Processing Frozen Sentences in Portuguese
Automatic Rule and Example Generation from a Lexicon-Grammar

Ana Isabel Silva Galvão
ana.s.galvao@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa
L²F - Spoken Language Systems Laboratory - INESC-ID Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal

May 2019

Abstract

Frozen sentences are multi-word expressions that constitute a large set of the Lexicon-Grammar of many languages, though their frequency in texts is often very low. Parsing frozen sentences is a challenging task because they are syntactically analyzable strings whose meaning is non-compositional. Given an existing Natural Language Processing (NLP) system for European Portuguese, the main goal of this project is to use the matrix containing the most recent linguistic description in order to be able to correctly translate it to Xerox Incremental Parser (XIP) rules, allowing for it to identify not only manually produced sentences, but also automatically generated ones from the base sentences by applying the transformations authorised by each construction.

In order to achieve that goal the rule generator was rebuilt so that the generated rules include not only the basic structure of the idiom, but also the several transformations or reduction of certain elements to pronouns that may be applied to each sentence. A module that automatically generates these automatically generated sentences from the base sentences was also developed.

Automatic validation was also implemented in order to verify the performance of the system, which was overall improved when compared to the previously existent system, allowing for a more correct and inclusive identification of frozen expressions.

Keywords: Natural Language Processing, Frozen Sentences, Verbal Idioms, Parsing Multiword Expressions, Part of Speech

1. Introduction

Verbal idioms are idiomatic (semantically non-compositional), Multiword Expressions (MWE) consisting of a verb and at least one constraint argument slot [3]. Therefore, they are considered frozen sentences because the verb and at least one of its arguments are frozen together, that is, they present idiosyncratic and semantically unpredictable distributional constraints. This means that, unlike free sentences, their meaning cannot be calculated from each individual component, but rather from the sentence as a whole [3]. Removing any element and replacing it with something else would turn the sentence to its literal meaning or result in an unacceptable utterance. However, usually, one or more of the argument noun phrases are distributionally free, which means that they can vary (within generic distributional constraints) without affecting the global meaning of the sentence. On the other hand, this type of sentences also differs from free sentences because they block transformations that should otherwise be possible, given the syntactic properties of the verb and its arguments [1]. One example of this type of sentences is: O João abriu os cordões à bolsa

In order for the sentence to maintain its meaning:

• None of the complements may have distributional variations (except for the subject);
• The combination abrir-cordões is frozen;
• cordões cannot be replaced by any other expression, nor be modified by adjectives;
• Replacing à bolsa by any other expression would turn the sentence to its literal sense.

Finally, frozen sentences represent a problem for many NLP systems because they must not be treated as a block [2], on the contrary, they have a syntactic structure that yields to analysis, unlike compound lexical items (nouns, adverbs, conjunctions, etc). Besides, their elements can appear discontinuously and they may also present some formal variations, often being ambiguous - the same
sequence may have a literal and a figurative meaning - and in that case only an extended context can disambiguate them [1].

Given these facts, it is possible to conclude that the integration of this specific type of expressions in NLP systems, in order to obtain an accurate semantic parsing, is a challenging task. A great amount of work has been done in this area, such as an European Portuguese annotated corpus built in the scope of the project PARSing and Multi-word Expressions (PARSEM E)\(^1\), an interdisciplinary scientific network devoted to the role of MWE in parsing\(^2\). For the purpose of this project, a previously built lexicon-syntactic matrix was used, which encodes the linguistic information, using the framework of Gross [7]. Its information will then be integrated into a fully-fledged NLP system built for Portuguese, the Statistical and Rule-Based Natural Language Processing Chain (STRING) [9]. The STRING system uses the XIP parser [3] to segment sentences into chunks and extract dependency relations among chunks’ heads [10]. Considering that most idioms have a “normal” syntactic structure, which follows the ordinary word combinatorial rules of the general grammar, STRING’s strategy consists in parsing them first as ordinary sentences and only then identifying specific word combinations, whose meaning should not be calculated in a compositional way. The idiomatic word combinations are identified by a dependency, \texttt{FIXED}, which takes as arguments the verb and the frozen elements of the idiomatic expression (the number of arguments depends on the type of verbal idiom involved) [3].

2. Goal

The main goal of this dissertation project is to use the matrix containing the most recent linguistic description in order to be able to correctly translate it to XIP rules, allowing for it to identify not only manually produced sentences, but also automatically generated ones from the base sentences by applying the transformations authorised by each construction. In order to do so, three essential tasks were considered:

- To rebuild the rule generator so that the generated rules include not only the basic structure of the idiom, but also the several transformations or reduction of certain elements to pronouns that may be applied to each sentence;
- To create a module that automatically generates sentences resulting from applying the aforementioned transformations to the base sentences;
- To create an automatic validator that compares the expected results to the obtained ones, after running both manually produced and automatically generated sentences in STRING. This validates not only the correctness of the generated rules, but also of the generated examples.

3. Document’s Structure

The remainder of this document is structured as follows:

- Section 7 briefly describes the goals of this project;
- Section 4 briefly describes STRING and its components;
- Section 5 briefly describes the related work, namely the representation of the frozen expressions on a lexicon-grammar matrix, saved in an XLSX file, and the previous implementation for automatic rule generation;
- Section 6 presents the issues relative to the previous implementations;
- Section 7 presents the developed solution during this project;
- Section 8 presents the the evaluation methods of the developed solution. The results and their analysis are also briefly presented;
- Section 9 presents the conclusions taken from this project, as well as the perspectives of future work.

4. STRING

STRING [9] is a hybrid statistical and rule-based NLP chain for Portuguese, that has been developed by Spoken Language Systems Laboratory (L^2F), at INESC-ID Lisboa. STRING has a modular structure and performs all the basic NLP tasks. The system’s architecture is shown in Figure 1. Its main components are:

1. **LexMan**: morphological tagger that receives the result of this segmentation as input and associates all possible Part-of-Speech (POS) tags to each segment [13];
2. **RuDriCo**: performs rule-based morphological disambiguation and it also performs segmentation changes to the input, like joining segments (compound words) [5, 6];
3. **MARv**: stochastic morphological disambiguator, and it receives the result of **RuDriCo**2 selecting the best POS tag to each segment;

---

\(^1\)https://typo.uni-konstanz.de/parseme/

\(^2\)For more information on this type of research please refer to [4].
4. **XIP**: finite-state incremental parser developed by XeroxRCE which uses a Portuguese rule-based grammar, and it is responsible for the syntactic analysis.

4.1. XIP

This module shall be briefly described based on the documents [11, 12]. This analyzer allows for the introduction of lexical, syntactic, and semantic information to the output of the previous modules, as well as performing the syntactic analysis of the text, and it is the most important module of STRING for this project. The crucial step of this module is the extraction of Dependency Rules. Dependencies are syntactic dependency relations between different chunks, chunk heads, or elements inside chunks and they allow a deeper and richer knowledge about the text’s information and content. Major dependencies correspond to the so-called deep parsing syntactic functions, such as **SUBJECT**, **DIRECT COMPLEMENT**, etc. Other dependencies are just auxiliary relations, used to calculate the deeper syntactic dependencies. For example, the **CLINK** dependency links each argument of a coordination to the coordinative conjunction it depends.

Verbal idioms are identified by STRING using a dependency **FIXED** linking the key elements of the structure (the main verb and frozen head nouns). The lexicon-grammar of verbal idioms was integrated in rule-based parsing module of the NLP. Since frozen sentences are syntactically well formed structures complying with the general word-combination rules of grammar, the following strategy was adopted to parse them. First, general parsing rules can be applied, as to any other structure. Then, another set of rules extracts the **FIXED** dependency based on the previous parse, and groups together the frozen elements of the idiom, while keeping intact the syntactic structure of the dependency. Finally, **FIXED** dependency is the one used to further calculate the semantic of the sentence [2]. To see a complete list and a detailed description of all syntactic dependency relations as of May 2016, please refer to [3, 4].

Hence, and returning to the dependency rules, the **pattern** contains a TRE that describes the structural properties of parts of the input tree. The **condition** is any Boolean expression supported by XIP (with the appropriate syntax), and the **dependency_terms** are the consequent of the rule.

The first dependency rules to be executed are the ones that establish the dependencies between the nodes, as seen in the next example:

|NP#1?, #2[last] | HEAD(#2, #1) |

This rule identifies **HEAD** relations (see below) in noun phrases. For example, in the NP *a bela rapariga* (‘the beautiful girl’), the rule extracts a **HEAD** dependency between the head noun *rapariga* (‘girl’) and the whole noun phrase — **HEAD**(rapariga, a bela rapariga).

As already stated, the main goal of the dependency rules is to establish dependencies between the nodes. The following output is the current result of applying these rules to the sentence *O Zé bateu em retirada*, lit: ‘Zé beat in retreat’ ‘to run away’:

|MAIN(bateu) DETD(Zé, 0) VDOMAIN(bateu, bateu) |

---

3The Portuguese grammar for XIP was initially developed under the scope of collaboration between L2F and Xerox Research Centre Europe, since 2004 [9, 10].
MOD_POST(bateu, retirada)
SUBJ_PRE(bateu, Zé)
FIXED(bateu, retirada)
NE_PEOPLE_INDIVIDUAL(Zé)
O>TOP{NP{O Zé} VF(bateu) PP{em retirada}}

The last two lines indicate that one named entity
NE has been captured and classified in this sentence: 
Zé has been identified as HUMAN INDIVIDUAL PERSON = PERSON. The tag NE_INDIVIDUAL_HUMAN is used to
see that the NE have been classified. The other de-
dpendencies listed above cover a wide range of binary
dependencies such as:

- The relation between a nominal head and a def-
deinite determiner (DETD);
- The verb (MAIN);
- The relation between a modifier and, in this
case, the verb it modifies (MOD_POST);
- The subject of the verb (SUBJ_PRE);
- The fixed dependency identified between the
verb and the noun (FIXED).

To see a complete list and a detailed descrip-
tion of all syntactic dependency relations as of May
2016, please refer to [7, 8]. XIP’s syntax for these
conditional statements also allows the operators &
for conjunction and | for disjunction. Parentheses
are also used to group statements and establish a
clearer precedence.

4.2. Frozen Sentences
Frozen sentences are elementary sentences where
the main verb and at least one of its argument
noun-phrases are distributionally constraint, and
usually the global meaning of the expression can-
ot be calculated from the individual meaning of
its constituents when used independently. There-
fore, the expression should be taken as a complex,
multiword, lexical unit [1]. 

To this date, a set of 2,561 European Portuguese
verbal idioms has been classified into 15 formal
classes according to their structure and distribu-
tional constraints, as well as their syntactic pro-
PERTIES. Table 1 shows the breakdown of frozen
sentences per class. The theoretical and methodologi-
cal framework of M. Gross [7, 8] was used to classify
this type of expressions. This framework bases its
classification on the structure of the sentence, as
well as on the number and type of arguments of the
main verb [1]. Ten classes are already considerably
developed, but the remaining four are still at a very
early stage. These are the classes CO, COE, CADV and
CV.

5. Related Work
A previous implementation for the automatic
generation of XIP rules was provided on three files.
The main one is named xipificator.pl. It also uses
a file named xipificator_aux_functions.pl and a
xipificator_validate.pl. These contain both the
auxiliary functions necessary for the intermediate
tasks as well as a validator of the generated rules.

5.1. Xipificator
The process starts with converting the XLSX file
to a CSV file, if necessary. After this it proceeds
with searching for the corresponding patterns. A
pattern defines a correspondence between a column
and an element. If none exists or it is marked as
AUTO the system will guess the pattern based on the
names of the filled columns. At last, it writes the
rule and a comment containing an example. The
method for writing a rule is the following:

1. Prints the restriction for the verb;
2. Prints the restriction for the negative form of
the verb;
3. Prints the restriction for the clitic;
4. Searches for the elements of a dependency and
prints their dependency links
5. Finally, a function makes recursive calls until
it reaches the last dependency.
Table 1: Summarized Class Structure, where $N$ represents a free noun phrase, while $C$ is a frozen constituent; the indices 0, 1, 2 and 3 correspond to the subject, first, second, and third complements. $Prep$ is a preposition; $w$ is any sequence of complements (eventually none).

<table>
<thead>
<tr>
<th>Class</th>
<th>Structure</th>
<th>Example</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>$N_0 \ V \ C_1$</td>
<td>O João não abriu a boca ‘be silent’</td>
<td>500</td>
</tr>
<tr>
<td>CDN</td>
<td>$N_0 \ V(C \ of \ N)_1$</td>
<td>O João atraiu os olhares da Ana ‘draw someone’s eye’</td>
<td>44</td>
</tr>
<tr>
<td>CAN</td>
<td>$N_0 \ V(C \ of \ N)_1 = C_1 \ to \ N_2$</td>
<td>O João calou a boca da Ana ‘shut up someone’</td>
<td>182</td>
</tr>
<tr>
<td>CNP2</td>
<td>$N_0 \ V \ N_1 \ Prep_2 \ C_2$</td>
<td>O Rui chamou a Inês à razão ‘call to reason’</td>
<td>172</td>
</tr>
<tr>
<td>C1PN</td>
<td>$N_0 \ V \ C_1 \ Prep_2 \ N_2$</td>
<td>A Maria desligou os aparelhos ao moribundo ‘switch off the machines’</td>
<td>255</td>
</tr>
<tr>
<td>C1P2</td>
<td>$N_0 \ V \ C_1 \ Prep \ C_2$</td>
<td>O João retomou o fio à meada ‘resume the thread’</td>
<td>291</td>
</tr>
<tr>
<td>CPPN</td>
<td>$N_0 \ V \ C_1 \ Prep_2 \ C_2 \ Prep_3$</td>
<td>O João vendeu gato por lebre à Maria ‘sell cat for hare’</td>
<td>46</td>
</tr>
<tr>
<td>CPP</td>
<td>$N_0 \ V \ Prep_1 \ C_1 \ Prep_2$</td>
<td>O Zé não morre de amores pela Ana ‘is not fond of’</td>
<td>181</td>
</tr>
<tr>
<td>CP1</td>
<td>$N_0 \ V \ Prep_1 \ C_1$</td>
<td>O Zé voltou à carga ‘charge again onto something’</td>
<td>662</td>
</tr>
<tr>
<td>CPN</td>
<td>$N_0 \ V \ Prep(C \ of \ N)_1$</td>
<td>O Zé caiu nas garras da Ana ‘fall in the claws of’</td>
<td>103</td>
</tr>
<tr>
<td>C0</td>
<td>$C_0 \ V \ w$</td>
<td>A sorte bafejou o Pedro ‘luck blown over someone’</td>
<td>21</td>
</tr>
<tr>
<td>C0E</td>
<td>$V \ w$</td>
<td>Vai pentear macacos! ‘go comb monkeys’, ‘get lost’</td>
<td>1</td>
</tr>
<tr>
<td>CADV</td>
<td>$N_0 \ V \ Adv$</td>
<td>O Pedro não nasceu ontem ‘was not born yesterday’</td>
<td>70</td>
</tr>
<tr>
<td>CV</td>
<td>$N_0 \ V(Prep) \ V_c \ w$</td>
<td>A resposta não se fez esperar ‘did not have to wait much for something’</td>
<td>13</td>
</tr>
</tbody>
</table>

Total 2,542
Finally, the system will try to extract the fixed dependency using the restrictions obtained, from a set of 2,522 sentences. This system does not predict, in its generated rules, the existence of sentences containing transformations.

5.2. Validator

There was also a module responsible for validating the values in the matrix, after this is converted to a CSV file, given how important it is that this conversion does not fail, and represents correctly the description in the XLSX file. The data validation is done by asserting which are the possible values for each column, using the following structure: [Column name, Validation Type, Possible Values]. The following validations are performed:

1. Verifying the consistency between each element. This means to verify whether the values of each column respect the properties of each sentence and, e.g. whether the values are consistent or inconsistent for nouns and prepositions;

2. The class validation is done by checking whether each class contains the expected values for each column, and respects the restrictions imposed on the elements of that class, including the symmetry property;

3. Checking the consistency between column values; validation of whether all the restrictions are being respected and there are no impossible combinations.

This module was not changed and it is being used in the current system, in order to validate the matrix.

6. Issues with the Previous Implementation

Given that the sentences are described in a matrix, the system was built around a static number of columns and attributes. After it was built, the matrix suffered a number of changes, including the addition of transformations, which had to be added to the rules. These changes caused the system to stop being able to generate XIP rules. Besides, neither transformations nor pronominalizations had been predicted on the previous system. These required adopting a new strategy for generating rules to recognize expressions containing elements that have undergone these formal changes. However, there are almost no manually generated sentences that contain transformations.

The way to deal with problems in this system was to insert a manually generated rule in matrix, which describes patterns that cannot be automatically generated by the system.

Another problem is that the current system is running one sentence at a time, initializing the system each time. This results in a big delay in obtaining results. Finally, in this implementation, the only validation method being used by the system is to verify whether the fixed dependency has been extracted. No further verification concerning the arguments of the dependency is done, and these may not be correct.

7. Solution

The main goal of this project is to use the matrix containing the most recent linguistic description and to be able to correctly translate it to XIP rules, allowing for the system to identify not only manually produced sentences but also other sentences, automatically from the examples encoded in the matrix by applying the transformations authorized by each construction. In order to do so, the rule generator was rebuilt so that the generated rules capture not only the basic structure of the idiom, but also the several transformations or reduction of certain elements to pronouns that may be applied to each sentence. A module was created for generating examples in an automatic way after several transformations have been applied to the manually produced examples provided in the matrix, such as pronominalization and the passive forms. This allows for the generation of rules that will also recognize them as frozen sentences. Finally, an automatic validator was developed. This validator receives as input the results of processing all the sentences, both those manually produced and those artificially generated, and compares them against what was the expected result. The main differences between the previous and current system are represented in Figure 2, where the green modules are the ones that were created from scratch, and the orange modules are the ones that were restructured or suffered some modification. The blue ones were left untouched, but they were integrated in the system. This chapter will start by describing the structure of the lexicon-syntactic matrix. Following this, the architecture and implementation of the new modules and the changes introduced on the existing ones will be detailed.

7.1. Lexicon-Syntactic Matrix

To develop this project, a set of 2,561 European Portuguese verbal idioms was used. This set was structured in 15 formal classes according to their structure and distributional constraints. These are described in a lexicon-syntactic matrix, a XLXS file, which will be used for both rule and example generation. The version of the lexicon-syntactic description used in this project is version 13 from April 2019. This version was meanwhile updated with corrections for the problems found during the development of this work.
The matrix file starts with a header - containing the name of the column - and it is followed by a set of properties for identifying a specific frozen sentence, one sentence per line. Each column contains an element of the rule, or a restriction to it. During the development of this work, several column values were either incoherent or even incorrect. Some of these problems were detected while generating the rules, while others were detected using the previously existing validator of the matrix. One important thing in this implementation is that, by pre-defining names for each column the program is able to automatically identify its pattern, which allows for the columns to appear in any order. By omission, the system contains the following features:

- **MOD**: modifier
- **CDIR**: direct complement
- **CIND**: indirect complement
- **PREDSUBJ**: subject’s predicate
- **PFX**: prefixed word

### 7.2. Rule Generation

This module receives as input the CSV file and outputs a file containing the XIP rules generated from that CSV, as well as a TXT file containing each sentence, the class it belongs to, and its expected output. This will later be used by the example validator. The CSV file is read into the module, which creates an internal structured representation of each line of the CSV, and its corresponding values. This allows the information to be encapsulated inside the program so that it is no longer necessary to access external files.

The process of generating the rules is complex, given that each line from the matrix is associated to a corresponding XIP rule, and each possible column value contributes with a restriction to that rule. The module starts with writing the example corresponding to the read line. Then, it verifies whether the rule is manually produced. If it is, the rule is read from the column `Manual`. If not, the rule for that sentence is generated according to the values of the lexicon-syntactic matrix. In case any transformation can be applied to that sentence, the restrictions associated with that transformation are added to the rule. If no transformation is applicable, it writes the rule and the expected value to be extracted from XIP for that sentence. Below, an example of the step-by-step generation process of a rule is presented, for the frozen sentence *O João virou o bico ao prego*, lit: ‘João turn the tip to the nail’, ‘to betray’ (class C1P2), which is depicted with its constituents in Figure 3.

![Figure 2: Comparing the two systems: orange represents what was re-written, green what was added.](image)

![Figure 3: A frozen sentence and the heads of its constituents.](image)
1. The \( \text{\texttt{V}} \) column is converted to the restriction as \\
\( \text{VDOMAIN}(\#?,\#2[\text{lemma:virar}]) \); \\
2. The first complement, column \( \text{\texttt{C1}} \) is converted into a restriction is encoded as \( \text{CDIR}[\text{post}] \) (the post flag refers to the post-verbal position) because there is no preposition associated to this complement: \( \text{CDIR}[\text{post}] (\#2,\#3[\text{surface:bico}]) \); So the XIP rule evolves into: \\
if ( \text{VDOMAIN}(\#?,\#2[\text{lemma:virar}]) & \\
\text{CDIR}[\text{post}](\#2,\#3[\text{surface:bico}]) & \\
... 
)
3. The next restriction to be encoded is \( \text{\texttt{Det1}} \), which produces the restriction \\
\( \text{DETD}(\#3,\#?[\text{surface:o}]) \). The rule evolves to: \\
if ( \text{VDOMAIN}(\#?,\#2[\text{lemma:virar}]) & \\
\text{CDIR}[\text{post}](\#2,\#3[\text{surface:bico}]) & \\
\text{DETD}(\#3,\#?[\text{surface:o}]) & \\
... 
)
4. The second complement, \( \text{\texttt{C2}} \), is then translated as \( \text{MOD}[\text{post}](\#2,\#4[\text{surface:prego}]) \). It is important to notice that its head is connected to the verb, instead of the previous complement, which is explicitly marked in the matrix by the property \( \text{AttachV} \). \\
if ( \text{VDOMAIN}(\#?,\#2[\text{lemma:virar}]) & \\
\text{CDIR}[\text{post}](\#2,\#3[\text{surface:bico}]) & \\
\text{DETD}(\#3,\#?[\text{surface:o}]) & \\
\text{MOD}[\text{post}](\#2,\#4[\text{surface:prego}]) & \\
\text{PREPD}(\#4,\#?[\text{surface:a}]) & \\
\text{DETD}(\#4,\#?[\text{surface:o}]) 
)
5. Next, \( \text{\texttt{Prep2}} \) is encoded as \\
\( \text{PREPD}(\#4,\#?([\text{surface:a}]) \). The XIP rule now becomes: \\
if ( \text{VDOMAIN}(\#?,\#2[\text{lemma:virar}]) & \\
\text{CDIR}[\text{post}](\#2,\#3[\text{surface:bico}]) & \\
\text{DETD}(\#3,\#?([\text{surface:o}]) & \\
\text{MOD}[\text{post}](\#2,\#4[\text{surface:prego}]) & \\
\text{PREPD}(\#4,\#?([\text{surface:a}]) & \\
... 
)
6. Finally, the last column to be encoded as a restriction is \( \text{\texttt{Det2}} \), producing \\
\( \text{DETD}(\#4,\#?([\text{surface:o}]) \). This results in the rule: \\
if ( \text{VDOMAIN}(\#?,\#2[\text{lemma:virar}]) & \\
\text{CDIR}[\text{post}](\#2,\#3[\text{surface:bico}]) & \\
\text{DETD}(\#3,\#?([\text{surface:o}]) & \\
\text{MOD}[\text{post}](\#2,\#4[\text{surface:prego}]) & \\
\text{PREPD}(\#4,\#?([\text{surface:a}]) & \\
\text{DETD}(\#4,\#?([\text{surface:o}]) 

Finally, to allow for an easier reading and correction the rule is represented in the rules’ file as:

```
//========================================================
// Example: O João virou o bico ao prego
//========================================================
if ( VDOMAIN(#?,#2[lemma:virar]) &
CDIR[post](#2,#3[surface:bico]) &
MOD[post](#2,#4[surface:prego]) &
PREPD(#4,#?([surface:a]) &
DETD(#4,#?([surface:o])
)
``` 

The following step is to integrate the rules in the XIP dependencies file. When running the aforementioned example on STRING, each variable of the rule will then be instantiated according to performed analysis of the elements of the sentence, obtaining the following dependency rules:

\begin{align*}
\text{VDOMAIN}(& \text{virou, virou}) \\
\text{CDIR}[\text{post}](& \text{virou, bico}) \\
\text{DETD}(& \text{bico, o}) \\
\text{MOD}[\text{post}](& \text{virou, prego}) \\
\text{PREPD}(& \text{prego, a}) \\
\text{DETD}(& \text{prego, o}) \\
\end{align*}

These dependency rules will then be compared against those found in the output provided by XIP:

\begin{align*}
\text{MAIN}(& \text{virou}) \\
\text{DETD}(& \text{João, 0}) \\
\text{DETD}(& \text{bico, o}) \\
\text{DETD}(& \text{prego, o}) \\
\text{VDOMAIN}(& \text{virou, virou}) \\
\text{MOD}_\text{POST}(& \text{virou, prego}) \\
\text{SUBJ}_\text{PRE}(& \text{virou, João}) \\
\text{CDIR}_\text{POST}(& \text{virou, bico}) \\
\end{align*}

Given that the elements of the generated rule are present in the output, and therefore the restrictions are satisfied, the \( \text{FIXED} \) dependency is extracted, as \\
\( \text{FIXED(virou,bico,prego)} \)

Whenever a transformation may be applied to a sentence, two things happen:

- The Example Generation module will automatically generate a sentence containing the transformation(s)
The restrictions relative to that transformations are added to the rule or, in case the transformation is either [Pass-ser] or [Pass-estar], no restrictions are added to the base sentence rule and, instead, a new rule is generated for the sentence after the transformation was performed.

The passive form is a special case, being the only type of transformation that generates a new rule. First, the verb itself has to be encoded in a different way. Then, a conversion of the constituents from the base sentence to the passive form is also performed. Whenever a sentence contains either a direct complement or a post modifier, it becomes a subject in the new rule, generated to represent the passive form of that sentence. Using the example O Rui deixou a Inês em paz, lit: ‘Rui left Inês in peace’ ‘to leave someone alone’, Inês plays the role of direct complement, CDIR[post]{#2,#3[UMB-Human,UMB-Human:~]}. to this construction. However, when transforming it to the passive form, this will become the subject SUBJ(#2,?). This conversion is performed by a table that corresponds what each element of the active form should be in the passive form containing, for now, only the elements CDIR and MOD[post].

A configuration file has been created in order to make it possible to determine which restrictions are to be applied to the generated rule. The controllable restrictions are determinants, prepositions and modifiers, both to the left and to the right of the frozen head noun, and any distributional constraints to any of the free complements/subject.

The generation starts with verifying, for each sentence, whether its description contains a positive value for each column [PronA1], [PronA2], [PronPos2], [Rdat1], [Rdat2], [PronD1], [PronD2], [Pass-estar] or [Pass-ser]. If so, the system will read, from the description of that sentence, each of its constituents. Using this, it generates a new sentence (one sentence per transformation), containing the mandatory complements after the changed required by each transformation have been applied to each. Although, in an initial phase, these new sentences were written alongside the corresponding base sentence in a text file, it was later decided that they should be written into separate files, according to the type of transformation applied to them. This allowed for easier manual verification and validation of the obtained sentences for each transformation. They are run on STRING and later validated separately from the base sentences, which allows for a clearer distinction of the system’s performance for each type of sentence, manually produced or automatically generated.

7.4. Example Validation

This example validator receives as input the output that was generated by STRING, written in an XML file. From here it extracts what was effectively obtained, and builds a text file with this information: the sentence, what was expected, and the obtained result, represented in Figure 2 as ‘output.txt’. Given that the rule generator already provides the system with what is expected, the next step is to compare the two: what was expected and what was acquired.

The validator considers 3 criteria in order to evaluate a success and “how much” of a success the detection had. This criteria range from the less specific to the more specific, in the following order:

1. Checking whether the FIXED dependency was extracted;
2. Checking whether the number of arguments of that dependency matches the expected number of arguments;
3. Asserting that the arguments of that dependency match the expected arguments.

The results of processing each sentence through STRING is an XML file with each sentence represented as an LUNIT. Each of these LUNITs will be parsed through until the FIXED dependency is found. When it is found, the arguments of this dependency...
are parsed. The element with index 0 is the verb, and all the other indexes correspond to the remaining constituents that are part of that dependency. The obtained dependency \texttt{FIXED} is then built in reverse, from what was obtained - given that it is from the output - until something such as \texttt{FIXED(0, 1, 2, 3...)} is obtained (with 0 being the verb, 1, 2 and 3 the remaining arguments). In case no \texttt{FIXED} dependency is extracted, the parser returns "FAILED".

After the example validation is concluded, an output report is generated by the system, containing a sentence, its expected value and its result value.

8. Evaluation
The evaluation was performed according to the aforementioned three criteria, in Section 7: checking whether the \texttt{FIXED} dependency was extracted; checking whether the number of arguments of that dependency matches the expected number of arguments; asserting that the arguments of that dependency match the expected arguments. One important note to be made is that it is crucial to interpret these results according to their lexical representativity; that is, the number of expressions in the lexicon. Bearing this in mind, and because the class representativity is not the same for all classes, the total result is not calculated as an average of the results of the classes. Instead, it takes the form of an intrinsic evaluation, the result of measuring the recall:

\[
    \text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegatives}}
\]

where TruePositive represents the number of dependencies correctly extracted and FalseNegatives represent the dependencies that were not extracted but should have been. One important highlight is that there are no FalseNegatives because all the sentences are assumed to be correct. For the generated sentences that implies performing a manual verification.

Table 2: Number of artificially generated sentences identified as frozen according to the defined criteria.

<table>
<thead>
<tr>
<th>Criteria</th>
<th># Developed system</th>
<th>% Developed system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed dependency</td>
<td>2,430</td>
<td>96%</td>
</tr>
<tr>
<td>Same number of arguments</td>
<td>2,401</td>
<td>94%</td>
</tr>
<tr>
<td>Exact arguments</td>
<td>2,337</td>
<td>92%</td>
</tr>
</tbody>
</table>

Table 3: Number of artificially generated sentences identified as frozen according to the defined criteria.

<table>
<thead>
<tr>
<th>Criteria</th>
<th># Developed system</th>
<th>% Developed system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed dependency</td>
<td>1095</td>
<td>93%</td>
</tr>
<tr>
<td>Same number of arguments</td>
<td>1090</td>
<td>93%</td>
</tr>
<tr>
<td>Exact arguments</td>
<td>1090</td>
<td>93%</td>
</tr>
</tbody>
</table>

Table 4: Number of sentences (manually and artificially generated) identified as frozen according to the defined criteria.

<table>
<thead>
<tr>
<th>Criteria</th>
<th># Developed system</th>
<th>% Developed system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed dependency</td>
<td>3,525</td>
<td>95%</td>
</tr>
<tr>
<td>Same number of arguments</td>
<td>3,491</td>
<td>94%</td>
</tr>
<tr>
<td>Exact arguments</td>
<td>3,427</td>
<td>92%</td>
</tr>
</tbody>
</table>

By observing the Tables 2, 3 and 4, it is possible to conclude that the system is very much successful in detecting frozen sentences, both manually produced and artificially generated. The development of the rule generation was done through several iterations. The result of each iteration required manual validation of the rules. Most problems that were detected were solved, and there are, currently, no known problems inherent to the rule generation process. However, there is always room for improvement, which will require case-by-case investigation and manual validation of each rule that failed to extract the dependency \texttt{FIXED}.

9. Concluding Remarks
This project aimed at improving the processing of frozen sentences, that is, multword verbal idioms, in the STRING system. The XIP module, responsible for detecting them, uses rules created by an existing system, which presented some fragilities, particularly when detecting sentences resulting from transformations of the sentences’ base form. This work contributed to improving this detection in the following manner:

- A new module that automatically generates sentences resulting from applying to them a set of transformations;
- The rule generator was re-written in order to accommodate the transformations that can be applied to the sentences;
• A new module was built, which automatically validates the output of the examples, comparing them against what was expected.

Generally speaking this work contributed to improving the overall performance of the STRING system. It did so by greatly improving the detection of sentences with transformations, as well as introducing a more thorough way of evaluating every sentence.

Another factor that was improved is the system’s speed. The main contributor to this is the pipeline script that automated the process of generating the rules, integrating them into the STRING system and validating the results of running both manually and artificially generated sentences. Finally, some errors were detected in the matrix while developing the rule generation. Therefore this work also helped improve the consistency of the lexicon-grammar, clarifying the meaning of some properties there encoded, as well as validating the values present there.

References


