LexMan: a Tokenizer and Morphological Analyzer with Transducers

Alexandre Manuel Fajardo Vicente

IST – Instituto Superior Técnico
L2F – Spoken Language Systems Laboratory – INESC-ID Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
amfv@ist.utl.pt

Abstract. The first two tasks performed by L2F’s NLP chain are the tokenization and morphological analysis. Actually, these tasks are performed by two different modules. This paper describes two approaches, based on transducers, that join these two tasks in one module, the LexMan. This allows the transfer of joining rules (for compound words identification) from the rule-based disambiguation module, RuDriCo, to LexMan.

Furthermore, we describe the changes made to the dictionary used by the morphological analyzer to complement the words of the dictionary with prefixes that can be added to them. That allow us to increase the recall of words that can be identified and tagged during the morphological analysis.

1 Introduction

Natural Language Processing (NLP) is a sub-field of the Artificial Intelligence area that has, as one of its main goals, the enhancement of the interactions between humans and computers, allowing humans to interact with machines as naturally as possible, using natural language. Automatic translation and spelling correction of text are examples of practical applications of NLP.

The systems developed in this area are usually composed by different modules, organized in a pipeline. The L2F\(^1\) uses a NLP chain called STRING [Mamede et al., 2012] (see Figure 1), to perform various tasks like anaphora resolution, Named Entities identification and classification and extraction of semantic relations between Named Entities. The first module receives the input text and tokenizes it. This module uses regular expressions, written in the Perl Programming Language, to identify words (simple and compounds), punctuation and others symbols. Some sequences of text like numbers and numerals (cardinals, ordinal, fractional, and roman numerals), HTTP, IP and email addresses, and biblical references are treated differently. Their tokens are formed but they are also tagged in this module. The result of the tokenization is sent to the next module.

\(^1\) Spoken Language Systems Laboratory at INESC-ID Lisboa.
The second module, LexMan\textsuperscript{2} [Diniz and Mamede, 2011], is a morphological analyzer that receives the result of tokenization from the first module of the STRING as input. This module attaches to each token that was not previously tagged by the first module, all of its possible part-of-speech (POS) tags. The tag set has 12 categories and is composed by 11 fields. At the end, the module performs a segmentation into sentences.

The STRING’s processing continues with RuDriCo [Diniz, 2010] [Diniz et al., 2010]. This module applies a set of rules to its input. There are two types of rules: disambiguation and segmentation rules. The disambiguation rules are applied to the ambiguous tokens to which more than one tag has been attributed by LexMan. The segmentation rules can be joining or expansion rules. The joining rules change the segmentation by converting two or more tokens into a single one (for example, a compound like lua-de-mel “honeymoon”). The expansion rules divide a token into two or more tokens (for example, a contraction of a preposition and a determiner neste = em + este “in” “this”). Figures 2 and 3 represent the segmentation of the sentence A República Checa é na Europa. (Czech Republic is in Europe.”) before and after RuDriCo’s processing. The joining rules are used to join the tokens of compound words like República Checa. The word na is the result of the contraction of em with a, therefore it is divided into two tokens. MARv [Ribeiro, 2003] is a statistical part-of-speech disambiguator.

\textbf{Fig. 1.} L\textsuperscript{2}F’s NLP chain (STRING)

\begin{center}
\begin{tikzpicture}[node distance=2cm,auto]
  \node (input) at (0,0) {Input};
  \node (tokenizer) [right of=input] {Tokenizer};
  \node (lexman) [right of=tokenizer] {LexMan};
  \node (ru) [right of=lexman] {RuDriCo};
  \node (mar) [right of=ru] {MARv};
  \node (output) [right of=mar] {Output};
  \draw[-latex] (input) -- (tokenizer);
  \draw[-latex] (tokenizer) -- (lexman);
  \draw[-latex] (lexman) -- (ru);
  \draw[-latex] (ru) -- (mar);
  \draw[-latex] (mar) -- (output);
\end{tikzpicture}
\end{center}

\textbf{Fig. 2.} Segmentation of the sentence A República Checa é na Europa. before RuDriCo’s processing

\begin{center}
\begin{tabular}{ccccccc}
A & República & Checa & é & na & Europa & .
\end{tabular}
\end{center}

\textbf{Fig. 3.} Segmentation of the sentence A República Checa é na Europa. after RuDriCo’s processing

\begin{center}
\begin{tabular}{ccccccc}
A & República Checa & é & em & a & Europa & .
\end{tabular}
\end{center}

\textsuperscript{2} Lexical Morphological Analyzer
This module receives the output produced by RuDriCo and uses the Viterbi algorithm [Viterbi, 1967], to choose the most likely part-of-speech tag for the ambiguous tokens that still have more than one tag attached after RuDriCo’s processing. The MARv’s language model is based on second order models (trigrams), which encode the label contextual information, and unigrams, which encodes the lexical information.

XIP\textsuperscript{3} [Xerox, 2003] is the last module of STRING. It’s the module responsible to process the syntactic analysis.

The main goal of this work is to join the tokenization and morphological analysis into one module, the LexMan, using transducers. The LexMan already uses a transducer as a dictionary to tag the tokens. Each token created by the tokenizer that doesn’t already have a tag is transformed into a transducer. That transducer is composed with the transducer of the dictionary and the transducer produced by the composition is analyzed to retrieve the tags of the segment. Three transducers are built to form the final transducer of the dictionary. There is a transducer for the verbs, for the non-verbs and an extra one for the clitics that do not need to be concatenated to a verbal form. Figure 4 shows how the transducer of the non-verbs is built. The other transducers are built in a similar way. The last three transitions of each path are used to index files that keep the morphological information of each word.

The changes made to the tokenization process also allow us to transfer, from RuDriCo to LexMan, the joining rules that do not need any type of context to be applied. With less rules to apply, we expect that RuDriCo’s performance improves.

Finally, the words that are included in the transducer of the dictionary are complemented with the addition of prefixes. This way, the recall of words that can be identified and tagged using the dictionary can be increased.

\textsuperscript{3} Xerox Incremental Parser
2 State of the Art

There is a relation between regular expressions and transducers. A regular expression can be converted to a transducer [Karttunen et al., 1996] [Beesley, 1998] [Karttunen, 2001]. This means that transducers can be used to tokenize a text.

Transducers can be built using a set of operations like union, concatenation, intersection and difference. This means that the construction of a transducer can be divided in sub-problems. Small and simple transducers can be combined to form a more complex transducer. The set of operations that can be applied to a transducer is available using libraries. Two known libraries are the FSM library, developed by AT&T, and the OpenFst library, developed by Google Research and NYU’s Courant Institute. These are some well known systems based on transducers:

- SMORPH [Aıt-Mokhtar, 1998],
- INTEX [Mota, 1999].

SMORPH is a morphological analyzer that groups the tasks of normalization, tokenization and morphological analysis, which precede the syntactic analysis, using transducers. SMORPH was built after a careful analysis of the most common issues that these systems must solve.

INTEX is a system that was built with the purpose of being used in the automatic analysis of large-sized texts (millions of words). One of its modules is a morphological analyzer, based on transducers. The transducers can be used during the construction of more complex transducers.

The use of transducers to perform the tokenization is an approach that makes possible to join the tokenization and the morphological analysis in one single task [Aıt-Mokhtar, 1998] [Garrido-Alenda et al., 2002].

3 Tokenizer

The Figure 5 represents the transducer that was developed to perform the tokenization. The transducer of the tokenizer is made of multiples transducers which are combined together, using operations from the transducer libraries, to form the transducer represented in Figure 5.

The regular expressions that matched numbers and numerals, HTTP, IP and email addresses and biblical references were converted to transducers. All the other regular expressions and the joining rules from RuDriCo were included in the transducer of the dictionary. A transducer for unknown words was built. That transducer is responsible to tokenize sequences of characters that are unknown to the others transducers. Finally, there are three types of separators: words separators (e.g. whitespaces), right side separators (e.g. abbreviation) and double side separators (e.g. punctuation symbols). All these transducers are combined with epsilon arcs in a way that make possible the tokenization of a large text.

4 http://www2.research.att.com/~fsmtools/fsm/ (last access: 13-05-2013)
5 http://www.openfst.org (last access: 13-05-2013)
Each transducer starts with a special transition. The output symbol of the first transition is an identifier that allows to know later, during the morphological analysis, which transducer was used to identify the text. A cost is also assigned to that transition.

To join the tokenizer to the morphological analyzer, two approaches were built.

4 First Approach - ShortestPath

The architecture of the first approach is shown in Figure 6. The text received as input is normalized. For instance, newlines and tabs are replaced by whitespaces. Unknown characters (e.g., Chinese characters) are replaced by a marker. The marker will be replaced back to its original character after the LexMan’s processing. The text is converted to a transducer that is composed with the transducer of the tokenizer shown in Figure 5. In this first approach, the transducers of the dictionary and the separators do not contain any morphological information. We then apply the ShortestPath operation on the transducer produced by the composition. This operation looks to the costs of the paths and produces a transducer with the path that has the minimal cost for each token. That gives us the best tokenization possible for the text.
This transducer is then analyzed and the tokens that were found are composed with the transducer of the morphological analyzer. We analyze the transducer produced by each composition and retrieve all the tags for that segment.

If the word is unknown, the composition will produce an empty transducer. In that case, a module called Guesser looks at the termination of the word and tries to guess the correct tags. After the compositions of the tokens, the sentence splitter organizes them according to their sentences. The last task is the reconstruction of the tokens that contain an unknown character. The marker for this character is replaced by the character that originally appeared in the text.

5 Second Approach - Prune

The architecture of the second approach is shown in Figure 7. This approach is different from the first one because it only requires one composition. The
normalization remains as in the first task and the transducer that is created is composed with the tokenizer. The difference is that in this approach, each path of the transducer of the dictionary, the right and the double side separators, maintain the last three transitions that can be used to retrieve the morphological information of the word. After the composition, we apply the Prune operation to select all the paths that have a cost equal to the path with the minimum cost of the transducer. This way, we obtain the best tokenization and we also have access to all their morphological information.

The next tasks remain the same. The sentence splitter divides the tokens according to the sentence they belong and the markers of the unknown characters are replaced by the original character.

6 Third Approach - Prune with Prefixes

The concatenation of a prefix to a word depends on the first letter of the word. In some cases, it may be necessary to apply some transformations to the prefix, like adding a hyphen. We created a module that can generate all the possible prefixes. Given a lemma of the prefix and a rule that decides which transformation needs
to be applied depending on the first letter of the word the prefix’s added to, all the possible versions of the prefix are generated. In Portuguese, the prefixes can be added to nouns, adjectives, adverbs and verbs. For instance, the prefixes anti and anti- can be added to adjectives that begin with the letter r.

Each generated prefix is built in a transducer, like shown in Figure 8. The transducer of the Figure 8 represents a small set of the prefixes that can precede the adjectives that start with the letter r. The paths of the words start where the paths of the prefixes end. This is shown in Figure 9. The equivalent transducers are combined and, at the end, we obtain a new version of the transducer of the dictionary. During the morphological analysis, if the transition with the symbol PREFIXSTART is found, it means that the identified word has a prefix. In that case, the tag is changed to reflect this information.

Fig. 8. Transducer of the prefixes

Fig. 9. Adjectives that start with r and their prefixes
7 Evaluation

7.1 Corpus

Each solution was processed with a corpus. The outputs of the new solutions were compared with the one produced by the original solution. The differences were evaluated one by one to see if they were the results of the changes introduced in this work or an error.

The debug of these results allowed us to correct some minor problems and complete the development of each solution with success.

7.2 Performance

To evaluate the performance, the processing time of different-sized files was measured. These files are characterized in Table 1. Each one of the solution is evaluated and compared to the initial solution, that existed before this work started. Each version was tested with the version of the dictionary that existed at the time. The comparison between the solutions is shown using the chart of the Figure 10. Since both LexMan and RuDriCo were affected by this work, the values compared are calculated using the sum of the processing of these two modules, for each solution. Each file was tested three times to allow us to calculate an average. We tested all the approaches: ShortestPath, Prune and Prune with prefixes. The addition of prefixes allowed us to remove some words from the dictionary because those words could now be identified with the prefixes. We tested that solution with two different dictionaries.

As we can see in Figure 10, as the number of sentences increases, the solutions built in this work had a better performance than the original solution. We can also conclude that the solution based on the Prune operation is better than the one that uses the ShortestPath operation.

In Table 2, we present the differences, in percentage, between the original solution and the new solutions. The new solutions are slower than the original for the smaller file, but for all the others files, they present a significant improvement.

<table>
<thead>
<tr>
<th>File</th>
<th>N° of sentences</th>
<th>N° of words</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parte08-1.txt</td>
<td>1</td>
<td>32</td>
<td>0.21</td>
</tr>
<tr>
<td>Parte08-10.txt</td>
<td>10</td>
<td>380</td>
<td>2.22</td>
</tr>
<tr>
<td>Parte08-100.txt</td>
<td>100</td>
<td>2388</td>
<td>14.61</td>
</tr>
<tr>
<td>Parte08-500.txt</td>
<td>500</td>
<td>11189</td>
<td>69.51</td>
</tr>
<tr>
<td>Parte08-1000.txt</td>
<td>1000</td>
<td>22403</td>
<td>140.17</td>
</tr>
<tr>
<td>Parte08-5000.txt</td>
<td>5000</td>
<td>109902</td>
<td>686.74</td>
</tr>
<tr>
<td>Parte08-10000.txt</td>
<td>10000</td>
<td>218530</td>
<td>1368.86</td>
</tr>
</tbody>
</table>

Table 1. Characteristics of the files used during the evaluation.
The addition of the prefixes to the transducer of the dictionary caused some loss in the performance time. On the other hand, the removal of words from the dictionary allowed an increase of performance.

8 Conclusion and Future Work

The experiments here made show that the solutions that were built in the context of this work have a better performance when it comes to process large files. When we processed the file with 10,000 sentences, the new solutions were 65% to 70% faster than the original solution.

The solution based on the Prune operation is more efficient. Furthermore, this solution is more flexible to changes than the original solution. The introduction of prefixes also allowed to increase the recall of words identified by LexMan.

To conclude, we believe that this work is a significant improvement over the original tokenizer and morphological analyzer.

As future work, it would be interesting also to add suffixes to the dictionary. If it happened, the recall of words correctly identified would increase.
References


