basis; See moin.delph-in.net/ClimbTop for more project information and software download.

4.1 The Grammar Matrix and CLIMB

Climb (Fokkens 2011, Fokkens et al. 2012) builds on the LinGO Grammar Matrix (Bender et al. 2010), sharing with it the key idea of grammar customization. Grammar customization is the practice of generating grammar code from stored analyses on the basis of some input. There are, however, important differences, stemming from the goals of the projects: the Grammar Matrix aims to help (possibly novice) grammar engineers start new grammars, where the goal of climb is to support long-term grammar development for experienced grammar engineers.

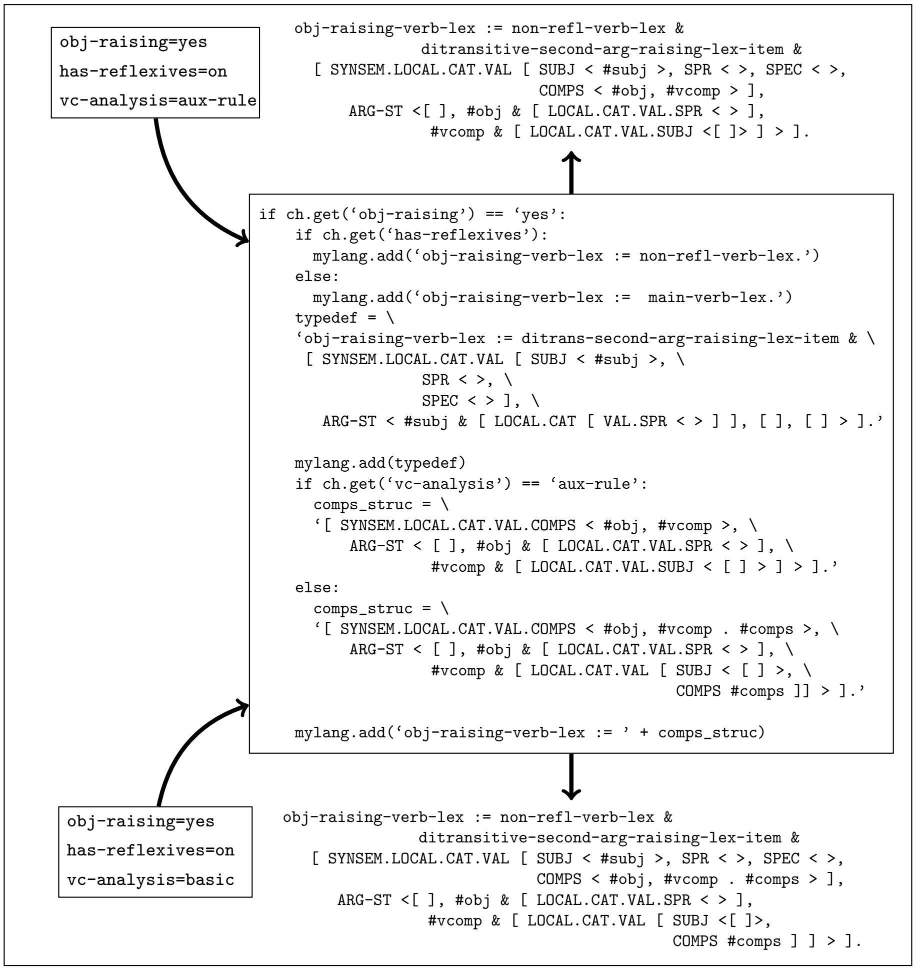
The Grammar Matrix applies grammar customization to the problem of rapid development of precision grammars. It stores a core grammar and libraries of analyses of cross-linguistically varying phenomena, which users can select in order to create a grammar fragment for (ideally) any human language. The project emphasizes typological coverage, and therefore can only add phenomena slowly, as each one must be grounded in thorough typological research. Climb generalizes the idea of grammar customization to metagrammar engineering, placing the customization source code under control of the grammar engineer, so that different levels of parameterization can be achieved in individual grammar development projects. Users are encouraged to explore the possibilities of the customization system and expand it for their language specific needs. This entails further differences between the projects: Matrix users access the system through a web-based questionnaire that hides the underlying hpsg analyses, while climb users are actively developing hpsg analyses (as implemented in tdl). Also, using grammar customization in the development of language-specific resources frees the grammar engineer to focus on phenomena as they manifest in the language(s) at hand, without concern for the full range of typological variation.

4.2 Procedural CLIMB

The original version of climb (procedural climb) builds directly on the Grammar Matrix, by simply taking the customization system (minus the web front-end), and allowing grammar engineers to extend it for particular languages. A climb metagrammar takes a choices file as its input and produces a grammar in tdl. The choices file specifies phenomena and properties that are generated using the metagrammar's libraries, which mainly consist of procedural functions that add type definitions to the grammar based on definitions in the choices file.

Fig. 3 gives a sample analysis as implemented in procedural climb. The metagrammar code contains control statements which check for certain properties in the choices file and statements which output tdl accordingly. The small boxes to the left of the figure are partial choices files, while the (declarative) tdl code at the top and bottom shows the system output according to those choices. This example illustrates the ways in which the analysis of object raising verbs depends on the presence or absence of reflexives in the grammar specification, on the one hand, and the chosen analysis of verbal clusters on the other.

Procedural climb can capture alternative analyses and thus achieves the goal of maintaining analyses in parallel. It has some additional advantages common



```
obj-raising=yes
has-reflexives=on
vc-analysis=aux-rule
```

```
obj-raising-verb-lex := non-refl-verb-lex &
             ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp > ],
      ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL.SUBJ <[ ]> ] > ].
```

```
if ch.get(‘obj-raising’) == ‘yes’:
    if ch.get(‘has-reflexives’):
      mylang.add(‘obj-raising-verb-lex := non-refl-verb-lex.’)
    else:
      mylang.add(‘obj-raising-verb-lex :=  main-verb-lex.’)
    typedef = \
    ‘obj-raising-verb-lex := ditrans-second-arg-raising-lex-item & \
     [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, \
                  SPR < >, \
                  SPEC < > ], \
       ARG-ST < #subj & [ LOCAL.CAT [ VAL.SPR < > ] ], [ ], [ ] > ].’

    mylang.add(typedef)
    if ch.get(‘vc-analysis’) == ‘aux-rule’:
      comps_struc = \
      ‘[ SYNSEM.LOCAL.CAT.VAL.COMPS < #obj, #vcomp >, \
          ARG-ST < [ ], #obj & [ LOCAL.CAT.VAL.SPR < > ], \
                   #vcomp & [ LOCAL.CAT.VAL.SUBJ < [ ] > ] > ].’
    else:
      comps_struc = \
      ‘[ SYNSEM.LOCAL.CAT.VAL.COMPS < #obj, #vcomp . #comps >, \
          ARG-ST < [ ], #obj & [ LOCAL.CAT.VAL.SPR < > ], \
                   #vcomp & [ LOCAL.CAT.VAL [ SUBJ < [ ] >, \
                                              COMPS #comps ]] > ].’

    mylang.add(‘obj-raising-verb-lex := ’ + comps_struc)
```

```
obj-raising=yes
has-reflexives=on
vc-analysis=basic
```

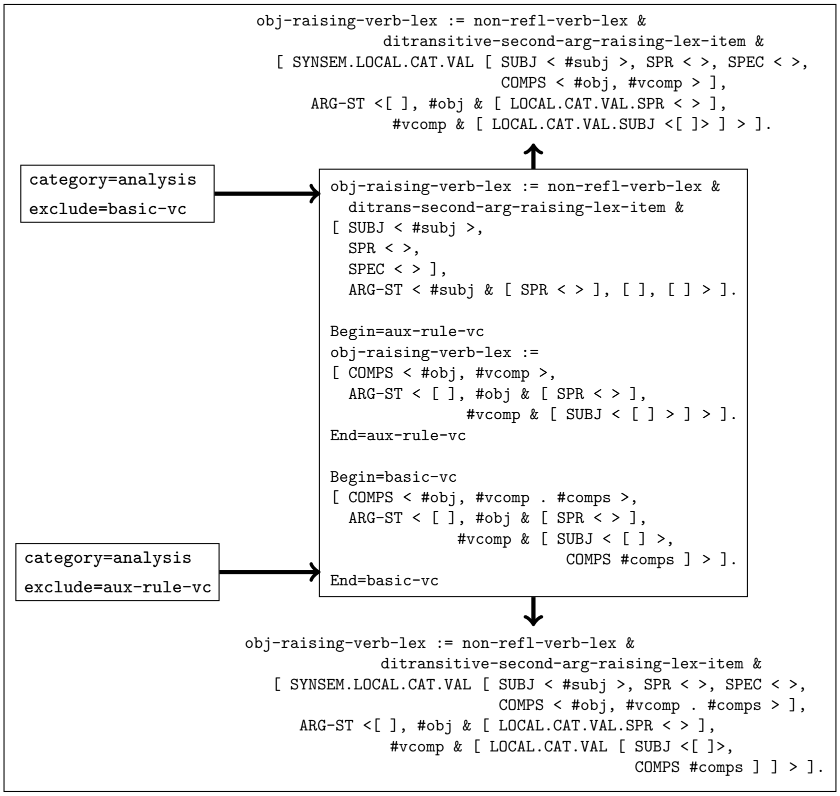
```
obj-raising-verb-lex := non-refl-verb-lex &
             ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp . #comps > ],
     ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
               #vcomp & [ LOCAL.CAT.VAL [ SUBJ <[ ]>,
                                          COMPS #comps ] ] > ].
```

Fig. 3. Snippet of procedural CLIMB code with alternative choices and their output

to metagrammar approaches, including increased modularity, more phenomena-oriented organization, facilitation of multilingual grammar development and application dependent variations in grammars. A draw-back of this approach, however, is that it requires implementing procedural functions in Python. Even for grammar engineers proficient in programming, it can be a nuisance to switch back and forth between declarative definitions of constraints in the grammar and procedural code invoking constraints based on choices in the metagrammar.

4.3 Declarative CLIMB

Declarative CLIMB addresses this drawback of procedural CLIMB. Fig. 4 shows the implementation of a basic type for object raising in declarative CLIMB. The definitions in declarative CLIMB are written directly in TDL, where paths in type definitions may optionally be abbreviated. A small set of additional declarative statements is used to identify where analysis-specific parts of an implementation start. The example includes alternative additions made to the basic type depending on the analysis chosen for verbal clusters (like in Fig. 3). The choices file indicates which analyses in the metagrammar should not be selected for the generated grammar. Procedural Python implementations are still used to interpret a choices file and merge type definitions from different locations in the metagrammar, but grammar engineers can treat these like a black box.



```
obj-raising-verb-lex := non-refl-verb-lex &
              ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp > ],
    ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
             #vcomp & [ LOCAL.CAT.VAL.SUBJ <[ ]> ] > ].
```

```
category=analysis
exclude=basic-vc
```

```
obj-raising-verb-lex := non-refl-verb-lex &
  ditrans-second-arg-raising-lex-item &
[ SUBJ < #subj >,
  SPR < >,
  SPEC < > ],
  ARG-ST < #subj & [ SPR < > ], [ ], [ ] > ].

Begin=aux-rule-vc
obj-raising-verb-lex :=
[ COMPS < #obj, #vcomp >,
  ARG-ST < [ ], #obj & [ SPR < > ],
               #vcomp & [ SUBJ < [ ] > ] > ].
End=aux-rule-vc

Begin=basic-vc
[ COMPS < #obj, #vcomp . #comps >,
  ARG-ST < [ ], #obj & [ SPR < > ],
              #vcomp & [ SUBJ < [ ] >,
                             COMPS #comps ] > ].
End=basic-vc
```

```
category=analysis
exclude=aux-rule-vc
```

```
obj-raising-verb-lex := non-refl-verb-lex &
              ditransitive-second-arg-raising-lex-item &
  [ SYNSEM.LOCAL.CAT.VAL [ SUBJ < #subj >, SPR < >, SPEC < >,
                           COMPS < #obj, #vcomp . #comps > ],
    ARG-ST <[ ], #obj & [ LOCAL.CAT.VAL.SPR < > ],
             #vcomp & [ LOCAL.CAT.VAL [ SUBJ <[ ]>,
                                        COMPS #comps ] ] > ].
```

Fig. 4. Snippet of declarative CLIMB code with alternative choices and their output

4.4 SHORT-CLIMB

Both procedural and declarative CLIMB assume that the grammar is built within a CLIMB approach from the ground up. But mature grammars can also benefit from the ability to explore alternative analyses (even if most of what has been implemented before adopting CLIMB is kept constant). It is not practical to take a very large grammar and re-implement it from the ground up just to get this benefit. Instead, SHORT-CLIMB[5] provides support for maintaining alter-

[5] Starting High On a Roof Top CLIMB

native analyses for grammars that have been developed the traditional way. An alternative analysis can be included by writing declarative statements on types and (partial) type definitions to be added to the grammar, types or constraints to be removed from the grammar and types that need to be modified.

The SHORT-CLIMB processing code takes a grammar written in TDL and a set of modifications as input and produces a revised grammar. It can optionally also provide a SHORT-CLIMB definition file to convert the revised grammar into the original grammar. Finally, it can produce a reduced grammar that only contains the common properties of both analyses, together with two files that can either create the original or the revised analysis using SHORT-CLIMB software. As such, SHORT-CLIMB can be used to convert a traditionally written grammar to declarative CLIMB step by step.

5 Scalability

CLIMB has been used to develop gCLIMB, a metagrammar for Germanic languages, developed with the goals of exploring the efficiency of alternative analyses and the applicability of the CLIMB methodology at the scale of resource grammars (Fokkens, in progress). The results in Fokkens (in progress) indicate that CLIMB scales well for longterm projects. gCLIMB has been developed for two and a half years and in spite of it having been a side project for most of that time, it already covers a wide range of phenomena for German. It contains alternative analyses for word order and auxiliaries, two phenomena that interact heavily with other phenomena in the grammar together resulting in five variations. In addition, the system provides for different approaches to the lexicon, including fully inflected or abstract forms of morphemes as well as a variation that integrates the lexicon and morphology of Cheetah (Cramer, 2011). In total, there are 25 different choices files specifying grammars representing different combinations of these analytical possibilities. As far as quantity is concerned, gCLIMB was built on top of a version of the Grammar Matrix that had 3,651 lines of code in the linguistic libraries. The metagrammar's libraries now consist of 13,032 lines of code, compared to 6,250 lines of code in the current version of the Grammar Matrix customization system. The choices files for these grammars have 1,970 lines of definitions (for small grammars used for testing) up to 3,657 lines (for creating the complete Cheetah core grammar and incorporating its lexicon). A customized grammar consists of approximately 7,300 lines of code.

6 Related Work

Metagrammars have been in use for over a decade, with notable long term efforts such as the MetaGrammar project (Candito 1998, Clément and Kinyon 2003) and GF (Ranta 2004, Ranta 2011). They have supported code sharing (Clément and Kinyon 2003, Ranta 2011), increased modularity, provided means for generalizations (Candito, 1998), combined expertise from linguists and engineers (Ranta, 2004) and facilitated multilingual grammar development (Ranta, 2011).

The two projects that most closely resemble declarative climb in their architecture are Sygal and Wintner's (2011) approach for modular development of typed unification grammars and the CoreGram Project (Müller, 2013). Like the work mentioned above, neither of these has addressed systematic comparison of alternative analyses, instead focusing on increasing modularity (Sygal and Wintner, 2011) and cross-linguistic grammar sharing (CoreGram). These difference in goals are reflected in differences in architecture.

Sygal and Wintner (2011) propose a structure that allows engineers to implement modules defining a part of a type hierarchy. These modules can be combined to form a complete grammar by specially defined functions. They point out the similarity between their approach and the Grammar Matrix customization system, where the main difference is that the grammar engineer does not have control over customization. This is exactly where climb differs from the Grammar Matrix. Where Sygal and Wintner improve modularity by developing a mathematically well-founded model, climb resulted from a practical approach to implementing alternative analyses, which requires increased modularity. This difference is also reflected in the verification of the approach. The former has been tested in a proof-of-concept implementation of the small hierarchy from in the appendix of Pollard and Sag (1994). Climb includes a grammar for German covering phenomena including subordinate clauses, complex predicates and extraposed comparatives. It seems promising that the theoretically well-founded approach and the practical broad-coverage approach are similar.

The CoreGram Project (Müller, 2013) is a bottom-up approach for cross-linguistic sharing of analyses between hpsg grammars running in TRALE (Meurers et al., 2002). Implementations that can be used for all grammars form a common core. When properties are observed that are shared by a subset of languages, they can be placed in subcores. Definitions belonging to the core, a subcore or language specific properties are defined in different files, and complete grammars are created by selecting the appropriate files. This approach partially provides the same functionality as declarative climb. In fact, the CoreGram approach can be used to monitor alternative analyses, but this is not (yet) a goal of the CoreGram project. Declarative climb differs from the CoreGram in providing a few additional features to facilitate the accommodation of alternative analyses. First, climb allows the metagrammar engineer to consolidate all constraints associated with one alternative in a single location, even if in the compiled grammar they affect many different types, allowing the grammar engineer two views: one based on analyses and one based on complete types. Declarative climb also supports the possibility of defining abbreviated paths which to a certain extent supports the exploration of alternative feature geometries. These differences can all be traced back to the different goals of CoreGram and climb, respectively.

To our knowledge, the only work that suggests that a metagrammar can be used to test alternative analyses is Fokkens (2011), but that work focuses on efficiency of implemented grammars. As such, we believe this work is the first to address the role of a metagrammar in hypothesis testing for syntactic research.

7 Conclusion

This paper outlined the role metagrammars can play in linguistic hypothesis testing. We described CLIMB: a metagrammar engineering approach that provides the means to create HPSG grammars written in TDL. Three versions of CLIMB have been presented: procedural CLIMB following Fokkens (2011), declarative CLIMB and SHORT-CLIMB. Declarative CLIMB allows grammar engineers to define their metagrammar libraries declaratively in TDL and has been developed to make the approach more accessible to grammar engineers who are not trained in procedural programming. SHORT-CLIMB can be used to create libraries that apply changes to large grammars that have been developed using the traditional (non-metagrammar) way. It thus provides the possibility of testing alternative analyses systematically in a large scale grammar.

The advantages of implementing grammars for hypothesis testing are well-known (Bierwisch 1963, Müller 1999, Bender 2008b). Metagrammar engineering adds an additional advantage in that allows the syntactician to systematically explore alternative analyses over time. In traditional grammar engineering, analyses may be so deeply embedded in the grammar it becomes too cumbersome to go back and change them. Metagrammar engineering makes it possible to monitor multiple models of a grammar. If we include two alternative analyses in a metagrammar and evidence for a particular analysis is found in the far future, we have the complete paths of both analyses. There is thus no need to dive into past decisions instead we can simply follow the path of the right choice as if we time traveled and corrected our past decision.

References

Bender, E.M.: Evaluating a crosslinguistic grammar resource: A case study of Wambaya. In: Proceedings of ACL-08: HLT, Columbus, Ohio (2008) 977–985

Bender, E.M.: Grammar engineering for linguistic hypothesis testing. In: Proceedings of the Texas Linguistics Society X Conference: Computational Linguistics for Less-Studied Languages, Stanford, CSLI Publications (2008) 16–36

Bender, E.M.: Reweaving a grammar for Wambaya: A case study in grammar engineering for linguistic hypothesis testing. LILT **3** (2010) 1–34

Bender, E.M., Drellishak, S., Fokkens, A., Poulson, L., Saleem, S.: Grammar customization. Research on Language & Computation **8**(1) (2010) 23–72

Bender, E.M., Flickinger, D., Oepen, S.: Grammar engineering and linguistic hypothesis testing: Computational support for complexity in syntactic analysis. In Bender, E., J.E., A., eds.: Language from a Cognitive Perspective: Grammar, Usage and Processing. Stanford: CSLI Publications. (2011) 5–29

Bierwisch, M.: Grammatik des deutschen Verbs. Volume II of Studia Grammatica. Akademie Verlag (1963)

Butt, M., Dyvik, H., King, T.H., Masuichi, H., Rohrer, C.: The parallel grammar project. In: Proceedings of the 2002 workshop on Grammar engineering and evaluation-Volume 15, Association for Computational Linguistics (2002) 1–7

Candito, M.: Building parallel LTAG for French and Italian. In: Proceedings of ACL and ICCL, Montreal, Canada (1998) 211–217

Chomsky, N.: Aspects of the Theory of Syntax. MIT Press, Cambridge, MA (1965)

Chomsky, N.: Knowledge of Language: Its Nature, Origin, and Use. Praeger, New York (1986)

Clément, L., Kinyon, A.: Generating parallel multilingual LFG-TAG grammars with a MetaGrammar. In: Proceedings of ACL, Sapporo, Japan (2003)

Copestake, A.: Appendix: Definitions of typed feature structures. Natural Language Engineering **6** (2000) 109–112

Cramer, B.: Improving the feasibility of precision-oriented HPSG parsing. PhD thesis, Saarland University (2011)

DeWitt, B.S.: The many-universes interpretation of quantum mechanics. In: Proceedings of the International School of Physics "Enrico Fermi" Course IL: Foundations of Quantum Mechanics. (1972)

Everett, H.: "Relative State" Formulation of Quantum Mechanics. Reviews of Modern Physics **29** (1957) 454–462

Fokkens, A.: Metagrammar engineering: Towards systematic exploration of implemented grammars. In: Proceedings of ACL, Portland, Oregon, USA (2011)

Fokkens, A., Avgustinova, T., Zhang, Y.: CLIMB grammars: three projects using metagrammar engineering. In: Proceedings of LREC 2012, Instanbul, Turkey (2012)

Fokkens, A.: Enhancing Empirical Research for Linguistically Motivated Precision Grammars. PhD thesis, Saarland University (in progress)

Marimon, M.: The Spanish resource grammar. In: Proceedings of LREC, European Language Resources Association (2010)

Meurers, W.D., Penn, G., Richter, F.: A web-based instructional platform for constraint-based grammar formalisms and parsing. In: Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics. (2002) 19–26

Müller, S.: Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche. Max Niemeyer Verlag, Tübingen (1999)

Müller, S.: The CoreGram project: Theoretical linguistics, theory development and verification. In: Proceedings of the workshop on High-level Methodologies for Grammar Engineering. (2013)

Pollard, C., Sag, I.: Head-Driven Phrase Structure Grammar. University of Chicago Press, Chicago, USA (1994)

Ranta, A.: Grammatical Framework: A type-theoretical grammar formalism. Journal of Functional Programming **14**(2) (2004) 145–189

Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford, USA (2011)

Riezler, S., King, T.H., Kaplan, R.M., Crouch, R., III, J.T.M., Johnson, M.: Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In: Proceedings of ACL. (2002)

Sag, I.A., Wasow, T.: Performance-compatible competence grammar. In Borsley, R., Borjars, K., eds.: Non-Transformational Syntax: Formal and Explicit Models of Grammar. Wiley-Blackwell (2011)

Siegel, M., Bender, E.M.: Efficient deep processing of Japanese. In: Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization, Taipei, Taiwan (2002)

Soames, S.: Linguistics and psychology. Linguistics and Philosophy **7** (1984) 155–179

Sygal, Y., Wintner, S.: Towards modular development of typed unification grammars. Computational Linguistics **37**(1) (2011) 29–74

Wasow, T.: On constraining the class of transformational languages. Synthese **39** (1978)

A Toolkit for Engineering Type-Logical Grammars

Richard Moot

LaBRI (CNRS), Bordeaux University
Richard.Moot@labri.fr

This demo shows a multi-purpose toolkit for annotating and parsing type-logical grammars. The current implementation and demo focus on the analysis of French but the tools are fully general and can be adapted to other languages. Figure 1 gives an overview of the different tools and resources.

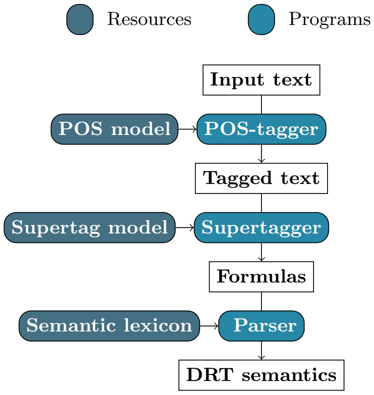


Fig. 1. An overview of the toolkit

The POS-tag model and the supertag model have been trained using the tools of [1]. They provide first part-of-speech tags then a set of contextually likely formulas (*supertags*) for each of the words in an input sentence. These formulas are the input to the chart parser, a stripped-down and optimized version of the Grail parser for type-logical grammars [2], which produces a representation of the meaning of the sentence but also LaTeX ouput of the parse/proof.

In addition to these programs, a number of interface tools is provided to simplify user interaction. For example, an annotation tool helps bootstrap unannotated parts of the corpus. It uses the POS-tagger and supertagger to give an annotation for a sentence and allows the user to interactively correct this sentence: either by selecting an alternative proposed by the supertagger or by manually typing the correct formula. To help prevent errors, the number of times this word-POS tag combination occurs with this formula is listed and the full annotation can be passed directly to the parser.

All tools and resources are released under the GNU Lesser General Public License.

References

1. Clark, S., Curran, J.R.: Parsing the WSJ using CCG and log-linear models. In: Proceedings of the 42nd annual meeting of the Association for Computational Linguistics (ACL-2004). pp. 104–111. Barcelona, Spain (2004)
2. Moot, R.: Wide-coverage French syntax and semantics using Grail. In: Proceedings of Traitement Automatique des Langues Naturelles (TALN). Montreal (2010)

eXtensible MetaGrammar: a modular tool for grammar engineering

Simon Petitjean

LIFO, University of Orleans

1 Introduction

Designing and maintaining a large scale formal grammar is a long and complex task, because it means manipulating and checking thousands of structures. XMG [1] is a compiler inspired by the concept of MetaGrammar [2] created to address this issue. The idea behind XMG is to generate the grammar by describing fragments of the grammar rules, and combining them to produce the whole set of rules.
Linguists can work with different languages, get interested in different representation levels of these languages, and use different formalisms to represent the linguistic information. Dimensions are a concept that XMG uses to separate these levels of description, and provide adapted languages to describe the structures that they imply. Linguistic theories are represented into the compiler by the means of principles. Principles are set of constraints that can be applied to dimensions. For example, the unicity principle ensures that a property is unique inside a model.
A high level of modularity is necessary for the compiler to be easily adapted for new linguistic theories, coming with their own dimensions and principles. For this reason, the compiler can be built dynamically by choosing compiler bricks and combining them.

2 Compiling a MetaGrammar

Generating a grammar with XMG is going through a set of compiling steps, summarized in figure 1. The first ones take a MetaGrammar file, compile it and execute the generated code. This leads to a set of accumulations, corresponding to dimensions. Then, for each dimension, the last dedicated compiling steps extract all models from the descriptions, producing the grammar.

3 Compiler Bricks

Every part of the compiler that defines a dedicated language and associates compiling steps to this language is called a Compiler Brick. These bricks can correspond to dimensions, but also to finest reusable parts of the compiler. For example, to declare and manipulate Attribute Value Matrices inside a dimension, one just needs to instantiate the AVM brick.

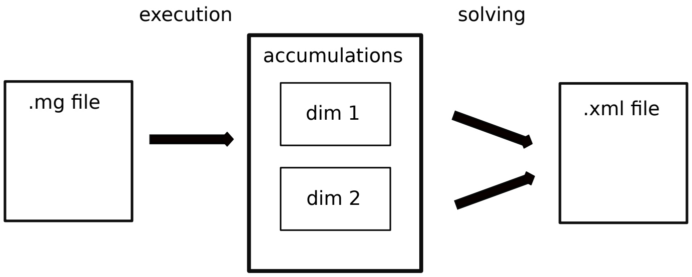


Fig. 1. A basic view of the compiler

4 Extensions

Originally designed to deal with two tree based formalisms for syntax, Tree Adjoining Grammars [3] and Interaction Grammars [4], and a predicate based formalism for semantics ("flat" semantics), XMG has lately been extended with new dimensions. Another level of linguistic description has been explored: the morphological one. Two different bricks, based on different linguistic theories, were created. One of them has been used in [5]. A new semantic dimension is also available, handling frame based semantics. It has been used together with the second morphological brick in [6].

References

1. Petitjean, S.: Spotting and improving modularity in large scale grammar development. In: Proceedings of the Student Workshop of the 18th European Summer School in Language, Logic and Information (ESSLLI 2012), Opole, Poland (2012)
2. Candito, M.: A Principle-Based Hierarchical Representation of LTAGs. In: Proceedings of COLING 96, Copenhagen, Denmark (1996)
3. Joshi, A.K., Schabes, Y.: Tree adjoining grammars. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Springer Verlag, Berlin (1997)
4. Perrier, G.: Interaction Grammars. In: Proceedings of COLING 2000, Saarbrücken, Germany (2000)
5. Duchier, D., Magnana Ekoukou, B., Parmentier, Y., Petitjean, S., Schang, E.: Describing Morphologically-rich Languages using Metagrammars: a Look at Verbs in Ikota. In: Workshop on "Language technology for normalisation of less-resourced languages", 8th SALTMIL Workshop on Minority Languages and the 4th workshop on African Language Technology, Istanbul, Turkey (2012)
6. Lichte, T., Diez, A., Petitjean, S.: Coupling trees, words and frames through XMG. In: Proceedings of the Workshop on High-level Methodologies for Grammar Engineering (HMGE 2013), Düsseldorf, Germany (to appear)

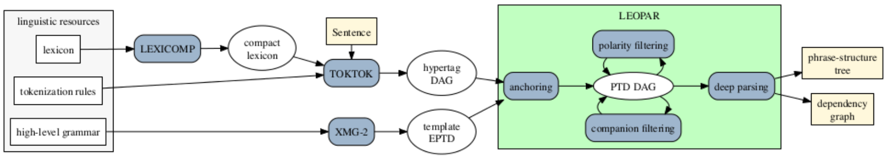
Leopar: an Interaction Grammar Parser

Guy Perrier and Bruno Guillaume

LORIA, Université de Lorraine, Nancy, France

1 Overview

Leopar[1] is a parser for natural languages which is based on the formalism of Interaction Grammars. The parsing principle (called "electrostatic parsing") consists in neutralizing opposite polarities: a positive polarity corresponds to an available linguistic feature and a negative one to an expected feature. The structures used in IG are underspecified syntactic trees decorated with polarities and called *Polarized Tree Descriptions* (PTDs). During the parsing process, PTDs are combined by partial superposition guided by the aim of neutralizing polarities: two opposite polarities are neutralized by merging their support nodes. Parsing succeeds if the process ends with a minimal and neutral tree. The figure below describes Leopar and the toolchain around it (EPTD stands for *Elementary Polarized Tree Descriptions*).



2 Main modules

Anchoring: each lexicon entry is described with an hypertag (i.e. a feature structure which describes morpho-syntactic information on the lexical unit); in order to preserve tokenization ambiguities, tokenization is represented as a Direct Acyclic Graph (DAG) of hypertags; for each hypertag describing a lexical unit, the relevant EPTDs are build by instanciation of template EPTDs defined in the grammar.

Filtering: paths in the DAG produced by the anchoring represented the set of lexical selections that should be considered by the parsing process. In order to reduce the number of paths and so to speed up the deep parsing, the next step in Leopar is to filter out paths that are doomed to fail. Two kinds of filters are used: polarity filters remove lexical selections for which the set of polarities in not well-balanced and companion filters remove lexical selections

[1] `http://leopar.loria.fr`

for which it can be predicted (from knowledge on the template EPTDs) that some polarity will fail to find a dual polarity (called a companion) able to saturate it.

Deep parsing: the atomic operation used during deep parsing is the node merging operation. At each step, two dual polarities are chosen; the two nodes carrying these polarities are merged and tree description around the two are superposed. Of course, in case of dead-lock, backtracking is used to chose another pair of polarities.

Phrase-structure trees and dependency graphs: (E)PTD are tree descriptions which describes constraints on the phrase-structure tree. The parsing process aims to build a phrase structure tree which is a model the EPTDs chosen for each lexical unit. Dependency graphs are build from the phrase structure tree but also with information about taken from the parsing process itself.

Ygg, parsing French text using AB grammars

Noémie-Fleur Sandillon-Rezer

LaBRI, CNRS   351 cours de la Libération, 33405 Talence

This demo focuses on the use of Ygg, our sentence analyzer which uses a probabilistic version of the CYK algorithm [Yo1]. An overview of our various softwares is explained by the scheme 1.

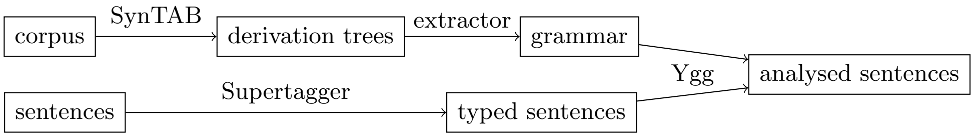


Fig. 1. Processing line.

In order to run our analyzer, we need an input grammar and sentences to analyze. The raw sentences come from various corpora, and will be typed by the Supertagger [Mo2]. The AB grammar [La1] is extracted from the French Treebank, with SynTAB, described in detail in [SM1]: 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.

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 $a \to b\ b\backslash a$).

Finally, we use the CYK to create derivations trees and λ-terms corresponding to them.

References

[La1] Lambek, J.: The mathematics of sentence structure. The American Mathematical Monthly 65 (1958)

[Mo2] Moot, R.: Semi-automated extraction of a wide-coverage type-logical grammar for french. Proceedings TALN 2010, Monreal (2010)

[SM1] Sandillon-Rezer, NF., Moot, R.: Using tree tranducers for grammatical inference. Proceedings LACL 2011, Montpellier (2011)

[Yo1] Younger D.: Context free grammar processing in n^3 (1967)

Author Index