From raw corpus to word lattices: robust pre-parsing processing

Benoît Sagot and Pierre Boullier

INRIA - Projet Atoll
Domaine de Voluceau, Rocquencourt B.P. 105
78153 Le Chesnay Cedex, France
{benoit.sagot,pierre.boullier}@inria.fr

Abstract

We present a robust full-featured architecture to preprocess text before parsing. It converts raw noisy corpus into a word lattice that will be used as input by a parser. It includes sequentially named-entity recognition, tokenization and sentence boundaries detection, lexicon-aware named-entity recognition, spelling correction, and non-deterministic multi-words processing, re-accentuation and de-/re-capitalization. Though our system currently deals with French language, almost all components are in fact language-independent, and the others can be straightforwardly adapted to almost any inflectional language. The output is a lattice of words that are present in the lexicon. It has been applied on a large scale during a French parsing evaluation campaign, showing both extreme efficiency and very good precision and recall.

1. Introduction

Pre-processing of raw text before parsing is usually seen as an easy task on which no further research is worth doing. However, experiments show that this step is crucial when dealing with real-life corpora, and that available tools are not always satisfying, for example because they lack a spelling error correction component, because they are specialized in some kind of corpus, or because they are not able to handle non-determinism.

We recently took part to the French parsing evaluation campaign EASy, and had to parse a set of about 35,000 sentences coming from very diverse corpus (journalistic, e-mail, medical, legal, oral, literature, and so on) with a correct to very poor quality. Hence, we had to design a very robust pre-processing system to turn this extremely noisy text into individual tokenized sentences,[1] with a minimal loss of information, and without losing the link between output words[2] and original tokens of the corpus.[3]

We first give an overview of the architecture of our system. Then we briefly focus on the different components, namely named-entity recognition steps, tokenization and spelling error correction, and non-deterministic multi-word identification, re-accentuation and de- or re-capitalization. We conclude with a brief evaluation of the system.

2. Overall architecture

The overall architecture of our pre-processing system is illustrated in Figure 1. During the whole process, input tokens are stored in *comments* (surrounded by braces and decorated with their position in the input string) which are immediately followed by the associated word-form.[4]

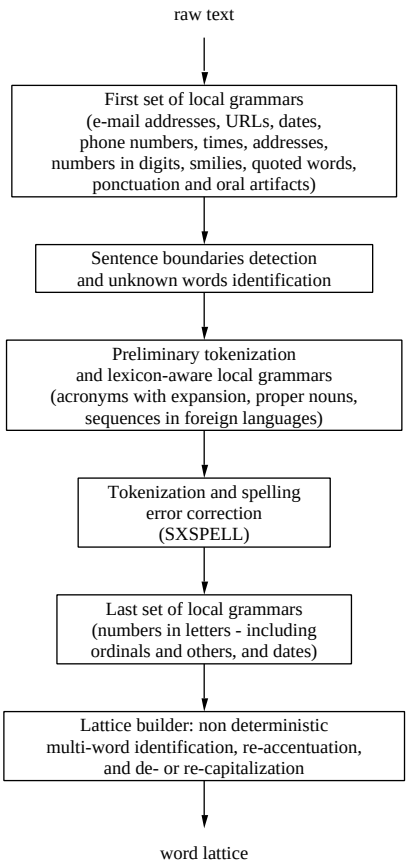

Figure 1: Overall architecture of the system.

For example,

contactez-moi au 1 av. Foch, 75016 Paris, ou par e-mail à my.name@my-email.com.

[1] Corpora were in fact already splitted into sentences, but only partly. Hence, we almost ignored this segmentation.

[2] In this paper, we use the badly defined word *word* as a synonym of *word form* in the sense of (Clément and de La Clergerie, 2004).

[3] This is needed to be able to link back the output of the parser to tokens of the corpus, even if words can cover many input tokens, and tokens many words.

[4] We use the following conventions: an artificial token (e.g., a named-entity identifier) starts with a "_" ; in the corpus, characters "_", "{" and "}" are replaced by the artificial tokens _UNDERSCORE, _O_BRACE and _C_BRACE. Thus, these three characters are available as meta-characters.

will become, if ignoring ambiguities, something like

{contactez$_{0..1}$} contactez {-moi$_{1..2}$} moi {au$_{2..3}$} à {au$_{2..3}$} le {1 av. Foch, 75016 Paris$_{3..9}$} _ADDRESS {,$_{9..10}$} , {ou$_{10..11}$} ou {par$_{11..12}$} par {e-mail$_{12..13}$} e-mail {à$_{13..14}$} à {my.name@my-email.com$_{14..15}$} _EMAIL {.$_{15..16}$} . {.$_{15..16}$} _SENT_BOUND.

3. Sentence boundaries detection and named-entities recognition

Real-word corpus are not like sentences built by linguists. They include sequences of tokens that are not analysable at a syntactic nor morphological level, but belong to productive patterns, which means that they have to be identified before spelling error correction. Most of them are grouped under the term *named entities* (Maynard et al., 2001). However, we will use this term in a slightly broader sense, including all such sequences of token, even if not usually considered as named entities (e.g., numbers). We call *local grammar* a grammar recognizing named-entities of a given family.

We designed a set of large-coverage robust[5] local grammars, implemented as *perl* programs involving numerous regular expressions.

Some named entities contain characters that are usually ponctuation marks, most importantly the period (e.g., in URLs), but also the comma (e.g., in addresses) and all kind of other characters (e.g., in smilies). Therefore, some local grammars must be applied *before* tokenization, including in our system:

e-mail addresses with detection of erroneous spaces,

URLs with detection of many kinds of errors and formats,

dates including various formats as well as date ranges (e.g., *du 29 au 31 janvier*[6] will become *du _DATE au _DATE*, even if *29*, if isolated, would not be recognized as a date),

telephone numbers in various formats,

times including several formats as well as time ranges (e.g., *2-3 heures*, *3 ou 4 minutes*[7], etc.),

addresses in a lot of different formats,

numbers including different formats, as well as ordinals written with digits (e.g., *2ème – 2nd*),

smilies such as *:-)* or *:D*,

quoted words : *un «test»*[8] becomes *un {«test»} test*,

formatting artifacts to deal with special ponctuation phenomena (like replacing *(...)* by a single-word *(...)*) and with oral transcription artifacts (repetition more than twice of the same word, or more than once if it belongs to a predefined list, removal of hesitation markers, and so on).

After the application of these local grammars, we segment the text in sentences. This task is performed by a huge set of *perl* regular expressions that extends the basic ideas proposed for example in (Grefenstette and Tapanainen, 1994), helped by a list of known words containing a period (often abbreviations). It is designed to be able to handle all kind of false negatives and false positives that arise in real-life corpus. After this step, the artificial word *_SENT_BOUND* represents sentence boundaries.

We then apply the tokenizer and spelling error correcter described in the next section in a degraded way, in the sense that no spelling error correction is performed, but the text is tokenized in the same way it would be with error correction. The aim of this is to identify words in the input string that can not be analysed as known words (present in the lexicon or *easily* correctable) or combinations of known words (in French, things like *l'idée*, *anti-Bush* or *done-m'en*, for example,[9] are valid combinations of correctable words – *done* should be *donne*).

Once unknown words are identified (recall that *unknown* means here that it is not tokenizable in a way that would give only words present in the lexicon or easily correctable), special local grammars that take this information into account are applied. They recognize:

acronyms that are followed or preceded by their expansion, with various typographic possibilities,

proper nouns preceeded by a title (like *Dr.* or *Mr*),

phrases in other languages than French.

The two last local grammars deserve a special comment. They are based on the following technique. Let $w_1 \dots w_n$ be a sentence whose words are the w_i's. We define a tagging function t that associates (thanks to regular expressions) a tag $t_i = t(w_i)$ to each word w_i, where the t_i's are taken in a small finite set of possible tags (resp. 9 and 12 for the two local grammars). Hence, a sequence of tags $t_1 \dots t_n$ is associated to $w_1 \dots w_n$. Then, a (huge) set of finite transducers is performed over $t_1 \dots t_n$, transforming it in a new sequence $t'_1 \dots t'_n$ of tags. If in this sequence a sub-sequence $t'_i \dots t'_j$ matches a given pattern, then the corresponding sequence of words $w_i \dots w_j$ is considered recognized by the grammar.

Let us consider for example the following sentence,

Peu après , le Center for irish Studies publiait . . .,[10]

where *Center*, *irish* and *Studies* have been identified as unknown words. It gets the following tags: `cnpNFFucn...`(`c` stands for *capitalized*, `n` for *probably French* (default case), `p` for *ponctuation*, `N` for *known as French*, `F` for *known as foreign* and `u` for *unknown*). Regular expressions on these tags lead to `cnpNffffn...`, where *f* stands for *foreign*, meaning that *Center for irish Studies* is recognized as a phrase in a foreign language.[11] The sentence becomes (*_FP* stands for *foreign phrase*):

Peu après , le {Center for irish Studies} _FP publiait . . .

[5] By robust, we mean that named-entities with errors are also recognized, like *ttp:/strange.url.com /index.html*.

[6] *from the 29th to the 31st of january*

[7] *3 or 4 minutes*

[8] *a "test"*

[9] *the idea*, *anti-Bush* or *give me some*

[10] *Soon afterwards, the Center for irish Studies published . . .*

[11] In fact, we also designed a prototype tool to identify the language of such a phrase. In this case, the correct answer, English, is correctly found.

4. Tokenization and spelling error correction

4.1. An isolated-word corrector: SXSPELL

The next step in our system is the spelling error corrector. Real-life corpus have diverse rate of spelling errors, that can go from virtually zero (as in literature corpus) to an extremely high rate (as in e-mail corpus). Moreover, if they remain uncorrected, misspelled words become unknown words for the parser. This must be avoided as much as possible, since they usually get default underspecified syntactic information, which leads both to low precision and very high ambiguity at the syntactic level. Therefore, we designed a spelling error corrector, named SXSPELL.

A lot of work has been done on spelling correction (see for example the review of (Kukich, 1992)). Techniques used for isolated-word correction mainly fall in two categories: trained and untrained. Trained techniques cover stochastic (often n-gram based) techniques and neural nets. Untrained techniques include *minimum edit distance* (based on operations like insertion, deletion, substitution or swapping) and *rule-based* techniques (based on context-sensitive rewriting rules, the origin of which comes from finite-state phonology). The latter is clearly more powerful and more adapted to the task,[12] but the cited operations can also be useful as such. Hence, our corrector is rule-based, but these operations are also available to build underspecified rules.

Applying a rule is called an *elementary correction*. We associate to each rule a *local cost* and a *composition cost*. The total cost of a correction is the sum of the local costs of all elementary corrections, plus, if more that one elementary correction has been performed, the sum of all composition costs. This allows to have a global cost that is more than the sum of local costs. The best correction is of course the one with the lower total cost.

Our purpose was to have an efficient implementation of these simple techniques, even if used with numerous appropriate rules and a real-size spelling lexicon (our spelling lexicon for French language has more than 400,000 different inflected forms and parts of multi-word units). To achieve this goal, we considered the spelling lexicon as a deterministic finite automaton $\mathcal{F}$, the input word w as a finite transducer $\mathcal{T}_w^0$, and rewrite rules as finite transducers $\mathcal{T}^i (i > 0)$. First, we compute the finite transducer $\mathcal{T}_w^{all}$ of all possible sequences of characters that can be obtained from w by applying the rules, and their costs.[13] Then we extract from $\mathcal{T}_w^{all}$ all words that indeed exist in the lexicon, by intersecting $\mathcal{F}$ with $\mathcal{T}_w^{all}$.

The difficulty of this approach is not the underlying theory, which is well known, but comes from the size of the automata that we have to handle. Indeed, with a typical number of rules of several hundreds, the automaton $\mathcal{T}_w^{all}$ has easily billions and billions of paths. And it has to be intersected with $\mathcal{F}$ and its 400,000 paths. Therefore, we extensively used tabulation and compact representation techniques. One must admit that the feasability of such an approach was not *a priori* clear, but we have very good results, both in terms of quality (with appropriate rules) and response time (with an appropriate threshold cost).

4.2. In-sentence spelling correction

Spelling error correction can not be performed on a purely isolated-word basis. Indeed, at least four phenomena involve the environment of a word during recognition by the lexicon or during its correction:

- words starting with a capital letter,
- words that have initial position in the sentence (which interacts strongly with the previous point),
- multi-words that are consequence of productive derivational morphology (e.g., *anti-Bush*) or syntactic agglutination (e.g., *préchoisis-t'en*,[14] that must be tokenized as *pré- / choisis / t' / en*),
- spelling errors that involve more than one token (e.g., *corre ction* instead of *correction*) or more than one word (e.g., *unproblème* instead of *un problème*[15]).

Hence, we developed a full-featured in-sentence spelling corrector, which is able to deal with these phenomena and to send queries to SXSPELL, so as to simultaneously tokenize and correct the text (we do not correct capitalized words, but other unknown words can remain if no correction is found for a word that costs less than a given threshold). It turned out that the interaction between tokenization of multi-words, capitalization and spelling error correction is not easy to deal with, especially when one deals with the first token of a sentence. However, we defined some heuristics that give pretty good results.

5. Non-deterministic light spelling correction and multi-word identification

In many cases, the simple concatenation of words cannot express the subtleties and ambiguities of natural languages. Therefore, the output of our process is a lattice (or DAG, standing for Direct Acyclic Graph) of word-forms (or words), which can be given as input to our syntactic parsers.[16] Moreover, we do not produce only *simple* DAGs in the sense of (Barthélemy et al., 2001), because they are not sufficient (see for example Figure 2).

Let us consider the French phrase *pomme de terre cuite*.[17] Each word is a valid inflected form, as are the compound words *pomme de terre* and *terre cuite*.[18] Therefore, it is represented by the DAG shown in Figure 2.

[12] A very simple example of that is the following: *o* and *eau* are two possible spellings for the [o] sound in French. Thus, transforming *o* into *eau* is a reasonable rule. It is more natural and more sensible w.r.t. correction costs, to see this operation as a replacement of *eau* by *o* than as two deletions followed by a substitution.

[13] Of course, a threshold cost can be given as a parameter, thus preventing from computing too many very costly corrections.

[14] *pre-chose one of them for you*

[15] *a problem*

[16] Most classical parsers are not able to handle DAGs as input, which leads to the need of an extra step before parsing, namely (super-/hyper-)tagging, which may delete valid alternatives.

[17] This can mean either *cooked potato*, *cooked clay apple* or *terracotta apple*, which leads to the 3 different paths in the graph.

[18] respectively *potato* and *terracotta*.

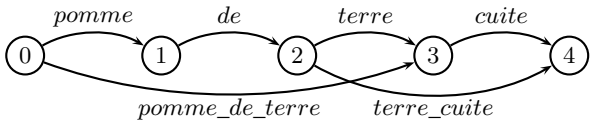

Figure 2: DAG associated to *pomme de terre cuite*.

On the contrary, French language (as others) has *agglutinates*. For example, *du* is either a valid word (meaning *some*) or must be decomposed as *de le* (meaning *of the*). It is therefore represented as shown in Figure 3.

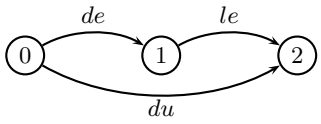

Figure 3: DAG associated to *du*.

These operations are performed as follows. The input of the DAGing step is considered as a (linear) DAG $\mathcal{D}$. To each compound and to each agglutinate of the lexicon is associated a transducer. The composition of all these transducers is applied to $\mathcal{D}$, possibly creating new paths.

The resulting DAG is then passed through other transducers that create other alternatives. For example, capitalized words for which the non-capitalized word is present in the lexicon are represented as an alternative between both. Unknown words remaining at this point (including many capitalized words) and for which adding a diacritic on some letters leads to a known word are also represented as an alternative between both.[19] Finally, unknown words in the DAG are all replaced by one of two special entry of the lexicon, *_UW* and *_uw*, according to their capitalization. The resulting DAG is the final output.

6. Evaluation

The evaluation of such a system is difficult, because we lack an appropriate gold-standard corpus. However, some insights can be given thanks to tests we did on a 1,100,000-word journalistic corpus.[20] The whole process[21] takes 13'01", which corresponds approximately to 1400 tokens/sec. Considering the complexity of the performed tasks, and in particular the sizes of the automata involved in SXSPELL, this is a very good performance.

We also selected a few named-entity families for which over-generating detectors can be easily designed, so as to allow a manual validation. Results are shown in Table 1.

The evaluation of the sentence boundary detection needs a manual annotation. We did it on the first 400 sentences of the corpus, which gives a 100% precision rate and a 100% recall rate. This is pretty satisfying, considering the fact that our journalistic corpus is full of quotations, footnotes, book references and meta-information that makes sentence boundary detection pretty difficult.

Named-entity family	Occ.	Precision	Recall
URLs	174	100%	100%
(surface) addresses	35	100%	100%
Phrases in foreign lang.[22]	42	83%	88%

Table 1: Partial evaluation of named-entities recognition.

We do not give any evaluation result for the spelling error corretor. Indeed, it depends mostly on the quality of the corpus (rate of rare and foreign words, quality of the spelling) and even more on the quality of the lexicon, whose evaluation is not our purpose here. Quantification of these crossed influences deserves further investigation.

7. Conclusion

We have presented a full-featured architecture that produces words lattices out of raw text, and is able to handle various phenomena that occur at a high frequency in real-life corpus. This includes several named-entity families, spelling errors, tokenization ambiguities while detecting sentence and word boundaries, and lexical ambiguities between words differing only by diacritics or capitalization. Moreover, our system is extremely efficient, and gives high-quality results. Such a pre-processing is a crucial step to be able to parse correctly real-life corpus.

Further work include an even better treatment of derivational morphology, extension of existing named-entity recognizers and design of new ones, and a few adaptations to be compliant to the current ISO working draft on normalization of morphosyntactic annotation (Clément and de La Clergerie, 2004), based on XML representation of tokens, words (or word-forms) and lattices. We also intend to make the whole system available under a free-software licence in the near future.

8. References

Barthélemy, François, Pierre Boullier, Philippe Deschamp, and Éric de La Clergerie, 2001. Guided parsing of range concatenation languages. In *Proceedings of ACL'01*. Toulouse, France.

Clément, Lionel and Éric de La Clergerie, 2004. Terminology and other language resources – Morpho-Syntactic Annotation Framework (MAF). ISO TC37SC4 WG2 Working Draft.

Grefenstette, Gregory and Pasi Tapanainen, 1994. What is a word, what is a sentence? Problems of tokenization. In *Proceedings of the 3rd Conference on Computational Lexicography and Text Research*. Budapest, Hungary.

Kukich, Karen, 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439.

Maynard, Diana, Valentin Tablan, Cristian Ursu, Hamish Cunningham, and Yorick Wilks, 2001. Named entity recognition from diverse text types. In *Proceedings of RANLP 2001*. Tzigov Chark, Bulgaria.

[19] We also try and correct parts of compound words that do not exist as standalone words but do not take part one of their compound words. For example, *brac* in French exists only as part of the phrase *bric à brac*. Thus, *un brac* has not been corrected by the previous step, but is corrected here as *un bras*.

[20] We did evaluations on the different corpus of the parsing evaluation campaign cited above, but we are not yet allowed to publish these results. We can just say that the frequency of detection of named-entities strongly depends on the kind of corpus.

[21] Test performed on an AMD Athlon™ XP 2100+ (1.7 Ghz) architecture running Mandrake Linux 10.1.

[22] Test performed only on the first 2000 sentences, because manual annotation is necessary.