Chapter 7. **K. Gerdes, S. Kahane, R. Bawden, J. Beliao, E. de la Clergerie, I. Wang**

Annotation Tools for Syntax

Abstract. This chapter is devoted to the presentation of the tools and methods used for the different steps of the semi-automatic syntactic annotation: automatic preprocessing; microsyntactic parsing with the FRMG tool, correction of the parsing with the Arborator tool, agreement analysis, post-validation correction, and development of the final format of the Rhapsodie syntactic treebank. As FRMG is a parser for written French that was not configured to analyze disfluencies and reformulation, we used our manual pile marking to unfold the piles and produce a series of simplified "sentences" with only government relations. Despite having two annotators plus a validator for the corrections, we found a substantial number of errors in the post-validation procedure by using a set of rules to determine the well-formedness of the trees.

1. Introduction

Our annotation strategy relies on the fact that relatively good tools for the automatic analysis of French written texts were in existence at the start of the Rhapsodie project (Bourigault *et al.* 2005; de la Clergerie 2005a; Boullier & Sagot 2005). Adapting these tools to spoken French would have constituted a project in itself, and a much more ambitious one than our annotation project. We believe however, and this is one of the explicitly stated aims of the project, that Rhapsodie is an essential step towards the development of spoken language resources which can contribute to the development and the training of specialized parsers.

Our approach was to use existing tools developed for written texts without substantially modifying the tools themselves. This implies a “manual” pretreatment of the transcribed texts: The segmentation into Government Units (GU) and Illocutionary Units (IU) described in Chapters 4 and 6 aims at providing such a pretreatment as it describes syntactic phenomena that are typical of spoken texts. We showed in Chapter 3 that our pretreatment is of theoretical and practical use and could constitute a satisfying syntactic analysis of speech in its own right. In this section, we nevertheless present all the steps of our annotation process, as this will allow for a greater understanding of the choices that were made (for example the analysis of lists during the pretreatment phase).

Our annotation procedure comprises several steps which alternate between automatic and manual treatment.

Input: Raw transcription (i.e. without syntactic enrichment). This consists of an orthographic transcription including speech overlaps, truncated morphemes, etc. (see Chapter 2).

Level 1: Simple automatic pretreatment. Annotation of trivial disfluencies (such as word repetition) and identification of potential associated nuclei (*um*, *uh*... but also *like*, *you know*...). This automatic step is very rough and needs to be corrected at level 2.

Level 2: Manual macrosyntactic and paradigmatic annotation. The general idea is that the markup simultaneously constitutes:

- a coding of specific structures that we expect not to be able to compute automatically, and which would cause difficulties for parsers (originally programmed for written text) in particular lists, adnuclei, and parentheticals;
- a structure which is satisfactory in itself and permits a preliminary study of the

syntax-prosody interface.

A tool, called Pilepilot, was developed to check the well-formedness of this level of annotation.

Level 3: Computing the parser input. The markup of level 2 is used to produce a series of simplified “sentences” that we expect the parser to handle well. This is not just a segmentation of the transcription but a set of multiple sentences in which lists have been unfolded, each sentence containing just one layer.

Level 4: Parsing. The FRMG parser (French MetaGrammar parser, de la Clergerie 2005a) is used to obtain a POS tagging and a dependency analysis for each unfolded segment.

Level 5: Creation of the microsyntactic analysis of the original transcription. This step includes the conversion of the dependency labels and POS tags into Rhapsodie's norms, fusion of the parse structures from the unfolded segments, and inclusion of paradigmatic and inherited links stemming from level 2 markup.

Level 6: Microsyntactic annotation. A manual correction of level 5 using the Arborator, a collaborative online tool for the annotation of dependency and POS (Gerdes 2013). This level is particularly demanding and the final size of the corpus is therefore mainly determined by human time-consumption at this step. In order to ensure a high quality of annotation, we proceed by independent double annotation and validation.

Level 7: Rule-based post-validation correction. We implement rules to determine the well-formedness of the trees and search for structures that do not obey them.

Level 8: Production of the combined syntactic annotation. The macrosyntactic annotation of level 2 is combined with the final microsyntactic annotation for outputs in various formats.

In the following sections, we will justify this processing chain, explain the main steps, and present the major tools we used.

2. Parsers for Written and Spoken French

2.1. Parsers for French

In the French parsing evaluation campaigns EASy and Passage (de la Clergerie *et al.*, 2008), more than 10 mostly rule-based systems competed in (non-recursive) chunking and dependency parsing on several document styles (journalistic, literary, questions, medical, e-mail) including a small corpus of speech transcriptions from the DELIC corpus (519 sentences).

More recently, since the end of the Passage project, we have seen the emergence of statistical parsers for French, with some improvements in performance. However, most of these systems are designed for parsing written French and are trained on very specific journalistic texts (with the notable exception of domain adaptation to the heterogeneous SEQUOIA Treebank, Candito & Seddah 2012). Very few parsing systems are concerned with spoken French. Since the completion of the parsing stage for Rhapsodie, one other system has also been developed: the Jsafran platform (Cerisara & Gardent 2011) for semi-automatic speech processing, with the adaptation of the MALT dependency parser on spoken corpora (also referred to as JSYNATS). The F-scores are inferior to results on written dependency treebanks, no doubt due to the smaller training corpus and the supposedly greater syntactic complexity of oral corpora.

FRMG, the parser used for Rhapsodie, was trained and successively improved on the evaluation corpus of the EASy and Passage campaigns, reaching an F-measure of 90% for chunks and 69.23% for dependencies in the latest evaluation (September 2013). The measures on the transcription subcorpus are lower, although not dramatically so, ranging from 58.79% to 75.77%.

2.2. The difficulty of parsing spoken language

Although a parser designed for written texts can process speech transcriptions relatively successfully, provided the appropriate pre- and post-processing is done, as was the case for Rhapsodie, there are several good reasons for the design or the tuning of a specific parser for speech.

One of these reasons concerns sentence segmentation. Most parsers, including FRMG, are designed to process a single well-delimited sentence at a time. However, it may be difficult to segment a speech stream into sentences when relying solely on prosody (see Chapters 6 and 18) (in fact the task is sometimes far from straightforward even for written documents). A better approach would of course be the use of cues from both syntax and prosody to segment the flow of speech into words. The complexity of the segmentation and historical reasons may explain why almost no parsing system handles such streams. Historically, language processing on written texts is organized as pipelines of components, with for instance a first component in charge of word- and sentence-segmentation, followed by a tagger, and then a parser. This modularity helps in the design of each component. Traditional approaches based on chart parsing for more or less complex syntactic formalisms involve time and space worst-case complexities that very quickly increase with respect to sentence length *n*, with for instance complexities in *O*(*n*3) for Context-Free Grammars (CFG) and *O*(*n*6) for Tree Adjoining Grammars (TAG). Even if these complexities are rarely met for real grammars and real sentences, it is costly to consider long sentences. Furthermore, some parsing strategies (such as Cocke-Kasami-Younger, CKY) do not satisfy the valid-prefix property (Nederhof, 1999) which ensures that words may be consumed incrementally from left to right, building partial parse structures (that can be completed given some potentially new suffix string). And, finally, to further reduce complexity also with respect to grammar size, many systems make use of the list of words in the sentence prior to parsing in order to filter out large parts of the grammar. Similar considerations hold for statistical approaches.

A second reason for the adaptation of parsers for speech transcription is the presence of a set of specific phenomena mostly found in speech, in particular those related to breaks in the linear structure of the sentence and which motivated the multi-layer scenario used in Rhapsodie. However, most macrosyntactic and paradigmatic structures that are typical of spoken language have comparable constructions in written language. For instance, sentence segmentation can be difficult in cases of embedded citations or itemized lists, and elliptic or multiple coordination is a constant source of parsing error. Written parsers could therefore also greatly benefit from improvements in (possibly incremental) speech parsing techniques.

3. Segmentation and choice of a formalism

The lack of availability of tools for the segmentation of transcriptions of spoken French obliged us to carry out the segmentation manually. As shown in Chapters 3, 4, and 6, a text can be segmented in accordance with different principles and we used two different segmentations: an illocution-based or macrosyntactic segmentation into Illocutionary Units (IUs), and a government-based or microsyntactic segmentation into Government Units (GUs).

These structures can be represented by phrase structure or dependency trees. For our manual annotation of the macrosyntax, we chose a third method: a markup with segmenters, which is a degenerated form of parenthesizing, thus simplifying the manual on-text annotation procedure. We decided to indicate the boundary between macrosyntactic constituents with the following tags:

- // for the boundary between two IUs;
- < for the right boundary of a prenucleus;
- > for the left boundary of a postnucleus.

For instance, example (1) contains the macrosyntactic annotation.

(1) *alors < là < la psychiatrie < c'est autre chose //* [Rhap-D0006, CFPP2000]

so < there < the psychiatry < that's another thing //

'in this case, psychiatry is a different thing'

This simplification is unambiguous under the hypothesis that an IU has exactly one nucleus, prenuclei only to its left, and postnuclei only to its right.1 We have several reasons for our choices:

1. The macrosyntactic structure is very rarely recursive. Embedded IUs exist, but are rather rare, thus allowing for the encoding of most macrosyntactic structures with just these 3 segmenters. Any embedded IU has to be parenthesized explicitly (using square brackets, see example (2) and Figure 1).
2. Due to the non-recursivity property, it is the most economic encoding we can imagine. For (2), with only 4 tags we can encode a constituent structure with 5 constituents (Figure 1), whereas a complete parenthesizing would need 10 brackets.
3. From a practical point of view, this marking does not need any pretreatment. Tags can be added directly to the orthographic transcription, that is, directly into the text.
4. Such an encoding is very intuitive because it is not very different from the punctuation used in writing: < and > correspond to commas, while // corresponds to full stops. The main difference is that our "punctuation" is formally defined and can be selected unambiguously, contrary to classical punctuation whose expressiveness can vary depending on microsyntactic, macrosyntactic, or prosodic criteria (Benzitoun *et al.* 2010; Deulofeu 2011; Pietrandrea *et al*. 2014).

(1) (2) *ce qui est horrible < c'est de se dire [je n'en sortirai jamais //] //* [Rhap-D2001, Mertens]

'what is horrible < it's to say to yourself [I'll never pull through //] //'

Figure 1. A recursive macrosyntactic constituent tree for (2)

We also decided to simultaneously indicate the segmentation of the text into government units (GUs), because in most cases the boundary of an IU, or even of an IC, is also a boundary of a GU. Consequently, we consider that a macrosyntactic tag corresponds by default to a GU boundary; and when it does not, we combine the tag with the + sign to indicate that the GU continues beyond it. In (3), the prenucleus *ensuite* ‘afterwards’ forms a GU with the nucleus *vous avez été ruinée* ‘you were ruined’; this GU is interrupted by the parenthetic IU *disons les mots comme ils sont* ‘let's say the words as they are’.

(3) *ensuite <+ "euh" (disons les mots comme ils sont //) vous avez été ruinée //* [Rhap-D2001, Mertens]
‘afterwards <+ "uh" (let’s say the words as they are //) you were ruined //’

4. Manual annotation with Pilepilot

The annotation of the transcription faces a triple challenge:

- Analyzing in greater detail the syntactic structure of any transcription of an oral text reveals a great number of transcription errors, even for high quality transcriptions;
- This correction process, like the macrosyntactic analysis itself, requires direct access to the sound recordings for verification;
- Although the macrosyntactic annotation has been simplified as much as possible (see above), manual annotation is always an error-prone activity.

Figure 2. A screenshot of the Pilepilot macrosyntactic annotation tool

The Pilepilot addresses these points directly: It is an online tool, accessible from within any standard-compliant browser, and provides the ability to upload the sound file together with the temporal alignment (in Praat's Textgrid format) to the server. The transcription can be pasted into a text field (bottom of Figure 2), where it can be directly edited and enriched with the markup. A double click on any word allows the annotator to play the corresponding sound file as many times as needed, starting with the selected word. Even after correction of the transcription, the approximate temporal position of a selected word is found. A sound player with a slider (top of Figure 2) is included in the window and also allows manual movement inside the sound file.

Moreover, as soon as the text field loses focus (for example if the annotator clicks anywhere else in the browser window), the syntax of the annotation is verified. Correct closures of brackets are checked as well as correct positions of the other segmentation characters. Pilepilot is written in Python and Javascript. The currently used version of Pilepilot does not include online storage or versioning of the segmented transcriptions. For this we used a simple Mediawiki server with one page per sample into which the user has to copy and paste the transcription with its markup.

5. Unfolding-Refolding

Existing parsers for French have not been developed to process transcriptions of speech, as explained in 2.1, nor have they been tuned to treat the markup that we have introduced at level 2 (Section 3). However, these tags allow us to automatically segment the text and to provide the parser with sections it is capable of analyzing because they correspond in general to simpler and more standardized text segments. The analysis on these text segments is then used to compute a syntactic analysis of the actual text.

We first cut the annotated text into GUs (see Chapter 4) and then unfold the lists by listing all possible paths resulting from choosing one layer of each list (Gerdes & Kahane 2009, see also Chapter 5). Example (4) illustrates this point.

(4) *{ le point de vue du spectateur | le point de vue esthétique } se dégage "euh" { à la fin du dix-septième siècle | au dix-huitième siècle } //* [Rhap-M2002, Rhapsodie]

‘{ the viewpoint of the audience | the esthetic viewpoint } emerges "uh" { at the end of the 17th century | in the 18th century} //’

Figure 3. Schematic grid analysis of example (4)

As indicated by the lines in Figure 3, the two lists in (4) give us four paths presented in (5) (plus the segment *euh*).

(5) **a.** *le point de vue du spectateur se dégage à la fin du dix-septième siècle*

b. *le point de vue du spectateur se dégage au dix-huitième siècle*

c. *le point de vue esthétique se dégage à la fin du dix-septième siècle*

d. *le point de vue esthétique se dégage au dix-huitième siècle*

Some fragments of text are therefore duplicated to be parsed multiple times. The syntactic analyses of these fragments, if identical, are recombined later. The unfolded segments are analyzed by the FRMG parser and we thus obtain multiple analyses that have to be recombined to form the complete parse tree (Belião 2012). For example, the layers of the subject list of example (4) appear in different parsed segments (*le point de vue du spectateur* in (5a) and (5b) and *le point de vue esthétique* in (5c) and (5d)), which have to be fused in the following way: the verb *dégage* 'emerges' that appears in all unfolded segments has to be fused, the first layer (*le point de vue du spectateur*) has to retain the *subject* relation, the second layer (*le point de vue esthétique*) has to be considered as an inherited subject, and between the two layers we have to construct a paradigmatic relation. The same holds for all other segments that have been parsed multiple times.

Figure 4. Resulting dependency structure of example (4)

A more complex example, shown in (6), including embedded and discontinuous lists, unfolds to (7).2

(6) *L1 il faut compter "euh" "pff" { { l'équivalent de quarante euros | quelque chose comme ça } |} //+ (L2 "ah" "oui" "ah" je pensais plus que ça //) L1 "oui" {| { quarante | quarante-cinq } } //* [Rhap-D0009, PFC]

'L1 it takes "uh" "pff" { { the equivalent of forty euros | something like that } |} //+ (L2 "ah" "yes" "ah" I thought more than that //) L1 "yes" {| { forty | forty-five } } //'

(7) **a.** *euh* 'uh'

b. *pff* 'pff'

c. *ah* 'ah'

d. *oui* 'yes'

e. *ah* 'ah'

f. *oui* 'yes'

g. *je pensais plus que ça* 'I thought more than that'

h. *il faut compter l'équivalent de quarante euros* 'it takes the equivalent of forty euros'

i. *il faut compter quelque chose comme ça* 'it takes something like that'

j. *il faut compter quarante* 'it takes forty'

k. *il faut compter quarante-cinq* 'it takes forty-five'

The macrosyntactic structure of (6) is composed of three simple IUs, and some segments in (7) stem from different IUs: for example (7k) is composed of parts stemming from the first IU (*il faut compter* 'it takes') and the third IU (*quarante-cinq* 'forty-five').

Another complication arises when the parser returns different analyses for the same segment. Moreover, some unfolded segments may be ungrammatical. This can appear in cases such as (8-9) where the speaker corrects herself with an article of a different gender.

(8) *le livreur s'éloigne pour voler { un | une } baguette* [Rhap-M0024, Rhapsodie]

'the delivery man moves away in order to steal {a *(masc.)* | a *(fem.)*} baguette'

(9) **a.** *le livreur s'éloigne pour voler **un baguette**

b. le livreur s'éloigne pour voler **une baguette**

In these cases, the fusion into a unique structure forces us to make a choice between the different analyses. We applied the heuristics that the last layer is the correct one (often equivalent to the corrected layer), except in cases of layers that are marked as being unsaturated.

6. Parsing with FRMG

FRMG is a freely available wide-coverage French grammar, used to prepare syntactic annotations for Rhapsodie (through online access via a Web service), even if it is primarily designed for handling written texts. FRMG is part of a long tradition of parsing and of development of linguistically oriented grammars, including chart parsing, tree adjoining grammars, and meta-grammars.

Tree Adjoining Grammars (TAGs, Joshi *et al.* 1975) use grammars formed of elementary trees that may be combined by the operations of substitution and adjunction to build full parse trees. The elementary trees provide an extended domain of locality adequate for capturing long distance interaction between various components, for instance between a verb and its arguments, even when they are extracted. However, this richness leads to a high number of trees for any realistic large coverage grammar, practically a few thousand trees and easily more than ten thousand trees. Progressively designing and maintaining so many trees becomes an almost impossible task. An option is to generate the grammar from a higher-level description, as explored with meta-grammars (Candito 1996).

The first level of FRMG is therefore a meta-grammar (de la Clergerie 2005a) that relies on a hierarchy of classes with multiple inheritance. A class contains elementary constraints on the grammatical structures (for instance, stating that a determiner node should precede the noun node in a nominal clause, or that there exists an inflection node dominating the verbal node in a verbal clause). A class will inherit all the constraints provided by its parent classes and may be combined to other classes through a mechanism of resource provider/consumer in order to generate minimal TAG trees.

The second level of FRMG is the TAG grammar generated by its meta-grammar. A particularity of the TAG level is the generation of *factorized trees*, through the use of regular operators such as the disjunction of subtrees within a tree, the free ordering of sibling subtrees, the optionality of a subtree, or the repetitiveness of a subtree. By using these factorization operators (automatically deduced from the indications of the meta-grammar), it is possible to get a very compact grammar of 359 elementary trees, with all verbal constructions covered by only 43 trees anchored by verbs. Some of these verbal trees are actually quite complex and could not really have been written by hand.

The third level of FRMG concerns the compilation of the grammar into a chart-like parser running within the tabular logic-programming environment DyALog (de la Clergerie 2005b). The resulting parser takes into account a word lattice (equivalent to a Directed Acyclic Graph (DAG)) produced by SxPipe (Sagot & Boullier 2005), a word and sentence segmenter with recognition of Named Entities (NEs). By default, the parser uses a top-down parsing strategy, advancing from left to right, and preserving the valid-prefix property.

The parser first tries to return the forest of full parses, that is, parses covering the whole sentence. When no full parse can be found, the parser switches to a correction mode where potential points of failure are identified, leading to the addition of a few extra edges in the lattice, similar to the existing ones but with fewer instantiated morphosyntactic feature values. The scheme allows the simulation of a relaxation of the grammatical constraints, for instance to relax the agreement constraints between a determiner and its nominal head, or between a subject and its verbal head. When the correction mode fails, the parser finally switches to a robust mode where it returns sequences of maximal partial parse trees.

The shared derivation forest built by FRMG is converted into a dependency forest, using the fact that most trees are lexically anchored and using the anchor lexeme as the source or target of the dependencies. For non-anchored trees, a pseudo head is introduced (with no lexical content, see the nodes *Vmod, end,* and *S* in Figure 5 as an example). The dependency trees of the shared forest can be ranked using a set of hand-crafted disambiguation rules applied on the dependency edges. In more recent versions of the parser, automatically extracted weights are assigned to the edges based on partially supervised learning, giving state of the art results compared to purely statistical parsers (de la Clergerie 2013).

The modularity of FRMG facilitates the addition of new syntactic constructions for spoken language. However, we also believe that some phenomena should be handled at parsing level rather than at (meta)grammar level. For instance, this idea could be explored for lists, handling them by applying the same rule several times to each layer.

7. Integration of FRMG into Rhapsodie's annotation process

As outlined in Section 5, the annotation of the illocutionary unit (10a) is analyzed and transformed into a list of simpler segments containing neither lists nor discourse markers; a sample segment is (10b). We parse segments like (10b) with FRMG and extract the dependency structure (Figure 5) from the complete analysis.

(10) a. *"euh" elle en a profité pour { voler une baguette de pain* | *^et s'enfuir } //*

[Rhap-M0002, Rhapsodie]

‘"uh" she took the opportunity to { steal a baguette | ^and run away} //’

b. *elle en a profité pour voler une baguette de pain*

‘she took the opportunity to steal a baguette’

Figure 5. The FRMG dependency structure

Figure 6. The resulting Rhapsodie dependency structure

The tokenization of FRMG is based on the *Lexique des Formes Fléchies du Français* (*Lefff*, Clément *et al.* 2004), which contains many multi-word expressions included on semantic grounds. An example is the multi-word preposition *tout au bout de* ‘at the very end of’. Rhapsodie follows the rule that tokens should be handled as multi-words only if their inherent syntactic structure is opaque (see Chapter 4). To realize this step homogeneously, we automatically collected all tokens containing spaces or hyphens and we manually created a file containing the segmentation these expressions into their internal dependency structure. The analysis of “tout au bout de” as a unique block is thus broken up into the following sub-structure (see Figure 7).

Figure 7. Dependency structure of a multi-word preposition

This correction step also includes certain inverse steps where falsely split tokens are glued together again, for example *micro-processeur* (FRMG cuts at the hyphen) or *chef-d'œuvre* (FRMG cuts at the apostrophe). The correction file contains close to 500 entries, including splitting-up, gluing together as well as other systematic corrections of FRMG’s output.

8. Correction with Arborator

Syntactically annotating spoken texts with dependency structures was unknown territory and it was expected that the biggest chunk of the annotation process would have to be manual; the automatic pre-analysis can only speed up and homogenize the work.

The software we used and further developed for the manual annotation was the Arborator (Gerdes 2013), an online tool for collaborative annotation of dependency corpora. Users of the Arborator have to sign in before accessing the corpus and they systematically work on their own version of the dependency tree. A hierarchy of administrators and validators provides the possibility of controlling the access to the other annotators' analyses (Figure 8).

Figure 8. Arborator's project page

Apart from the collaborative online configuration, what distinguishes the Arborator from other annotation tools is the graphical annotation interface allowing modifications directly to the dependency representation, unlike for example NotaBene (Mazziotta 2010) which relies on a standard graphical interface similar to file managers. For dependency structures it is even more crucial to develop a specific graphical interface than for phrase structure as dependency analysis consists of creating links between any pair of possibly non-contiguous words. Simple tables (e.g. CoNLL format, see Section 11) can represent dependency structures, but it is difficult for a human annotator to read and correct these formats without translating them into dependency graphs.

In the Arborator, the annotator directly edits the dependency trees, which are drawn directly onto the sentences as in the figures above. Pulling one word with the mouse and dropping it onto another word creates a dependency link between those two words. It is even possible to create DAG's, that is, words with multiple governors.

The administrator distributes the texts (in most cases pre-annotated) to the annotators and can at any time view the annotators' progress. Validators and administrators also have access to a graphical comparison tool: if a sentence has more than one dependency annotation, the differences can be rendered graphically (see Figure 9). This allows for a smooth and fast process of unifying the different analyses, used in level 7, see Section 10.

Figure 9. The comparison mode shows the differences between annotations (identical annotations in gray)

9. Agreement Analysis

Of the six dependency annotators, three were considered expert, in that they participated in the initial elaboration of the dependency analysis used in the annotation. According to Landis and Koch (1977)'s interpretation of the kappa coefficient, the agreement between the majority of the annotator pairs is considered almost perfect (inter-annotator agreement is between 0.76 and 0.95 for the different couples of annotators, with an average of 0.81). As expected, the least disputed relation was that of the subject: for 95.82% of the cases where an annotator considers a *sub* dependency the other annotator does the same. The percentage of agreement is particularly low for the oblique complement: only 41.76% of the *obl* links are also annotated *obl* by the other annotator and 31.49% of them are annotated as adjuncts (*ad)* by the other annotator, reflecting the difficulty of confining the scope of verbal valency. It is also noteworthy that paradigmatic relations were not confused with government relations, but 30.16% were forgotten or attributed to another conjunct by the other annotator.

The following matrices show the distribution of an annotator's choice of label for a given dependency (Table 1) and paradigmatic link (Table 2) in relation to the label assigned by a second annotator for the same link, if the link was annotated as a paradigmatic relation.

Table 1. Distribution of dependency relation labels

in comparison to a second annotator's choice (in %)

Table 2. Distribution of paradigmatic relation labels

in comparison to a second annotator's choice (in %)

	_coord	**_hyper**	**_inter**	**_disfl**	**_reform**	**_dform**	**_negot**	**none**	**total**
_coord	*72.29*	6.69	0.64	0.32	5.10	8.92	2.86	3.18	100
_hyper	31.34	*14.93*	2.99	1.49	20.90	17.91	2.98	7.46	100
_intens	2.53	2.53	*65.82*	21.52	0.00	1.27	2.53	3.80	100
_disfl	0.18	0.18	3.05	*78.24*	15.11	0.72	1.08	1.44	100
_reform	5.93	5.19	0.00	31.11	*33.33*	14.44	7.78	2.22	100
_dform	18.79	8.05	0.67	2.69	26.17	*32.89*	4.70	6.04	100
_negot	16.07	3.57	3.57	10.72	37.50	12.50	*12.50*	3.57	100
none	23.26	11.63	6.98	18.60	13.95	20.93	4.65	*0.00*	100

10. Post-validation correction

According to the calculation of inter-annotator agreement, based on the distribution and labeling of dependency relations, the level of agreement was high, showing that in most cases the annotation guide had been well followed and that there was relatively little disagreement amongst the analyses. However this calculation does not take into account errors made consistently by all annotators. Despite having two annotators plus a validator for the annotation of the treebank, human error remained relatively high, in particular for the direction of paradigmatic links, inherited dependencies, and parts of speech. In order to remedy this problem, we added an additional step to the annotation procedure.

For this last step of the workflow, we developed rules to determine the well-formedness of the trees, and searched for structures that did not obey them. In total we had 16 rules, most of which checked the compatibility between the dependency and the parts of speech of the words concerned. For example, a determiner is necessarily governed by a noun. The examples were automatically detected and manually corrected (Table 3).

Table 3. Distribution of detected errors concerning paradigmatic links

Type of error	Number of nodes corrected
More than one paradigmatic link to a single node	4
Paradigmatic link without inherited dependency	113

Paradigmatic link to a node which also has a plain dependency	49
Different types of plain/inherited dependency	24
Different types of inherited dependencies	8
Total	198

11. The distributed treebank format

The final step of the annotation chain consists of the fusion of the dependency and macrosyntactic annotations, as well as certain prosodic annotations for intono-syntactic studies. Among the various choices of annotation formats, we chose a tabular format, based on the CoNLL format (Buchholz & Marsi 2006), which follows standard conventions and is easy to parse.

The CoNLL format for dependency treebanks consists of a simple table with a numbered row for each lexeme. The columns contain the lexeme, the lemma, the POS as well as the governor with its functional relation (Table 4).

Table 4. Parts of example (4) in the CoNLL format for dependency treebanks

ID	Lexeme	Lemma	POS	Governor	Dependency relation
8	le	le	D	9	dep

9	point	point	N	14	sub
10	de	de	Pre	9	dep
11	vue	vue	N	10	dep
12	esthétique	esthétique	Adj	11	dep
13	se	se	Cl	14	obl
14	dégage	dégager	V	0	root

Multi-governor relations require either assigning as many columns as the lexeme has governors or duplicating the lexeme's row, thus allowing the specification of an additional governor. It is this second option which we chose for the output format of the Arborator (as the number of governors does not have to be fixed in advance of the annotation process). In Table 5, the lexeme *point* has two governors, encoded in different rows.

Table 5. The Arborator's encoding of multiple governors

ID	Lexeme	Lemma	POS	Governor	Dependency relation
1	le	le	D	2	dep
2	**point**	**point**	**N**	**9**	**para_hyper**
2	**point**	**point**	**N**	**14**	**sub_inherited**

3	de	de	Pre	2	dep
4	vue	vue	N	3	dep

In the Rhapsodie dependency structure, however, we have a maximum of three governors per token to which we assigned specific columns (Table 6).

Table 6. Rhapsodie table format encoding multiple governors

ID	Lexeme	Lemma	POS	Gov.	Dependency relation	Inherited Gov.	Inherited Relation
1	le	le	D	2	dep		
2	**point**	**point**	**N**	**9**	**para_hyper**	**14**	**sub_inherited**
3	de	de	Pre	2	dep		
4	vue	vue	N	3	dep		

The final distribution format of the Rhapsodie format has a tokenization that is theoretically neutral by relying simply on orthographic conventions: each chain of letters, spaces, apostrophes, and hyphens is given the status of tokens and therefore a separate line. This also allows for an easy integration of inter-word information such as the length of pauses. The delimitation of lexemes is encoded in a separate column *Lexeme* indicating the extension of the lexeme across different tokens (*B* for *begin* and *I* for *inside*, see Table 7). Such a table can be read by Arborator (Figure 10) and modified using Arborator.

(11) parce qu'en fait euh j'aime la biologie [Rhap-M1001, Rhapsodie]

'because in fact uh I like biology'

Table 7. Some columns of the Rhapsodie syntactic distribution format for example (11)

ID	Token	Lemma	Lexeme	POS	Governor	Dependency relation
1	parce	parce que	B	CS	0	root
2		^	I			
3	qu	^	I			
4	'	^	I			
5	en	en fait	B	Adv	0	root
6		^	I			
7	fait	^	I			
8						
9	euh	euh	B	I	0	root
10						
11	j	je	B	Cl	5	sub
12	'	^	I			

13	aime	aimer	B	V	1	dep
14						
15	la	le	B	D	7	dep
16						
17	biologie	biologie	B	N	5	obj

Figure 10. Arborator display of Table 7

The tabular representation for the lexeme extension also makes it possible to easily encode any kind of chunk by assigning a column that indicates the chunk's extension. In this way, we encode macrosyntactic and prosodic tree-structures.

12. Conclusion

The processing chain is now stabilized and can be used for future projects. All the tools have been published under open-source licenses and can therefore be easily adapted for specific needs. The Arborator in particular can be a cornerstone of any dependency annotation project due to its high stability and sophistication.

For Spoken French, parsers can be trained on the macrosyntactic and paradigmatic annotation proposed by the Rhapsodie treebank, thus automatizing levels 1 and 2 of our chain. Moreover, training a parser on the dependency analysis of spoken language may allow us to avoid adapting data to conform to the input for parsers trained on written language, which means that it is possible to sidestep the whole folding and unfolding process (levels 3 and 5). However, for other languages where equivalent linguistic resources are not available, the proposed chain may remain the appropriate approach for creating them. For our new project on Naija, the English creole spoken in Nigeria, we decided to manually annotate macrosyntax and lists, to manually annotate a dependency structure including micro and macrosyntax, as well as the macrosyntactic tags as punctuation signs, and to train the MATE parser (Bohnet 2010).