Appendix A
Extended List of Translation Examples

The list of automatic translations (MT) in French (fr) were generated by Google Translate[1] on 2014/04/15. Multiword expressions wrongly translated are in boldface. Some sentences contain more than one MWE, but only one is highlighted to increase readability. Reference sentences were manually translated by native speakers.

en source	We only go out **once in a blue moon**
fr MT	Nous allons seulement **une fois dans une lune bleue**
fr reference	Nous sortons **tous les 36 du mois**
en source	The **dry run** went on smoothly
fr MT	Le **fonctionnement à sec** est allé en douceur
fr reference	La **répétition** s'est bien passée
en source	They **carry** my project **through** despite the crisis
fr MT	Ils **portent** mon projet **à travers** malgré la crise
fr reference	Ils **maintiennent** mon projet malgré la crise
en source	The children **ate** my cookies **up**
fr MT	Les enfants **ont mangé** mes biscuits **jusqu'à**
fr reference	Les enfants **ont mangé tous** mes biscuits
en source	My friend is always **chasing rainbows**
fr MT	Mon ami est toujours **Chasing Rainbows**
fr reference	Mon ami **fait** continuellement **des plans sur la comète**

[1] http://translate.google.com/

C. Ramisch, *Multiword Expressions Acquisition*, Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-319-09207-2

en source	The boy **is in** his mother's **black books**
fr MT	Le garçon **est dans les livres noirs** de sa mère
fr reference	Le garçon **n'est pas dans les petits papiers** de sa mère
en source	MWEs are a **pain in the neck** for computers
fr MT	MWEs sont une **douleur dans le cou** pour les ordinateurs
fr reference	Les EPL sont comme une **épine dans le pied** pour les ordinateurs
en source	MWEs are a **tough nut to crack**
fr MT	MWEs sont un **dur à cuire**
fr reference	Les EPL sont **un vrai casse-tête**
en source	I never get on **cable cars**
fr MT	Je ne suis jamais sur les **voitures de câble**
fr reference	Je ne monte jamais dans les **téléphériques**
en source	They **brought about** the boat
fr MT	Ils ont **apporté sur** le bateau
fr reference	Ils ont **fait demi-tour** au bateau
en source	He rarely **gets drunk** at work
fr MT	Il **se fait** rarement **bu** au travail
fr reference	Il **est** rarement **ivre** au travail
en source	The **workshop proceedings** are online
fr MT	Les **travaux de l'atelier** sont en ligne
fr reference	Les **actes de l'atelier** sont en ligne
en source	We can **count** Poland **in**
fr MT	Nous pouvons **compter** Pologne
fr reference	Nous pouvons **compter sur** la Pologne

Appendix B
Resources Used in the Experiments

B.1 Data

B.1.1 Monolingual Corpora

- **English**
  - *British National Corpus (BNC)*. The BNC is a general-purpose corpus of British English, containing around 100 million words, mixing several genres like literature and newspapers It is one of the most popular corpora in NLP for English. It is annotated with POS.
  - *Genia corpus*. It is composed of a set of 2,000 English abstracts of scientific articles from the biomedical domain. It contains around 18K sentences and around 490K tokens. The corpus contains information about sentence and word boundaries, POS tags and terminological annotation with respect to the Genia ontology.
- **Portuguese**
  - *PLNBR-FULL corpus*. This corpus was built in the context of the PLNBR project (www.nilc.icmc.usp.br/plnbr). It contains 29,014,089 tokens of news text from *Folha de São Paulo*, a Brazilian newspaper, from 1994 to 2005. It can be considered as a general-purpose corpus of Brazilian Portuguese.

B.1.2 Multilingual Corpora

- *Europarl corpus (EP)* The EP corpus contains transcriptions of the sessions of the European Parliament. It contains around 50 million words of parallel text in 11 languages of the European Union, including Portugues, English and French, plus around 10 million words for other languages of countries that recently joined the European Union. It can be viewed as a general-purpose corpus as it runs over more than 10 years and the political debates have a wide range of discussion subjects. It is one of the most popular resources for SMT. We used two versions: the older one, v3, consists of extracts from the proceedings of the European Parliament during the period Apr/1996–Oct/2006; and the more recent version, v6, contains the same texts as in v3 plus recent transcriptions up to Dec/2010. EP is publicly available at http://www.statmt.org/europarl/.

- *The web as a corpus*. The web contains a large amount of textual data in several languages. It can be used to overcome data sparseness in traditional corpora. It is not a parallel corpus, but comparable corpora may be extracted from the web. It can also be thought of as a set containing several monolingual corpora, each one with millions of words. It is practically impossible to crawl and download all the ever-growing text of the web, but search engines can be used to estimate the counts of words in the web. We use Google and Yahoo search APIs and the implementation of the `mwetoolkit`.
- *TED talks corpus*. It contains transcriptions of the TED conferences, covering a great variety of topics. Talks are given in English and translated by volunteers worldwide into many languages. We used the English-French portion, which contains 141 K sentences and 2.5 M words in each language. The corpus is freely available at the Web Inventory of Transcribed and Translated Talks http://wit3.fbk.edu.

B.2 Software

B.2.1 Analysis Tools

- *Europarl corpus tools*. The EP corpus comes with some scripts for text tokenisation, sentence splitting and sentence alignment. These were used in some experiments.
- *TreeTagger*. The TreeTagger is a free downloadable POS tagger available for several languages, and with a good performance for English. It performs not only POS tagging but also sentence splitting, tokenisation and lemmatisation of the text. The TreeTagger is freely available at http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/. The tagset used by the TreeTagger in English is available at ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz.
- *PALAVRAS parser*. This deep syntactic parsing tool of Portuguese was used for the analysis of Portuguese text It supports tokenisation, sentence splitting, POS tagging, lemmatisation, dependency parsing annotation and shallow semantic annotation. In most cases, only the first four features were used.
- *RASP parser*. The RASP parser is a free downloadable tool for the syntactic analysis of English text. It provides not only POS tagging but also constituent and dependency trees. It is available at http://www.informatics.susx.ac.uk/research/groups/nlp/rasp/.

Appendix C
The `mwetoolkit`: Documentation

This appendix contains a snapshot of the `mwetoolkit` documentation. However, it is preferred to consult the latest and up-to-date documentation available on the website http://mwetoolkit.sf.net. This documentation was produced with the help of the people cited in Sect. C.8.

C.1 Design Choices

The `mwetoolkit` manipulates intermediary candidate lists and related elements as XML files. The use of XML as intermediary format has the advantage that it is readable and easy to validate according to a *document type definition* (DTD). It is also easy to import and export XML documents from and to other tools, as we describe in the next section. However, in terms of computational performance, the choice of an interpreted programming language like Python combined with a verbose file format like XML made some modules very slow and/or memory-consuming, requiring some optimisations. For example, the first versions of the indexing and candidate generation scripts were not able to deal with large corpora such as Europarl and the BNC. Therefore, some parts of the `mwetoolkit` were re-implemented in C. With the C indexing routine, for instance, indexing the BNC corpus takes about 5 min per attribute on a 3 GB RAM computer.

In the implementation, instead of using the XML corpus and external matching procedures, we match candidates using Python's built-in regular expressions directly on the corpus index. This avoids parsing a huge XML file and speeds up pattern matching. On a small corpus, the current implementation takes about 72 % the original time (using the XML file) to perform pattern-based extraction.

Our target users are researchers with a background in computational linguistics and with some experience using command-line tools. The method is not a push-button utility that acquires any type of MWE from any type of corpus: it requires some manual tuning, pattern definition and parameter tuning. In sum, some trial and error iterations are needed in order to obtain the desired output.

Although no graphical user interface is available, we developed a "friendlier" command line interface. In the original version, one needed to manually invoke the Python scripts passing the correct options. The current version provides an interactive command-based interface which allows simple commands to be run on data files, while keeping the generation of intermediary files and the pipeline between the different phases of MWE extraction implicit. At the end, a user may want to save the session and restart the work later. Although it is not a graphical interface it is far easier to use than previous versions. In the future, we would like to develop a graphical interface, so that the toolkit can be used by researchers who are not at ease with the command line.

The `mwetoolkit` is a downloadable, freely available and open-source set of scripts. However, for more up-to-date documentation, as well as for downloading and testing the tool, one should prefer the official project website hosted at http://mwetoolkit.sourceforge.net/.

C.2 Installing the `mwetoolkit`

C.2.1 Windows

Unfortunately, there is *NO WINDOWS VERSION AVAILABLE* of the `mwetoolkit` for the time being.

C.2.2 Linux and Mac OS

To install the `mwetoolkit`, just download it from the SVN repository using the following command:

```
svn co svn://svn.code.sf.net/p/mwetoolkit/code/
mwetoolkit
```

Once you have downloaded (and unzipped, in the case of a release) the toolkit, navigate to the main folder and run the command

```
make
```

for compiling the C libraries used by the toolkit. Do not worry about the warnings, they are normal. If you do not run this command, or if the command fails to compile the library, the toolkit will still work but it will use a Python version (much slower and possibly obsolete!) of the indexing and counting scripts. This may be OK for small corpora.

C.2.3 *Mac OS Dependencies*

In addition to `mwetoolkit` itself, you will need to download and to configure some specific libraries.

C.2.3.1 Coreutils Package (Through MacPorts)

To get this done is pretty simple, once you have MacPorts set up correctly (you can type man port and get a manual page), just run the following command:

```
sudo port install coreutils
```

If you don't have MacPorts yet, install it from http://www.macports.org/install.php/.

C.2.3.2 Simplejson (Python)

The Python installation comes with a handy utility called easy_install, which easily installs missing components: `sudo easy_install simplejson`

C.2.4 *Testing Your Installation*

The test folder contains regression tests for most scripts. In order to test your installation of the `mwetoolkit`, navigate to this folder and then call the script `testAll.sh`:

```
cd test ./testAll.sh
```

Should one of the tests fail, please send a copy of the output and a brief description of your configurations (operating system, version, machine) to our gmail, our username is `mwetoolkit`.

C.3 Getting Started

`mwetoolkit` works by extracting MWE candidates from a corpus using a set of morphosyntactic patterns. Then it can apply a number of statistics to filter the extracted candidates. Input corpora, patterns and candidates are stored as XML files, following the format described by the DTDs in the `dtd` directory in the distribution. The toolkit consists of a set of scripts performing each phase of candidate extraction and analysis; these scripts are in the `bin` directory.

mwetoolkit receives as input a corpus as a XML file. This file contains a list of the sentences of the corpus. Each sentence is a list of words, and each word has a set of attributes (surface form, lemma, part of speech, and syntax information, if available). To obtain this information from a plain textual corpus without annotations, usually a part-of-speech tagger is used, which takes care of separating the input in tokens (words) and assigning a part-of-speech tag to each word.

To obtain a XML corpus from a plain textual corpus, you can use a tagger program or parser, such as explained in Sect. C.5 and in Sect. C.6.

C.3.1 An Example

The toolkit comes with example files for a toy experiment in the directory `toy/genia`:

- `corpus.xml`—A small subset of the Genia corpus.
- `patterns.xml`—A set of patterns for matching noun compounds.
- `reference.xml`—A MWE reference (gold standard) for comparing the results of the candidate extraction against.

This directory also contains a script, `testAll.sh`, which runs a number of scripts on the example files. For each script run, it displays the action performed and the full command line used to run the script. It creates an output directory where it places the output files of each command.

Let's analyse each command that is run by `testAll.sh`. First, it runs `index.py` to generate an index for the corpus. This index contains suffix arrays for each word attribute in the corpus (lemma, surface form, part-of-speech, syntax annotation), which are used to search for and count the occurrences of an *n*-gram in the corpus. The full command executed is `index.py -v -i index/corpus corpus.xml`. The option `-i index/corpus` tells the script to use `index/corpus` as the prefix pathname for all index files (the `index` folder must exist). The `-v` option tells it to run in verbose mode (this is valid for all scripts).

After generating the index for the Genia fragment, it performs a candidate extraction by running:

```
candidates.py -p patterns.xml -i index/corpus >
candidates.xml
```

This invokes the candidate extraction script, telling it to use the patterns described in the file `patterns.xml`, and to use the corpus contained in the index files whose prefix is `corpus` (this is the same name given to the `index.py` script). Instead of using a patterns file, you could specify the `-n min:max` option to extract all *n*-grams with size between `min` and `max`.

Once candidates have been extracted, the counts of the individual words in each candidates are computed with the command:

```
counter.py -i index/corpus candidates.xml >
candidates-counted.xml
```

These counts are used by other scripts to compute statistics on the candidates. Word frequency cannot be computed directly from the XML file (it is done through binary search on the index). Instead of a corpus, you can count estimated word frequencies from the web, using either the Yahoo (option `-y`—DEPRECATED) or Google (option `-w`) search engine. You can also count word frequencies from an indexed corpus different from the one used for the extraction.

After word frequencies have been counted, association measures are calculated with the command:

```
feat_association.py -m mle:pmi:ll:t:dice
candidates-counted.xml >candidates-featureful.xml
```

The `-m measures` option is a colon-separated list specifying which measures are to be computed: Maximum Likelihood Estimator (`mle`), Pointwise Mutual Information (`pmi`), Student's t test score (`t`), Dice's Coefficient (`dice`), and Log-likelihood (`ll`, for bigrams only).

The association measures can be used in several ways. Here, we simply chose an association measure that we consider good, the t score, and sort the candidates according to this score, with the command:

```
sort.py -f t_corpus candidates-featureful.xml >
candidates-sorted.xml
```

The next script then works as Linux `head` command, cropping the sorted file and keeping only candidates with higher t score values. Finally, we compare the resulting candidates with a reference list containing some expressions that are already in a dictionary for the Genia biomedical domain. This is quite standard in MWE extraction, even though it only gives you an underestimation of the quality of the candidates as dictionaries are not complete. The command used in the evaluation is:

```
eval_automatic.py -r reference.xml -g
candidates-crop.xml > eval.xml 2> eval-stats.txt
```

The `-g` option tells the script to ignore parts of speech while the `-r` option indicates the file containing the reference gold standard in XML format. The final figures of precision and recall is in file `eval-stats.txt`. Remember that this is only a toy experiment and that with such a small corpus, the association measures cannot be trusted

For more advanced options, you can call the scripts using the `--help` option. This will print a message telling what the script does, what are the mandatory arguments and optional parameters. If you still have questions, write to our gmail address, username `mwetoolkit`, and we'll be happy to help!

C.4 Defining Patterns for Extraction

`mwetoolit` extracts MWE candidates by matching each sentence in the corpus against a set of patterns specified by the user. These patterns are read from XML files. This section describes the format of such files.

The root element of the XML patterns file is `<patterns>`. Inside this element comes a list of patterns, introduced by the tag `<pat>`. The `candidates.py` script will try to match each sentence of the corpus against each pattern listed:

```
<patterns>
    <pat>...</pat>
    <pat>...</pat>
    ...
</patterns>
```

C.4.1 Literal Matches

The simplest kind of pattern is one that matches literal occurrences of one or more attributes in the corpus. This is done with the tag `<w attribute="value" ... />`. For example, to match an adjective followed by a noun, one could use the pattern[1]:

```
<pat>
    <w pos="J" />
    <w pos="N" />
</pat>
```

C.4.2 Repetitions and Optional Elements

It is possible to define regular-expression-like patterns, containing elements that can appear a variable number of times. This is done with the `repeat` attribute of the `pat` tag and with the `either` element. Note that `pat` elements can be nested.

```
<patterns>
    <!-- Pattern for matching a simple noun phrase. -->
    <pat>
        <!-- optional determiner (appearing 0 or
             1 times) -->
        <pat repeat="?"><w pos="DT" /></pat>
```

[1]The actual part-of-speech tags depends on the convention used to tag the corpus, of course. Some tagging tools tag nouns with SUBST or NN, for instance.

```
        <!-- any number (including zero)
             of adjectives -->
        <pat repeat="*"><w pos="J" /></pat>
        <!-- one or more nouns -->
        <pat repeat="+"><w pos="N" /></pat>
    </pat>

    <pat>
        <!-- 3 to 5 adjectives -->
        <pat repeat="{3,5}"><w pos="J" /></pat>
        <!-- followed by the noun "dog" -->
        <w pos="N" lemma="dog" />
    </pat>

    <!-- A sequence of nouns or adjectives
         followed by a final noun -->
    <pat>
        <pat repeat="*">
        <either>
            <pat>
                <w pos="N"/>
            </pat>
            <pat>
                <w pos="J"/>
            </pat>
        </either>
        </pat>
        <w pos="N"/>
    </pat>
</patterns>
```

C.4.3 Ignoring Parts of the Match

You can discard parts of a match by specifying an `ignore` attribute to the `<pat>` element:

```
<pat>
    <!-- Match a determiner, followed by any number
         of adjectives, followed by a noun. The
         adjectives are discarded from the match. -->
    <w pos="DT" />
    <pat repeat="*" ignore="true"><w pos="J" /></pat>
    <w pos="N" />
</pat>
```

C.4.4 Backpatterns

It is possible to create patterns with backreferences. For instance, you can match a word that has the same lemma as a previously matched word. To do this, you assign an `id` to the first word, and use `back:id.attribute` as the value of an attribute in a subsequent word:

```
<pat>
    <!-- Match N1-prep-N1 compounds (e.g.,
         step by step, day after day) -->
    <!-- Match a noun, labeled n1 -->
    <w pos="N" id="n1" />
    <!-- Match a preposition -->
    <w pos="P" />
    <!-- Match a noun whose lemma is the same as
         the lemma of n1 -->
    <w pos="N" lemma="back:n1.lemma" />
</pat>
```

Previous versions of the toolkit used `<backw lemma="n1" />` instead of `<w lemma="back:n1.lemma" />`. There was no way of specifying both a literal attribute and a backreference with the old syntax.

C.4.5 Syntactic Patterns

The toolkit supports corpora with syntactic annotations: the `<w>` element can contain a `syn` attribute, which contains a list of the syntactic dependencies of the word in the sentence, in the format `deptype1:wordnum1;deptype2:wordnum2;...`, where `deptypen` is the type of the dependency, and `wordnumn` is the number of the word that is the target of the dependency (first word is 1). For example, `<w lemma="book" pos="N" syn="dobj:4" />` in the corpus represents a noun, book, which is the direct object of the fourth word in the sentence. (Again, the syntactic tag will vary depending on the convention used in the corpus.)

You can specify a pattern with syntactic dependencies with the attribute `syndep` in the `<w>` element of the patterns file. First you assign an `id` to a word, and then you refer back to it with the syntax `<w syndep="deptype:id">`. This is so that the pattern is not dependent on the actual word numbers. For example:

```
<!-- Match a verb and its direct object, with possible
     irrelevant intervening material. -->
<pat>
    <w pos="V" id="v1"/>
    <pat repeat="*" ignore="true"><w/></pat>
    <w pos="N" syndep="dobj:v1" />
```

```
</pat>
```

Currently only "backward" syntactic dependencies are supported. Support for forward dependencies is planned.

C.5 Preprocessing a Corpus Using TreeTagger

This section explains how to use a POS tagger, TreeTagger, to obtain a XML corpus from a plain textual corpus.

C.5.1 Installing TreeTagger

To install TreeTagger, just follow the instructions in the "Download" section of TreeTagger's webpage.[2] In addition to TreeTagger itself, you will need to download parameter files for each language you wish to use the tagger with. We recommend that you add the path to TreeTagger to your `PATH` variable as suggested by the TreeTagger installation script, this will allow you to call it without using the full path.

C.5.2 Converting TreeTagger's Output to XML

After installing TreeTagger, you can run it by running `path-to-tree-tagger/cmd/tree-tagger-language input-file`, where language is the language of the input file. TreeTagger will read the corpus from `input-file` and print each word, together with its lemma and part of speech, as a separate line to standard output.

`mwetoolkit` comes with a script, `treetagger2xml.sh`, which takes TreeTagger's output and converts it to XML. All you have to do is feed TreeTagger's output to it:

```
path-to-tree-tagger/cmd/tree-tagger-english
corpus.txt |
python path-to-mwetoolkit/bin/treetagger2xml.py
>corpus.xml
```

From there on you can process the XML corpus using `mwetoolkit` tools, such as is shown in Sect. C.3.

[2]http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/

C.6 Preprocessing a Corpus Using RASP

This page explains how to use a parser, RASP, to obtain a XML corpus from a plain textual corpus.

C.6.1 Installing RASP

RASP doesn't need to be installed. Just download it from RASP Download.[3]

C.6.2 Converting RASP's Output to XML

After downloading RASP, you can run it by running `path-to-rasp/scripts/rasp.sh < input-file`. RASP will read the corpus from `input-file` and print for each sentence it's words, together with surface form, lemma and part of speech. Then, it will print the grammatical relations, which can be viewed as a kind of dependency tree.

`mwetoolkit` comes with a script, `rasp2mwe.py`, which takes RASP's output and converts it to XML. All you have to do is feed RASP's output to it:

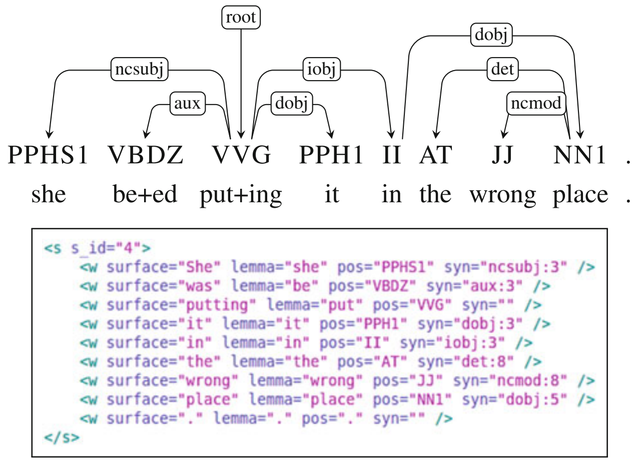
```
path-to-rasp/scripts/rasp.sh < corpus.txt |
python path-to-mwetoolkit/bin/rasp2mwe.py >corpus.xml
```

From there on you can process the XML corpus using `mwetoolkit` tools, such as is shown in Sect. C.3.

C.7 Examples of XML Files

Figure C.1 shows an example of sentence in a XML corpus file. There are four possible attributes that can be defined at the word level: `surface` for the surface form, `lemma`, `pos` for the part of speech and `syn` for the dependency syntactic relation. Syntactic relations are represented as a pair `type:parent` where the first element is the type of syntactic relation and the second element is the position of the parent word on which the current word depends. This example sentence was parsed using the RASP parser. Word indices start at 1. The corresponding tree representation would be:

[3]http://ilexir.co.uk/applications/rasp/download/


root
dobj
ncsubj
iobj
det
aux
dobj
ncmod
PPHS1 VBDZ VVG PPH1 II AT JJ NN1 .
she be+ed put+ing it in the wrong place .

```xml
<s s_id="4">
    <w surface="She" lemma="she" pos="PPHS1" syn="ncsubj:3" />
    <w surface="was" lemma="be" pos="VBDZ" syn="aux:3" />
    <w surface="putting" lemma="put" pos="VVG" syn="" />
    <w surface="it" lemma="it" pos="PPH1" syn="dobj:3" />
    <w surface="in" lemma="in" pos="II" syn="iobj:3" />
    <w surface="the" lemma="the" pos="AT" syn="det:8" />
    <w surface="wrong" lemma="wrong" pos="JJ" syn="ncmod:8" />
    <w surface="place" lemma="place" pos="NN1" syn="dobj:5" />
    <w surface="." lemma="." pos="." syn="" />
</s>
```

Fig. C.1 Example of sentence in a corpus

C.8 Developers

The `mwetoolkit` was developed and is maintained by:

- Carlos Ramisch
- Vitor De Araujo
- Sandra Castellanos
- Maitê Dupont
- Alexander Kobzar

Appendix D
Tagsets for POS and Syntax

D.1 Generic POS Tagset

The parts of speech described below compose a minimal, generic, coarse-grained set of tags used in Chaps. 5–7. They are not standard and were created only to explain the MWE extraction patterns using the same tags in all languages.

- **CC** conjunction
- **DT** determiner
- **J** adjective
- **N** noun
- **P** preposition
- **R** adverb
- **V** verb

D.2 RASP English POS Tagset

The tagset used by the RASP for English POS tags is available at http://ucrel.lancs.ac.uk/claws2tags.html, reproduced below.

- **!** punctuation tag—exclamation mark
- **”** punctuation tag—quotation marks
- **$** Germanic genitive marker—(’or ’s)
- **&FO** formula
- **&FW** foreign word
- **(** punctuation tag—left bracket
- **)** punctuation tag—right bracket
- **,** punctuation tag—comma
- **-** punctuation tag—dash
- **——** new sentence marker
- **.** punctuation tag—full-stop
- **...** punctuation tag—ellipsis
- **:** punctuation tag—colon
- **;** punctuation tag—semi-colon
- **?** punctuation tag—question-mark

C. Ramisch, *Multiword Expressions Acquisition*, Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-319-09207-2

- **APP$** possessive pronoun, pre-nominal (my, your, our etc.)
- **AT** article (the, no)
- **AT1** singular article (a, an, every)
- **BCS** before-conjunction (in order (that), even (if etc.))
- **BTO** before-infinitive marker (in order, so as (to))
- **CC** coordinating conjunction (and, or)
- **CCB** coordinating conjunction (but)
- **CF** semi-coordinating conjunction (so, then, yet)
- **CS** subordinating conjunction (if, because, unless)
- **CSA** 'as' as a conjunction
- **CSN** 'than' as a conjunction
- **CST** 'that' as a conjunction
- **CSW** 'whether' as a conjunction
- **DA** after-determiner (capable of pronominal function) (such, former, same)
- **DA1** singular after-determiner (little, much)
- **DA2** plural after-determiner (few, several, many)
- **DA2R** comparative plural after-determiner (fewer)
- **DAR** comparative after-determiner (more, less)
- **DAT** superlative after-determiner (most, least)
- **DB** before-determiner (capable of pronominal function) (all, half)
- **DB2** plural before-determiner (capable of pronominal function) (e.g. both)
- **DD** determiner (capable of pronominal function) (any, some)
- **DD1** singular determiner (this, that, another)
- **DD2** plural determiner (these, those)
- **DDQ** wh-determiner (which, what)
- **DDQ$** wh-determiner, genitive (whose)
- **DDQV** wh-ever determiner (whichever, whatever)
- **EX** existential 'there'
- **ICS** preposition-conjunction (after, before, since, until)
- **IF** 'for' as a preposition
- **II** preposition
- **IO** 'of' as a preposition
- **IW** 'with'; 'without' as preposition
- **JA** predicative adjective (tantamount, afraid, asleep)
- **JB** attributive adjective (main, chief, utter)
- **JBR** attributive comparative adjective (upper, outer)
- **JBT** attributive superlative adjective (utmost, uttermost)
- **JJ** general adjective
- **JJ** general comparative adjective (older, better, bigger)
- **JJT** general superlative adjective (oldest, best, biggest)
- **JK** adjective catenative ('able' in 'be able to'; 'willing' in 'be willing to')
- **LE** leading co-ordinator ('both' in 'both…and…'; 'either' in 'either… or…')
- **MC** cardinal number neutral for number (two, three…)
- **MC$** genitive cardinal number, neutral for number (10's)
- **MC-MC** hyphenated number (40–50, 1770–1827)
- **MC1** singular cardinal number (one)
- **MC2** plural cardinal number (tens, twenties)
- **MD** ordinal number (first, 2nd, next, last)
- **MF** fraction, neutral for number (quarters, two-thirds)
- **NC2** plural cited word ('ifs' in 'two ifs and a but')

- **ND1** singular noun of direction (north, southeast)
- **NN** common noun, neutral for number (sheep, cod)
- **NN1** singular common noun (book, girl)
- **NN1$** genitive singular common noun (domini)
- **NN2** plural common noun (books, girls)
- **NNJ** organization noun, neutral for number (department, council, committee)
- **NNJ1** singular organization noun (Assembly, commonwealth)
- **NNJ2** plural organization noun (governments, committees)
- **NNL** locative noun, neutral for number (Is.)
- **NNL1** singular locative noun (street, Bay)
- **NNL2** plural locative noun (islands, roads)
- **NNO** numeral noun, neutral for number (dozen, thousand)
- **NNO1** singular numeral noun (no known examples)
- **NNO2** plural numeral noun (hundreds, thousands)
- **NNS** noun of style, neutral for number (no known examples)
- **NNS1** singular noun of style (president, rabbi)
- **NNS2** plural noun of style (presidents, viscounts)
- **NNSA1** following noun of style or title, abbreviatory (M.A.)
- **NNSA2** following plural noun of style or title, abbreviatory
- **NNSB** preceding noun of style or title, abbr. (Rt. Hon.)
- **NNSB1** preceding sing. noun of style or title, abbr. (Prof.)
- **NNSB2** preceding plur. noun of style or title, abbr. (Messrs.)
- **NNT** temporal noun, neutral for number (no known examples)
- **NNT1** singular temporal noun (day, week, year)
- **NNT2** plural temporal noun (days, weeks, years)
- **NNU** unit of measurement, neutral for number (in., cc.)
- **NNU1** singular unit of measurement (inch, centimetre)
- **NNU2** plural unit of measurement (inches, centimetres)
- **NP** proper noun, neutral for number (Indies, Andes)
- **NP1** singular proper noun (London, Jane, Frederick)
- **NP2** plural proper noun (Browns, Reagans, Koreas)
- **NPD1** singular weekday noun (Sunday)
- **NPD2** plural weekday noun (Sundays)
- **NPM1** singular month noun (October)
- **NPM2** plural month noun (Octobers)
- **PN** indefinite pronoun, neutral for number (“none”)
- **PN1** singular indefinite pronoun (one, everything, nobody)
- **PNQO** whom
- **PNQS** who
- **PNQV$** whosever
- **PNQVO** whomever, whomsoever
- **PNQVS** whoever, whosoever
- **PNX1** reflexive indefinite pronoun (oneself)
- **PP$** nominal possessive personal pronoun (mine, yours)
- **PPH1** it
- **PPHO1** him, her
- **PPHO2** them
- **PPHS1** he, she
- **PPHS2** they
- **PPIO1** me
- **PPIO2** us
- **PPIS1** I
- **PPIS2** we
- **PPX1** singular reflexive personal pronoun (yourself, itself)
- **PPX2** plural reflexive personal pronoun (yourselves, ourselves)
- **PPY** you
- **RA** adverb, after nominal head (else, galore)

- **REX** adverb introducing appositional constructions (namely, viz, e.g.)
- **RG** degree adverb (very, so, too)
- **RGA** post-nominal/adverbial/ adjectival degree adverb (indeed, enough)
- **RGQ** wh- degree adverb (how)
- **RGQV** wh-ever degree adverb (however)
- **RGR** comparative degree adverb (more, less)
- **RGT** superlative degree adverb (most, least)
- **RL** locative adverb (alongside, forward)
- **RP** prep. adverb; particle (in, up, about)
- **RPK** prep. adv., catenative ('about' in 'be about to')
- **RR** general adverb
- **RRQ** wh- general adverb (where, when, why, how)
- **RRQV** wh-ever general adverb (wherever, whenever)
- **RRR** comparative general adverb (better, longer)
- **RRT** superlative general adverb (best, longest)
- **RT** nominal adverb of time (now, tommorow)
- **TO** infinitive marker (to)
- **UH** interjection (oh, yes, um)
- **VB0** be
- **VBDR** were
- **VBDZ** was
- **VBG** being
- **VBM** am
- **VBN** been
- **VBR** are
- **VBZ** is
- **VD0** do
- **VDD** did
- **VDG** doing
- **VDN** done
- **VDZ** does
- **VH0** have
- **VHD** had (past tense)
- **VHG** having
- **VHN** had (past participle)
- **VHZ** has
- **VM** modal auxiliary (can, will, would etc.)
- **VMK** modal catenative (ought, used)
- **VV0** base form of lexical verb (give, work etc.)
- **VVD** past tense form of lexical verb (gave, worked etc.)
- **VVG** -ing form of lexical verb (giving, working etc.)
- **VVN** past participle form of lexical verb (given, worked etc.)
- **VVZ** -s form of lexical verb (gives, works etc.)
- **VVGK** -ing form in a catenative verb ('going' in 'be going to')
- **VVNK** past part. in a catenative verb ('bound' in 'be bound to')
- **XX** not, n't
- **ZZ1** singular letter of the alphabet: 'A', 'a', 'B', etc.
- **ZZ2** plural letter of the alphabet: 'As', b's, etc.

D.3 RASP English Grammatical Relations

The set of grammatical relations used by the RASP in English is available at http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-662.pdf, reproduced below. It is organised as a tree of more generic to more specific relations. Underspecified relations may be assigned when the parser cannot disambiguate a construction.

- **dependent** : underspecified dependence
- **ta** : text adjunct
- **arg_mod** : underspecified argument or modifier
- **det** : determiner (articles, quantifiers, partitives)
- **aux** : auxiliary verb
- **conj** : coordination
- **mod** : underspecified modifier
- **arg** : underspecified argument
- **ncmod** : non-clausal modifier
- **xmod** : predicative modifier
- **cmod** : clausal modifier
- **pmod** : prepositional modifier
- **subj** : underspecified subject
- **ncsubj** : non-clausal subject
- **xsubj** : predicative subject
- **csubj** : clausal subject
- **subj_dobj** : underspecified subject or direct object
- **comp** : underspecified complement
- **obj** : underspecified object
- **dobj** : direct object
- **obj2** : second direct object (for double object verbs)
- **iobj** : indirect (prepositional) object
- **pcomp** : prepositional complement
- **clausal** : underspecified clausal
- **xcomp** : predicative complement
- **ccomp** : clausal complement

D.4 TreeTagger English POS Tagset

The tagset used by the TreeTagger in English is available at ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz, reproduced below.

- **CC** Coordinating Conjunction
- **CD** Cardinal Number
- **DT** Determiner
- **EX** Existential there
- **FW** Foreign word
- **IN** Preposition or subordinating conjunction
- **JJ** Adjective
- **JJR** Adjective, comparative
- **JJS** Adjective, superlative
- **LS** List item marker
- **MD** Modal
- **NN** Noun, singular
- **NNS** Noun, plural
- **NNP** Proper noun, singular
- **NNPS** Proper noun, plural
- **PDT** Predeterminer
- **POS** Possessive ending
- **PRP** Personal pronoun
- **PRP$** Possessive pronoun
- **RB** Adverb
- **RBR** Adverb, comparative
- **RBS** Adverb, superlative
- **RP** Particle
- **SYM** Symbol
- **TO** to
- **UF** Interjection
- **VB** Verb, base form
- **VBD** Verb, past tense
- **VBG** Verb, gerund or present participle
- **VBN** Verb, past participle
- **VBP** Verb, non-3rd person singular present
- **VBZ** Verb, 3rd person singular present
- **WDT** Wh-determiner
- **WP** Wh-pronoun
- **WP$** Possessive wh-pronoun
- **WRB** Wh-adverb

Appendix E
Detailed Lexicon Descriptions

E.1 Sentiment Verbs Extracted from Brazilian WordNet

List of sentiment verbs extracted from the Brazilian version of WordNet. We intend to investigate the relation between these verbs and the complex predicates extracted in Sect. 6.2.

abalar	apreciar	desadorar	desprezar
abominar	arrasar	desagradar	distrair-se
aborrecer-se	assanhar	desagradar-se	doer
abrandar	atormentar-se	desagradecer	embaraçar
acalmar	atraiçoar	desalentar-se	emburrar
acalmar-se	atrair	desangustiar	encantar
acender-se	atrapalhar-se	desanimar	encantar-se
acovardar-se	babar-se	desapoquentar	encorajar
adorar	cativar	desassossegar	enfurecer
afligir	chatear	desconfortar	enfurecer-se
agitar-se	cobiçar	desejar	enlouquecer
agradar	comover	desemburrar	enlouquecer-se
alarmar	comover-se	desemburrar-se	enlutar
alarmar-se	compadecer-se	desencabular	enlutar-se
alegrar	conciliar	desencorajar	entristecer
aliviar	confortar	desenjoar	entristecer-se
alterar	conquistar	desesperar-se	entusiasmar
alucinar	consolar	desfazer-se	entusiasmar-se
alvoroçar	consolar-se	desiludir	envaidecer-se
animar	consumir-se	desinteressar	envergonhar
antipatizar	decepcionar	desmotivar	espezinhar
apiedar	decepcionar-se	despertar	estimar
apoquentar	deleitar-se	despreocupar	estimular-se

C. Ramisch, *Multiword Expressions Acquisition*, Theory and Applications of Natural Language Processing, DOI 10.1007/978-3-319-09207-2

exasperar	impacientar-se	magoar-se	pirraçar
exasperar-se	incomodar	malucar	preferir
excitar	inferiorizar-se	nublar	preocupar-se
expectar	inquietar-se	nublar-se	rebaixar-se
expiar	intimidar	obsequiar	simpatizar
fascinar	intimidar-se	orgulhar-se	sossegar
frustrar	invejar	penitenciar-se	temer
fustigar	irar-se	perrengar	torturar
horrorizar	irritar-se	perturbar	venerar
horrorizar-se	irromper	perturbar-se	zangar
humilhar-se	lastimar		

E.2 Sentiment Nouns

List of Brazilian Portuguese nouns expressing sentiments. These nouns were identified using the morphosyntactic patterns described in Sect. 6.2.1.2 and manual validation.

admiração	coragem	instinto	rancor
adoração	culpa	interesse	receio
ambição	curiosidade	inveja	rejeição
amor	desejo	irritação	remorso
angústia	desespero	mágoa	repugnância
ansiedade	desprezo	medo	repulsa
antipatia	devoção	moleza	respeito
apego	dificuldade	necessidade	responsabilidade
apelo	disposição	nojo	sabor
apreço	dó	nostalgia	saudade
asco	dor	obsessão	segurança
aspiração	dor-de-cabeça	ódio	sensação
atração	dúvida	orgulho	sentimento
bronca	esperança	paciência	simpatia
carinho	expectativa	paixão	sintoma
certeza	fadiga	pâ nico	suador
cheiro	falta	pavor	suspeita
choque	fascinação	pena	tentação
ciúme	fobia	piedade	tranquilidade
compaixão	fome	prazer	trauma
complexo	frio	predileção	tristeza
confiança	gosto	preguiça	vergonha
consciência	horror	preocupação	vontade
constrangimento	ímpeto	pudor	
convicção	impressão	raiva	