

How to parse Portuguese

- [How to train the parser](#)
- [How to parse tagged files](#)

How to train the parser

```
$PARSING = /home1/b/beatrice/parsing (on cluster)
$UT = /home/migration/other/MIDENG/MIDDLE-FRENCH/ut (sic)
```

- construct training data
 - concatenate training files
 - strip comments using `clean-up-mjc`
 - strip ID nodes using `rm-id.prl`
 - check for illegal tags. This is probably easiest done by hand, using the following (`\011` = tab, `\012` is linefeed)

```
cat <training-data> | tr -s '\011' '\012' | tr -s ' ' '\012' | grep '(' |
sort | uniq > TMP
```

It is also possible to use CorpusSearch's `make_tag_list` utility (see `get-tags.q`)

- if the file contains morphological subtags (the ordinary case), run the training data through `utf8-html` (the parser strips the subtags, and the restore script currently handles only ASCII text).
- copy training data to appropriate directory on cluster
 - schematically: `$PARSING/DOMAIN/MODEL#`
 - example: `$PARSING/french/model3`
- add appropriate entry in `$PARSING/Makefile`
- show queue using `qstat` (`-f` shows all jobs)
- train parser
 - `cd $PARSING`
 - schematically: `submittrain2.sh DOMAIN MODEL#`
 - example: `submittrain2.sh gtrc 4`
- output of training is written to `$PARSING/DOMAIN/MODEL#`

How to parse tagged files

This section outlines the steps required to turn POS-tagged files into parsed files that are compatible with CorpusDraw or the older Emacs-based correction interface.

```
$PARSING = /home1/b/beatrice/parsing (on cluster)
$PORT = /home/migration/other/MIDENG/PORTUGUESE
$UT = /home/migration/other/MIDENG/MIDDLE-FRENCH/ut (sic)
```

- check for illegal tags

```
cat <training-data> | tr ' ' '\012' | grep . | cut -d '/' -f 2 | tr '+' '\012' |
sort | uniq > TMP
diff TMP $PORT/training-tagger/LEGAL-TAGS | grep '<'
```

- generate preparse files

- remove extraneous material
 - xml headers and footers
 - instructions for CorpusSearch
 - ID nodes
- if necessary, tag any other xml annotation as CODE
- convert slashes in end tags (</, />) to double dollar signs
- convert any parens to <paren> / <\$\$paren>
- remove any incorrect tags with \$UT/clean-up-lex
- remove any trailing line-final spaces
- convert to word-per-line format with `tr -s ' ' '\012'`
- check that each line contains a slash with `grep . $file | grep -v '/'`
- separate tokens by blank lines, adding one at end of file if necessary
- surround tokens by parens using `$UT/delimit-tokens-for-parser`
- add last character of file if necessary (e.g. "COD" becomes "CODE")
- `$UT/pos2psdForParser`

- sanity checks for .preparse files

- check for unbalanced parens with `forward-sexp`
- run sanity checks
 - `grep -v ' '`
 - `grep -v '^[^]* ([^]*)$'`
 - `grep '<[^>]*[()]'`
 - `grep '[].*[]'`
 - `grep '[{}]' | grep -v 'CODE'`
 - `grep '[<>]' | grep -v 'CODE'`

- if the file contains morphological subtags (the ordinary case), run the training data through `utf8-html` (the parser strips the subtags, and the restore script currently handles only ASCII text).
- copy .preparse file(s) to appropriate run#/files directory on cluster (may involve moving files to subdirectory so as to make use of `scp -r`)
- do not edit files on cluster with emacs - make changes on babel
- add appropriate entry in \$PARSING/Makefile
- best to use `classicalport.properties` (which allows parser to change POS tags); `classicalport2.properties` (which forces parser to respect given POS tags) parser often crashes due to sparse data bug
- show queue with `qstat (-f shows all jobs)`
- parse file(s)
 - `cd $PARSING`

- Schematically: `submittest.sh DOMAIN RUN#`
- Example: `submittest.sh gtrc 4`
- correct null parses
 - check whether null parses are represented by the preparse token (`grep '^((')` or by null (`grep '^null$'`)
 - if the latter, restore the text. For the moment, this needs to be done by hand (waiting for Seth to send me the appropriate script 7/3/08)
- correct various postparse problems with `.../parsing/ut/fix-postparse-problems`

```
FILE.preparse.parsed > FILE.preparse.parsed.nn
```

 - delete (`CODE +unknown+`)
 - fix (`CODE (CODE)`)
 - add root node to fixed null parses
 - switch order of words and tags in manually fixed null parses
- if necessary, restore subtags
 - in parsing directory, restore subtags (note quotes and path information in arguments)
 - Schematically:


```
make DOMAIN="dir" RUN="n" PREPARSE="path1" PARSED="path2" OUT="path3" restore
```
 - Example:


```
make DOMAIN="cordial" RUN="1" PREPARSE="cordial/run1/TEST.preparse"
PARSED="cordial/run1/TEST.preparse.parsed.nn" OUT="cordial/run1
/TEST.preparse.parsed.nn.with.subtags" restore
```
 - Common causes for the script to fail include `+unknown`, doubled tags, and the like. If possible, improve `fix-postparse-problems`.
- copy parsed files (with null parses fixed and with subtags) to postparse directory on babel. The scripts there assume the extension `.preparse.parsed.nn`.
- check for unbalanced parens with `forward-sexp`
- restore accented characters in output with `html-utf8`
- `../ut/make-psd-unfmt *.parsed.nn`
- pretty print using `CS ../queries/reformat.q *.psd.unfmt`
- if necessary, `add-blurb`
- write `.psd` file to appropriate `$PARSED` directory and archive copy to appropriate `$PARSED-ORIG` directory, using `$UT/fmt-to-psd *.psd.unfmt.fmt`
- if necessary, regenerate IDs with `re-ref`
- run appropriate queries to fix clitics and so on