# PUT
(Parser of the Università di Torino)

**User Manual and General Information**
**January, 2008**

***TABLE OF CONTENTS***

## 0. Foreword

This report includes four sections. The first one says how one can use the parser to work on the input data (user manual). The second one describes the format of the result of the analysis. It includes a list of the syntactic categories (POS) and of the other pieces of information that the parser includes in the result file. The third one presents a general description of the approach we adopted for parsing and the overall organization of the system. The last section describes one more functionality, enabling the user, after a manual correction of the output, to carry out some checks and some automatic counting of the errors.

N.B. In this manual, *HOME-DIR* refers to the directory where the system has been installed, followed by "/JAN-08-TULE". So, if it was installed in
*/home/nlresources/turinparser*
a file name as
**HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/SUBCAT-KB-ALL/lab-hierarchy.dat*
refers to the file:
*/home/nlresources/turinparser/JAN-08-TULE/SYSTEM/ALLLANG/KB-ALL/SUBCAT-KB-ALL/lab-hierarchy.dat*

## 1. User manual

### 1.1 Preliminary operations

- First of all, for installing the system, it is necessary to modify the files
  ```
  *HOME-DIR*/SYSTEM/compile-all
  ```
  and
  ```
  *HOME-DIR*/SYSTEM/nlpp
  ```
  changing in the second line, the value
  ```
  /home/lesmo/TULE/DOWNL/JAN-08-TULE/SYSTEM/
  ```
  with
  ```
  *HOME-DIR*/SYSTEM/
  ```
  Note that the new path must end with SYSTEM followed by a slash!

- If one wants to use the Lisp debugging facilities, it is possible to install the source version of the files. In this case, the same change as the one described above must be made to the file
  ```
  *HOME-DIR*/SYSTEM/nlpp-debug
  ```

- Now, it is possible to compile the parser, launching the Lisp interpreter from the directory *HOME-DIR*/SYSTEM and evaluating
  ```
  (load "compile-all")
  ```
  [1]

### 1.2 Basic operations

The operations needed to carry out the parsing process are very simple. Some care is required just for the placement of the data file on which the work must be done and for the way to tell the system which is (are) this file (these files).

[1] Some off-line facilities for generating morphological and subcategorization information are carried out by loading the files
```
*HOME-DIR*/PROC/lohier
```
and
```
*HOME-DIR*/PROC/loadsuff
```
These facilities should not be used by the standard user. Anyway, they require analogous changes for the *HOME-DIR* in the files just mentioned.

The operations are the following:

1. Insert the file (any file in a standard text format) in the directory
    *HOME-DIR*/DATI
   (but see point 5 below).
   Insert, as the two first lines of the file:
   ```
   FILE-LABEL xxx
   INITIAL-SENT-NUMBER n
   ```
   Where "xxx" is any character sequence, and "n" is an integer.
2. Start up the LISP interpreter ('cl', 'lisp', or similar) from the directory *HOME-DIR*/SYSTEM
3. Load the parser files (command '(load "nlpp")'); if one wants to use the source version - non-compiled -, the command '(load "nlpp-debug")' must be executed.
4. Answer '1' to the question "Which of the options do you want to follow?"
5. Specify what is the file to be parsed. This can be done in two ways:
   - Answering 'n' to the next question:
     "Do you want to carry out the work on all the files of the
      tagged corpus? (y/n)"
     and providing the parser with the name of the file (after the request
     "Name of the input file containing the raw text
      (between quotation marks)? Home directory: *HOME-DIR*/DATI"
     Note that the name must be a path starting from the directory *HOME-DIR*/DATI. This means that if the data have been put in the 'test' file in the subdirectory '*HOME-DIR*/DATI/MYFILES', the answer must be 'MYFILES/test'
   - Answering 'y' to the question, but after having inserted in the file "*HOME-DIR*/DATI/tagcorpus.dat" (with any text editor) the name of the files (one or more) to be tagged. The names must be included in a LISP list. An example is reported in the file *HOME-DIR*/DATI/tagcorpus.exm.

### 1.3 Other functionalities

The operations executed in case one answers with value other than 1 to the initial request of the system will be described afterward. They include: simple morphological analysis, PoS Tagging (without parsing), and evaluation. The latter is based on the availability of files containing the correct results; they have to be built manually in order to carry out the comparisons.

## 2. The parser output

The result of the parser appears in the file (files) having the same name as the input file (and in the same directory), but the extension .prs (ex. input in *HOME-DIR*/DATI/MYFILES/test; output in *HOME-DIR*/DATI/MYFILES/test.prs)

It consists in a set of 'sentences' (as identified by the parser). They are kept apart by an empty line, and by a line of the form
```
    ******* xxx-n *********
```
where "xxx" is the label (identifier of the file group, see point 1 in section 1.2 above), and "n" is the number of the sentence; the first sentence of the file will get the number "n" (as specified as INITIAL-SENT-NUMBER in the input file) the next "n+1", and so on.

Each sentence is described as a sequence of lines (records), one for each input 'word' (or punctuation mark, or special symbol) appearing in the input file. Some extra lines appear in case of compound words and traces.

*Example 1:*
*5 prova (PROVA NOUN COMMON F SING PROVARE TRANS) [3;DET+DEF-ARG]*

Each line includes four elements:

*a) The position of the word (or symbol) in the sentence (5, in the example)*

In the case of compound words (ex. "della": "of+the"), the first component ("di": "of") will get "K" as its position, while the second one ("la": "the") will get "K.1" (in the following line). Analogously, for words that include more than two components (ex. "prendiglielo": "take+it+from_him"), we will have more lines in the file, with positions: "K", "K.1", "K.2", ...

b) *The form of the input word ('prova', in the example)*
This element is case-sensitive, since upper-case and lower-case distinction mirrors the input element

c) *A list containing the data obtained by the morphological analyser and the tagger: (PROVA NOUN COMMON F SING PROVARE TRANS)*
These data are given in LISP-readable format (for instance special symbols are stored in LISP character format: #\;). The rest of this chapter aims at describing the format of these data.

d) *The information about the parsing tree: [3;DET+DEF-ARG]*
This stuff is described in the section 2.2. In this example, *3* specifies that the governor of *prova* in the dependency tree is the word in line *3*, and that the label of the connecting arc is *DET+DEF-ARG*. Section 2.2.1 is devoted to the arc labels.

## 2.1 Lexical Information[2]

The information is given as a list:

*(lemma categ type* $info_4$ *...* $info_n$*)*

The data '*lemma*' and '*categ*' refer to the same pieces of information in all lines:

- lemma: the normalized (quotation) form of the word. For instance, for nouns the masculine singular, for the verbs the infinite.
- categ: the syntactic category (PoS). The list of syntactic categories is reported in section 2.1.1.1

'*type*' appears for almost all the items and specifies the type (or subcategory) of the element. Some exceptions are elements of the categories '*punct*', '*special*', '*phras*', '*interj*' and '*marker*' (see §2.1.1). The complete description of the types is reported in section 2.1.1.2.

The further '$info_i$' are strictly dependent on the syntactic category (ex. gender and number for adjectives, tense and mood for verbs) and will be described in section 2.1.1.2.

### 2.1.1. The syntactic categories (PoS)

#### 2.1.1.1 List of defined categories

1. *ADJ* (adjectives)
2. *ADV* (adverbs)
3. *ART* (articles)
4. *CONJ* (conjunctions)
5. *DATE* (dates)
6. *INTERJ* (interjections)
7. *MARKER* (markers)
8. *NOUN* (nouns)
9. *NUM* (numbers)
10. *PHRAS* (phrasal)

[2] This and the next subsections provide the reader about basic information about the output of the parser. In order to get more details, as well as linguistic motivations for the choices made, the interested reader is addressed to the *Linguistic Notes*, available in the TUT site (http://www.di.unito.it/~tutreeb/ling-notes/ling-notes-index.htm)

11. *PREDET* (predeterminers)
12. *PREP* (prepositions)
13. *PRON* (pronouns)
14. *PUNCT* (punctuation)
15. *SPECIAL* (special symbols)
16. *VERB* (verbs)

#### 2.1.1.2 Comments, features and examples

This paragraph provides the user with general information about the elements included in the various categories. We include some comments about the category, with few (Italian) examples in context, with English (literal) translation. We also list the various subtypes of each category, with examples (out of context) taken from the corpus. For each type (or for the category, if it has no types) we provide the general form of a TUT line[3], corresponding to the parser output. Upper-case items correspond to constants, lower-case ones to variables, items enclosed in question marks to optional elements.

In the following description, the value *ALLVAL* refers to an undefined value, unifiable with all possible values.

All entries can include the marker *LOCUTION* at the end of the list of features. This specifies that the entry refers to a multi-word expression (see the next sub-section).

1. *ADJ (adjectives)*
   It includes various types of adjectives: standard (qualificativi) (bello -nice-, buono -good), interrogative ('quali' fiori vuoi comprare - 'which' flowers do you want to buy-), demonstrative ('questi' fiori sono belli - 'these' flowers are nice-), deictic (il 'prossimo' mese - [the] 'next' month), exclamative ('che' bei fiori! -'what' nice flowers!-), indefinite ('alcuni' ragazzi -'some' boys), possessive (la 'mia' auto - [the] 'my' car), ordinative (il 'terzo' successo - the 'third' success).
   - DEITT (deictic)
     *(lemma ADJ DEITT gender number)*
       *Gender: M, F, ALLVAL*
       *Number: SING, PL, ALLVAL*
     I: altro, fa, prossimo, scorso
     E: next
   - DEMONS (demonstrative)
     *(lemma ADJ DEMONS gender number)*
       *Gender: M, F, ALLVAL*
       *Number: SING, PL, ALLVAL*
     I: questo, quello
     E: such, this, that
   - EXCLAM (exclamative)
     *(lemma ADJ EXCLAM gender number)*
       *Gender: M, F, ALLVAL*
       *Number: SING, PL, ALLVAL*
     I: che
   - INDEF (indefinite)
     *(lemma ADJ INDEF gender number ?grade?)*
       *Gender: M, F, ALLVAL*
       *Number: SING, PL, ALLVAL*
       *Grade: SUPERL*
     I: nessun, alcuni, molti, qualsiasi [SUPERL for "moltissimi" - very many]
     E: numerous, certain, few
   - INTERR (interrogative)
     *(lemma ADJ INTERR gender number)*

[3] TUT (the Turin University Treebank: http://www.di.unito.it/~tutreeb/) is the dependency Treebank of Italian created and maintained by the NLP group of the Department of Informatica of the University of Torino. The description below applies in exactly the same way to the output of the parser and to the TUT data representation.

*Gender: M, F, ALLVAL*
*Number: SING, PL, ALLVAL*
I: che, quale, quanto
E: what, which

- ORDIN (ordinal)
  *(lemma ADJ ORDIN gender number value ?grade?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Value: an integer, £LAST, £N, £PENULT*
  *Grade: SUPERL*
  I: primo, ventesimo, ultimo, ennesimo [SUPERL for "ultimissimo" – very last]
  E: first, twentieth, last
- ORDINSUFF (ordinal suffixes)
  [they are not autonomous words, but appear in the dictionary, in order to cope with forms as 23rd, 45th, which are split in two parts by the tokenizer]
  *(lemma ADJ ORDINSUFF gender number)*
  *Gender: ALLVAL*
  *Number: ALLVAL*
  E: nd, rd, th, st
- POSS (possessive)
  [for this type, it is also specified the person, gender, and number of the possessor; for instance, in "mia" – my$_{f,sing}$ –, we have that the gender is feminine, the number is singular, the possessor person is 1$^{st}$, the possessor number is singular; in Italian, the gender of the possessor is not marked. Conversely, for "her", we get that gender and number are ALLVAL (as for all English adjectives), the possessor person is 3$^{rd}$, the possessor number is singular, and the possessor gender is feminine]
  *(lemma ADJ POSS gender number ?possessor-person? ?possessor-gender? ?possessor-number?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Possessor-person: P-1, P-2, P-3*
  *Possessor-gender: P-M, P-F, P-N, P-ALLVAL*
  *Possessor-number: P-SING, P-PL, P-ALLVAL*
  I: altrui, mio, nostri
  E: my, your, their
- QUALIF (qualificative)
  *(lemma ADJ QUALIF gender number ?grade?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Grade: COMPAR, SUPERL*
  [*COMPAR is used for 'maggiore', 'greater'; in Italian it is limited to a few adjectives, but in English it is productive: finer, closer, …. Conversely, in Italian, SUPERL (bellissimo – very nice – vicinissimo – very close) are productive, while in English they are not]*
  I: bello, grande, italiano
  E: nice, big, English

2. *ADV (adverbs)*
   It includes standard adverbs (spesso -often-, bene -well-), question adverbs (quando -when-, perchè -why-), and many other types. Probably, this is the most complex category, because the types (subcategories) partially include semantic information.
   - ADFIRM (adfirmative)
     *(lemma ADV ADFIRM)*
     I: certo
   - ADVERS (adversative)
     *(lemma ADV ADVERS)*

I: anzi, invece, però
E: although, though

- COMPAR (comparative)
  *(lemma ADV COMPAR)*
  I: più, meglio, peggio, così
  E: less, more
- CONCESS (concessive)
  *(lemma ADV CONCESS)*
  I: anche, pure
  E: also
- DOUBT (doubt)
  *(lemma ADV DOUBT)*
  I: forse
  E: perhaps
- EXPLIC (explicative)
  *(lemma ADV EXPLIC)*
  I: dunque
  E: that_is
- INTERJ (interjections)
  *(lemma ADV INTERJ)*
  I: comunque, tuttavia
  E: at_any_rate
- INTERR (interrogative)
  *(lemma ADV INTERR semtype)*
  *semtype: LOC, MANNER, REASON, TIME*
  I: come, dove, perché, quando
  E: how, where, when, why
- LIMIT (limit)
  *(lemma ADV LIMIT)*
  I: solo, soltanto
  E: just, only
- LOC (locative)
  *(lemma ADV LOC)*
  I: sopra, intorno, lassù, sottoterra
  E: there, within, below, here
- MANNER (manner)
  *(lemma ADV MANNER)*
  I: così, volentieri
  E: aloud, alright, well
  [this type includes also the adverbs derived from adjectives by means of the -mente suffix [I] (forte --> fortemente) or -ly suffix [E] (strong --> strongly)]
- NEG (negation)
  *(lemma ADV NEG)*
  I: non, senza, neanche, nemmeno
  E: not
- QUANT (quantification)
  *(lemma ADV QUANT)*
  I: meno, circa, assai, troppo
  E: little, rather, too
- REASON (motivation)
  *(lemma ADV REASON)*
  I: infatti, pertanto, quindi
- STRENG (strengthening)
  *(lemma ADV STRENG)*
  I: inoltre, perfino, persino, anche
  E: even, moreover
- SUPERL (superlative)
  *(lemma ADV SUPERL)*
  E: most
- TIME (time)
  *(lemma ADV TIME)*
  I: poi, prima, ormai, spesso

E: sometime, afterward, already

3. *ART (articles)*
   - DEF (definite)
     *(lemma ART DEF gender number)*
       *Gender: M, F, ALLVAL*
       *Number: SING, PL, ALLVAL*
     I: il, la, gli
     E: the
   - INDEF (indefinite)
     *(lemma ART INDEF gender number)*
       *Gender: M, F, ALLVAL*
       *Number: SING, PL, ALLVAL*
     I: un, una, un', uno, degli
     E: a, another,
   - GENITIVE (genitive):
     *(XGENITIVE ART GENITIVE)*
     *[English genitives (John's car) are represented by assuming that 's is an article, while, in the example, 'John' is linked to it via an arc labelled DET+GENITIVE-RESTR, and 'car' via DET+GENITIVE-ARG]*
     E: 's

4. *CONJ (conjunctions)*
   Conjunctions, both coordinating (e -and-, o -or-) and subordinating ('mentre' mangiava, leggeva il giornale - 'while' she was eating, she was reading the newspaper-; lo ha baciato 'perchè' lo amava - she kissed him 'because' she loved him-)
   - COORD (coordinative)
     *(lemma CONJ COORD semtype)*
       *semtype: ADVERS, COMPAR, CONCL, COORD, DISJ, EXPLIC, NEG, TIME*
     I: e, o, ma, eppure, inoltre
     E: and, but, or, neither, nor
   - SUBORD (subordinative)
     *(lemma CONJ SUBORD semtype)*
       *semtype: ADVERS, CAUS, COMPAR, CONCESS, CONCL, COND, CONSEQ, EXCLUDING, FINAL, GOAL, LOC, MANNER+TIME, MANNER+TIME+COMPAR, NEUTRAL, REASON, TIME*
     I: che, nonostante, poiché, quando
     E: since, that, to, unless
   - COMPAR (comparative)
     *(lemma CONJ COMPAR)*
     I: a, che, di, come
     E: than

5. *DATE (dates)*
   The dates, but just when they have been recognized on the basis of their structure (i.e. by the tokenizer). For instance, '10/5/07' will be recognized as a single element (a single output line), and it will get the category DATE. On the contrary, '10 maggio 2007' -10 May 2007- will be taken as three separate elements (a NUM, a NOUN, and another NUM). For this reason, they do not appear in the dictionaries, but only in the output.
     *(lemma DATE year)*
     *(lemma DATE day month year)*
   The two forms are associated with expressions as '08 (first form, year=2008) and 18/8/08 (second form: day=18, month=AUG, year=2008)
     No types.

6. *INTERJ (interjections)*
   *(lemma INTERJ)*
     No types

I: ah, bah, oh
E: alas

7. *MARKER (markers)*
This category has been created in order to handle typographic and formatting markers. Currently, it is used just for some markers which appear in the text of the used corpus, which are of the form <P Prose>, <N Smith John>. In principle, this POS can be associated with any kind of extra-text markers (ex. LaTex commands or HTML tags). However, this facility is currently very limited and there is no interface enabling a user to define easily a set of markers.
*(lemma MARKER)*
No types

8. *NOUN (nouns)*
- COMMON
  *(lemma NOUN COMMON gender number ?v-deriv? ?transitivity?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *v-deriv: a verbal lemma, or DUMMY*
  *transitivity: trans, intrans, refl*
  *v-deriv* specifies what is the verb (if any) from which the noun derives and *transitivity* says if that verb is transitive or not. These features are very important for the module which assigns automatically the grammatical relations (see section 3). For instance, with 'la caduta di Marco' -the fall of Marco-, since 'caduta' -fall- derives from 'cadere' -to fall-, which is intransitive, the arc connecting the noun 'caduta' to the preposition 'di' is assigned the relation N-SUBJ (nominal subject). On the contrary, with transitive verbs, the assigned label is *N-OBJ* ('la distruzione della città' –the destruction of the city-); of course, in this case, it is just a preference, since counterexamples do exist. *V-deriv* can assume the value *DUMMY*, in case one just wants to force the label assignment (*N-SUBJ* or *N-OBJ*) described above but no verb exists from which the noun derives (as for "avversione" – aversion)
  I: casa, ragazzo, sedia
  E: house, boy, chair
- PROPER
  *(lemma NOUN PROPER ?gender? ?number? ?semantic-class?)*
  *Gender: M, F*
  *Number: SING, PL*
  *Semantic-class: an ontology concept*
  Proper names can be found in the dictionary or obtained by the analyser as a default in presence of a capitalized item. In the second case, the *gender, number* and *semantic-class* features do not appear. In the first case, the dictionary is assumed to specify a *semantic-class* for the known name, which should correspond to a concept in an ontological knowledge base. The dictionaries for proper names are organized in the following way: a general dictionary includes all names in the version of the language associated with them; this applies to geographic names and to names of famous persons. So, *Italia*, *England*, *España* are in the general dictionary, while *Italy* and *Spain* are in the English dictionary for proper names, and *Inghilterra* e *Spagna* are in the Italian dictionary for proper names.
  I: Marco, Italia, Inghilterra, England
  E: Mary, Italia, Italy, England

9. *NUM (numbers)*
numbers, both in numeric form (123.451) and in character form (centotrentasette -onehundredthirtyseven-). Most numbers are not present in the dictionaries, since they are recognized by suitable automata:
**HOME-DIR*/SYSTEM/ITALIAN/KB-ITA/MORPHO-KB-ITA/numbautom-ita.lisp"*

**HOME-DIR*/SYSTEM/ENGLISH/KB-ENG/MORPHO-KB-ENG/numbautom-eng.lisp"*
**HOME-DIR*/SYSTEM/CATALAN/KB-CAT/MORPHO-KB-CAT/numbautom-ca.lisp"*
This is not available for Spanish
*(lemma NUM value)*
*value: the numeric value of the expression*
I: zero, venti, 127, 3.14
E: zero, twenty, 127, 3.14

*10. PHRAS (phrasals)*
Words playing the role of entire sentences.
*(lemma PHRAS)*
No types
I: sì, no
E: yes, no

*11. PREDET (predeterminers)*
Words (usually expressions of quantifiers) coming before a determiner (as in "tutti i ragazzi" – all [the] boys)
*(lemma PREDET gender number)*
*Gender: M, F, ALLVAL*
*Number: SING, PL, ALLVAL*
No types
I: tutto, ambedue
E: all, both

*12. PREP (prepositions)*
Both the normal (monosyllabic) prepositions ('di' -of-, 'a' -to-, 'da' -from-, ...) and the so-called 'polysyllabic' prepositions ('durante' -during-, 'sopra' -above-, 'davanti' -before-, ...)
*- MONO*
*(lemma PREP MONO)*
I: di, a, da, in
E: of, to, from, in
*- POLI*
*(lemma PREP POLI ?semtype?)*
*semtype: ADVERS, COMPAR, LOC, TIME*
I: durante, sopra, davanti, di fronte a
E: during, above, under, in front of

*13. PRON (pronouns)*
Beyond the personal pronouns ('io' -I-, 'tu' -you-, ...), also clitics (mangiando'lo' -eating+it-), the relative pronouns (la ragazza 'che' hai visto -the girl 'whom' you saw-, la casa 'dove' sono nato -the house 'where' I was born-, ...), the interrogative pronouns ('chi' hai incontrato? -'who' did you meet?-), the indefinite ones ('molti' credono in lui -'many' believe in him-), the possessive ones (il 'mio' –mine) and the exclamative ones ('che' hai fatto! -'what' have you done!-)
- *DEMONS* (demonstrative)
*(lemma PRON DEMONS gender number ?person? case)*
*Gender: M, F, ALLVAL*
*Number: SING, PL, ALLVAL*
*Person: 1, 2, 3, ALLVAL*
*Case: LSUBJ, LOBI, LIOBJ, OBL, or any combination of them connected via "+" (as LSUBJ+LOBJ+OBL)*
I: ciò, medesimo, questo, coloro
E: this, that,
- *EXCLAM* (exclamative)
*(lemma PRON EXCLAM)*
I: che, chi
E: what
- *INDEF* (indefinite)
*(lemma PRON INDEF gender number ?person? case)*

*Gender: M, F, ALLVAL*
*Number: SING, PL, ALLVAL*
*Person: 1, 2, 3, ALLVAL*
*Case: LSUBJ, LOBI, LIOBJ, OBL, or any combination of them connected via "+" (as LSUBJ+LOBJ+OBL)*

I: chiunque, nessuno, qualcosa
E: everything, nobody, something

- *INTERR* (interrogative)
  *(lemma PRON INTERR gender number ?person? case)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Person: 1, 2, 3, ALLVAL*
  *Case: LSUBJ, LOBI, LIOBJ, OBL, or any combination of them connected via "+" (as LSUBJ+LOBJ+OBL)*
  I: chi, che, cosa, quanto
  E: what, who
- *LOC* (locative)
  *(lemma PRON LOC gender number ?person? case ?clitic?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Person: 1, 2, 3, ALLVAL*
  *Case: LOC*
  *Clitic: CLITIC*
  I: ne, ci, vi
- *ORDIN* (ordinals)
  *(lemma PRON ORDIN gender number value)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Value: an integer, £LAST, £N, £PENULT*
  I: primo, secondo, cinquantesimo
  E: first, second, fiftieth
- *PERS* (personal)
  *(lemma PRON PERS gender number person case ?clitic?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Person: 1, 2, 3, ALLVAL, PARTIT (Italian partitives)*
  *Case: LSUBJ, LOBI, LIOBJ, OBL, or any combination of them connected via "+" (as LSUBJ+LOBJ+OBL)*
  *Clitic: CLITIC*
  I: io, tu, noi, lei, ci
  E: I, you, we, her
- *POSS* (possessive)
  *[For the "possessor" features, see the comments for possessive adjectives]*
  *(lemma PRON POSS gender number ?possessor-person? ?possessor-gender? ?possessor-number?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Possessor-person: P-1, P-2, P-3*
  *Possessor-gender: P-M, P-F, P-N, P-ALLVAL*
  *Possessor-number: P-SING, P-PL, P-ALLVAL*
  I: mio, tuo, nostro, proprio
  E: mine, yours
- *REFL-IMPERS* (reflexive-impersonal)
  *(lemma PRON REFL-IMPERS gender number person case ?clitic?)*
  *Gender: M, F, ALLVAL*
  *Number: SING, PL, ALLVAL*
  *Person: 1, 2, 3, ALLVAL, PARTIT (Italian partitives)*
  *Case: LSUBJ, LOBI, LIOBJ, OBL, or any combination of them connected via "+" (as LSUBJ+LOBJ+OBL)*
  *Clitic: CLITIC*
  *I:* ci, vi, si, se
  *E:* yourself, themselves

- *RELAT* (relative)
  *(lemma PRON RELAT ?gender? ?number? ?person? case)* or
  *(lemma PRON RELAT semtype)*
    *Gender: M, F, ALLVAL (absent for Italian "cui" – of which)*
    *Number: SING, PL, ALLVAL*
    *Person: 3*
    *Case: LSUBJ, LOBI, LIOBJ, OBL, or any combination of them connected via "+" (as LSUBJ+LOBJ+OBL)*
    *semtype: LOC, TIME*
  I: che, quale, cui, come, dove
  E: that, who, which, where

14. *PUNCT* (punctuation*)*
    Various punctuation marks, as periods, commas, parentheses, and so on.
    *(lemma PUNCT)*
    No types

15. SPECIAL (special*)*
    Special symbols, i.e. all characters which are not standard punctuation marks (ex. $, #, & ...).
    *(lemma SPECIAL)*
    No types

16. VERB (verbs*)*
    Main verbs, but also auxiliaries and modals. It must be noted that in all cases where the corresponding lemma is not explicitly present in the dictionary with another category, the past and present participles will be tagged as VERB. For instance, if 'interesting' appears as an ADJ in the dictionary, it is up to the tagger to choose between the ADJ and VERB (gerund) reading. Otherwise, it will appear in the input as a VERB.
    - *MAIN* (all standard verbs)
      *(lemma VERB MAIN mood tense ?transitivity? ?person? ?number? ?gender?)*
        *Transitivity: TRANS, INTRANS, REFL*
        *+ if tense = IND, CONG, CONDIZ, IMPER*
          *Person: 1, 2, 3*
          *Number: SING, PL, ALLVAL*
          *Gender: absent*
        *+ if tense = PARTICIPLE*
          *Person: absent*
          *Number: SING, PL, ALLVAL*
          *Gender: M, F, ALLVAL*
        *+ if tense = INFINITE or GERUND*
          *Person, nember and gender are absent*
      I: Andare, mangiare, dare, essere (in "essere intelligente")
      E: go, eat, give, be (in "to be intelligent")
    - *AUX* (auxiliaries)
      *(lemma VERB AUX mood tense ?transitivity? ?person? ?number? ?gender?)*
        *Same features as for MAIN*
      I: essere (in "essere baciato"), avere, venire, stare
      E: be (in "to be kissed")
    - *MOD* (modals)
      *(lemma VERB MOD mood tense ?transitivity? ?person? ?number? ?gender?)*
        *Same features as for MAIN*
      I: dovere, potere, volere
      E: must, can, will

### 2.1.2 Multiword expressions

Currently, the parser recognizes a limited number of multi-word expressions (about 300 for Italian and about 30 for English); the list is in the files **HOME-DIR*/SYSTEM/ITALIAN/KB-ITA/DICTION-KB-ITA/locutions-ita.dat* and **HOME-DIR*/SYSTEM/ENGLISH/KB-ENG/DICTION-KB-ENG/locutions-eng.dat*.

Examples of multiword expressions are "più o meno" -more or less-, "per esempio" -for instance-, "in order to". In the output, locutions have the same features as all other entries (i.e. they depend on the syntactic category), except for the presence of the item LOCUTION at the end of the syntactic information described in the previous subsection.

## 2.2 Structural information (the parse tree)

As stated above, the structure of the dependency tree is encoded via arcs from the daughters to the mother. They are expressed in the form:

*[parent;label]*

where *'parent'* is the position (see 2. above) of the word that governs the one under consideration, and *'label'* is the label of the arc connecting the daughter (dependent) to the parent (head). Of course, the root of the tree has a special *'parent'* pointer, encoded as *'0'* (zero).

### 2.2.1 The arc labels

The arc labels (dependency relations) are organized according to a hierarchical scheme. The whole hierachy currently used is stored in the file *"*HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/SUBCAT-KB-ALL/lab-hierarchy.dat"*. We only report here some short comments on the top levels of the hierarchy[4]. We also note that the labels carry both syntactic and semantic information, and that the parser assigns the labels by adopting a strategy of least-commitment, thus omitting, except in special cases, the semantic component.

This last point requires some words of explanation. The label hierarchy is such that every label has to be seen as a 'specialization' of its parent. For instance, the *RMOD* label (associated with any restrictive modifier is specialized in eight different types, according to the syntactic category of the dependent (*ADVB-RMOD, ADJC-RMOD, ...*). In their turn, these labels are specialized further in terms of semantic valency of the dependent. For instance, *ADVB-RMOD* (restrictive modifiers of aadverbial type) dominates 14 different labels (*ADVB+ADVERS-RMOD, ADVB+COMPAR-RMOD, ADVB-RMOD-LOC*, etc.), that may be specialized further (e.g. *ADVB-RMOD-LOC+FROM*). The parser, at the current level of development, limits itself to the assignment of the *ADVB-RMOD* (or *ADJC-RMOD, NOUN-RMOD, ...*), without taking more detailed decisions (apart from some cases, as, for instance, the negations - *ADVB-RMOD-NEG* - which are reasonably certain). So, if the choice of the parent is correct (i.e. if the word X actually is an adjunct of the verb Y), the link is considered as correct, avoiding the risk of errors (which could raise the total number by another 5%, with respect to the figures reported in the last section).

With this premise, we report below a small portion of the label hierarchy, just to give a feeling of its structure (in the file

*"*HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/SUBCAT-KB-ALL/lab-hierarchy.dat"*

it is possible to find further details and a number of examples). Note that in the adopted scheme, the "category" part of the label has a different role for *ARG* and for *RESTR*. For ARG, it encodes the category of the governor (e.g. *ADJC-ARG* means "argument of an adjective"), while for RMOD, it encodes the category of the modifier (i.e. *ADJC-RESTR* means "an adjective that is a modifier of something else").

Currently, 260 labels are in use, although this number gives only an idea of the size, since the labelling scheme is dynamic.

[4] More details can be found in the TUT site (http://www.di.unito.it/~tutreeb/), in the Tools section of the top page: "Labels of the edges"

```
DEPENDENT
  |--> FUNCTION
  |       |--> ARG
  |       |       |--> ADJC-ARG
  |       |       |--> ADVB-ARG
  |       |       |--> CONJ-ARG
  |       |       |--> DET-ARG
  |       |       |--> NOUN-ARG
  |       |       |--> VERB-ARG
  |       |               |--> VERB-SUBJ
  |       |               |--> VERB-OBJ
  |       |--> RMOD
  |       |       |--> ADJC-RMOD
  |       |       |--> ADVB-RMOD
  |       |       |--> DATE-RMOD
  |       |       |--> NOUN-RMOD
  |       |       |--> NUM-RMOD
  |       |       |--> PREP-RMOD
  |       |       |--> PRON-RMOD
  |       |       |--> VERB-RMOD
  |       |--> APPOSITION
  |--> NOFUNCTION
          |--> AUX
          |       |--> AUX-PASSIVE
          |       |--> AUX-PROGRESSIVE
          |       |--> AUX-TENSE
          |--> CONTIN
          |       |--> CONTIN+DENOM
          |       |--> CONTIN+LOCUT
          |       |--> CONTIN+PREP
          |--> COORDINATOR
          |       |--> COORD
          |       |--> COORDANTEC
          |       |--> COORD2ND
          |--> EMPTYCOMPL
          |--> INTERJECTION
          |--> SEPARATOR
          |       |--> OPEN
          |       |--> CLOSE
          |       |--> END
          |--> VISITOR
```

### 2.2.2 Traces and coindexing

Although the output of the parser includes a number of traces, it is necessary to make clear that the treatment of traces and coindexing is currently in an extremely preliminary phase. In the following, we report here some comments on the function, the use, and the representation of traces.

The parse tree is a tree where intersecting arcs are not allowed. In other words, it fully respects the projectivity condition[5]. This implies the adoption of some machinery enabling us to represent phenomena as movements. Moreover, in Italian, null dependents are common (resulting from pro-drop). To handle these phenomena, we introduced the traces. A trace, as in all classical approaches, stands for a position in the tree where a dependent should appear, but, on the contrary, is absent (an 'empty' or 'null' dependent). This 'hole' has been left after a movement, a sharing, or, more simply, a deletion (as in the pro-drop case).

[5] Intersecting arcs can arise due to parser errors.

These empty dependents are represented by traces, which are introduced in the tree in the following way:

1. the referent of the missing dependent is found (if any)
2. the position of the missing dependent is determined (but this is not very relevant, because the order of Italian is only partially fixed (i.e. it is a partially configurational language).
3. a line representing a trace is inserted in that position. This line has the form:

   *posit t [ref] syntinfo struct*

where:

- *posit* identifies the position of the trace in the sentence, and has the form 'ind.tr', where 'ind' is the position of the word preceding the trace, and 'tr' is 10 (and subsequent integers, 11, 12, ..., for adjacent traces).
- *t* is fixed
- *ref* is the referent, encoded as 'posV', where 'pos' is the position of the line of the referent in the sentence, and V can assume the values:
  - f (trace of the whole subtree that has the word in position 'pos' as root),
  - p (trace of a portion of that subtree),
  - w (trace of the sole referent word)

  In case the referent is not available, this third element of the trace is simply '[]'.
- *syntinfo* is a copy of the syntactic information of the referent (if any); if there is not such a referent, it has the form:

  *(GENERIC-T PRON PERS ALLVAL ALLVAL ALLVAL)*
- *struct* has the standard form '[parent;label]'

More information about traces can be found in the *Linguistic Notes* (see Footnote 2).

## 3. The parser architecture

In this chapter, we provide an overview of the overall architecture of the parser, starting from the input text file. We mention some top-level functions that take care of the various operations. The functions are listed below in a tree-like fashion, representing the nesting of function calls. Within parentheses, it is reported the name of the file where the function is defined. In these file names, we omit the full path, so that the base directory must be assumed, for all files, to be **HOME-DIR*/SYSTEM/ALLLANG/PROC-ALL*.

The top function (evaluated when the user answers *'1'* to the initial question - see section 1) is *'ana-and-tag-and-parse'*, which refers to the sequence 'morphological analysis'+'tagging'+'parsing'. All functions whose name is 'ana_XXX' are in the file *"MORPHO-PROC-ALL/analizzatore"*.

```
ana-and-tag-and-parse ("main")
 |--> ana-text+tag ("top-level-fun")
 |     |--> tokenize ("MORPHO-PROC-ALL/tokenizer")
 |     |--> analyser ("MORPHO-PROC-ALL/analizzatore")
 |     |     |--> analyser_elem ("MORPHO-PROC-ALL/analizzatore")
 |     |          |--> anal-scan-el ("MORPHO-PROC-ALL/analizzatore")
 |     |               |--> analyser-single-elem ("MORPHO-PROC-ALL/analizzatore")
 |     |                    |--> ana_year
 |     |                    |--> ana_date
 |     |                    |--> ana_sigla
 |     |                    |--> ana_np
 |     |                    |--> ana_gw     (access to the dictionaries)
 |     |                         |--> ana_noun
 |     |                         |--> ana_adj
 |     |                         |--> ana_verb
 |     |                         |--> ana_adv
 |     |                         |--> ana_enclitica
 |     |--> menudisamb+tag ("TAGGER-PROC-ALL/menudisamb")
 |--> parse-single-f ("PARSER-PROC-ALL/chunk-parser")
      |--> parse-sentences ("PARSER-PROC-ALL/chunk-parser")
             |--> remove-parentheses ("PARSER-PROC-ALL/chunk-parser")
             |--> apply-loc-parserules ("PARSER-PROC-ALL/chunk-parser")
             |--> link-conjunctions ("PARSER-PROC-ALL/chunk-parser")
             |--> find-v-complements ("PARSER-PROC-ALL/chunk-parser")
             |--> attach-unlinked ("PARSER-PROC-ALL/chunk-parser")
             |--> add-parentheses ("PARSER-PROC-ALL/chunk-parser")
```

## 3.1 The tokenizer

The first step consists in the identification of the tokens composing the input sentence (tokenization). This task is carried out on the basis of the data contained in the automaton *"*HOME-DIR*/ALLLANG/KB-ALL/MORPHO-KB-ALL/token-autom"*.

The output of the tokenizer is a list of "tokens". Note that the segmentation in tokens can be ambiguous, as, for instance, in the case of "37.", which may be a real number or an integer followed by a separator. Beyond the segmentation problems, any token can be ambiguous; the various possibilities are called "scan interpretations" (e.g., "Donato", or "Mark" can be proper nouns or verbs). Finally, every scan interpretation may be composed of more than one "scan elements" (this feature is what enables us to cope with the segmentation ambiguities mentioned above).

Each 'scan element' is associated with a 'scanner-category', which is one of the following:
- GW: General Word
- NOMEP: Proper name
- SIGLA: Abbreviation
- SEGNOINTER: Punctuation Mark
- SPECIAL: Special Symbol
- DATE: Date
- YEAR: Year
- NUMBER: number

The structure of the tokenizer output is as follows:

```
      output --> token*
      token --> scan-interp | (scan-interp++)
      scan-interp --> (scan-element+)
      scan-element --> ((ascii-code+) scanner-category)
```

Note that, because of previous design choices, a non-ambiguous token lacks a level of parentheses (I have used ++ to refer to the presence of at least two elements).

```
Examples (Asino = Donkey):
(unambiguous):
          asino: (((97 115 105 110 111) GW))
(ambiguous):
          Asino: ((((97 115 105 110 111) GW)) (((65 115 105 110 111) NOMEP)))
(ambiguous in segmentation):
          37.:   ((((51 55) NUMBER) ((46) SEGNOINTER))
                  (((51 55 46) NUMBER)))
```

Finally, note that it is a task of the tokenizer to split the text into sentences, on the basis of the punctuation marks. The decision of the tokenizer can be overcome in the following step, forcing the concatenation of two portions of the input that the tokenizer took as different sentences (for instance, in the case where the separation was established inside a parenthesis).

## 3.2 The analyzer

The task of the analyzer is to determine which of the hypotheses issued by the tokenizer are compatible with the lexicon (and the morphological knowledge). In other words, the analyzer implements the access to the dictionary by making hypotheses about the suffixes of a token and looking for the root in the dictionary. Beyond this (which is the standard treatment for tokens of 'scanner-category' = GW), the analyser takes also care of:
- lookup of proper nouns and codes in the dedicated dictionaries
- emission of 'proper name' hypotheses, in case a token is unknown, but it begins with upper case.
- management of multi-word expressions
- handling of derivations (ex. affermare --> affermazione [to state --> statement], agile --> agilità [agile --> agility]).

The morphological analysis is carried out starting from the end of the 'word' (i.e. token resulting from the tokenizer) and travelling across an 'augmented' automatons (files "*HOME-DIR*/SYSTEM/ITALIAN/KB-ITA/MORPHO-KB-ITA/network-ita.dat", "*HOME-DIR*/SYSTEM/ENGLISH/KB-ENG/MORPHO-KB-ENG/network-eng.dat"), which encode the possible suffixes. The automatons are rather complex, since they takes care also of clitics. The result of this step (hypothesis of split root+suffix) is then checked by searching the dictionary for the root (the dictionaries are in the files "*HOME-DIR*/ITALIAN/KB-ITA/DICTION-KB-ITA/dictionary-ita.dat", "*HOME-DIR*/ENGLISH/KB-ENG/DICTION-KB-ENG/dictionary-eng.dat"). In this phase, when there is no match and the word is capitalized, then a reading as a proper noun is assumed. The possible suffixes for a given root are specified in the dictionary via the feature CLASSE. The possible values of this feature are described in the files "*HOME-DIR*/DESCR/morph-adj", "*HOME-DIR*/DESCR/morph-noun", and "*HOME-DIR*/DESCR/morph-verb".

The output of the analyzer includes six level of parentheses, to represent all different types of ambiguity (in the grammar below, I have highlighted the different levels with different symbols, but of course, in LISP all parentheses are equal).

| | |
|---|---|
| element | ---> {scan-res$_1$ ... scan-res$_{n1}$} |
| scan-res | ---> [scan-el$_1$ ... scan-el$_{n2}$] |
| scan-el | ---> <word-interp$_1$ ... word-interp$_{n3}$> |
| word-interp | ---> (lex-el$_1$ ... lex-el$_{n4}$) |
| lex-el | ---> /amb-el$_1$ ... amb-el$_{n5}$\ |
| amb-el | ---> `lexical data' |

It must be observed that we maintain the alignment with the tokenizer output; so, the actual lexical ambiguities (e.g. a GW that is both a noun and a verb) are 'added' to the ones obtained by the tokenizer.

For instance, far "Barbara." we get:

*** Tokenizer output:**

```
(((((66 97 114 98 97 114 97 46) SIGLA))
 ((((66 97 114 98 97 114 97) SIGLA) ((46) SEGNOINTER))
 ((((66 97 114 98 97 114 97) NOMEP) ((46) SEGNOINTER))
 ((((98 97 114 98 97 114 97) GW) ((46) SEGNOINTER)))
```

The first interpretation is useful for inputs as "Dr.", where the period is part of the abbreviation; the second one for "... in the UE.", where UE is a code, and the period is a separator; the third is the right one (the most obvious); the last one would be correct in a context as: "Com'è questa cultura? Barbara." (How is this culture? Barbaric.); note that for GW interpretation, the leading character has been converted to lower case, to make easier the access to the dictionary.

*** Output of the morphological analyzer**[6]**:**

```
      ((NIL)                 'Barbara.' is not a known code
       (NIL ((((#\. CAT PUNCT)))))
                             The period is ok, but 'Barbara' as a code is not
       (((((BARBARA CAT NOUN PROPER YES GENDER F SEMTYPE ££NAME))))
        ((((#\. CAT PUNCT)))))
                             Ok Barbara proper noun, followed by the period.
       (((((BARBARO CAT NOUN CLASSE (3) GENDER F NUMBER SING)))
         (((BARBARO CAT ADJ CLASSE (1) TYPE QUALIF GENDER F NUMBER SING))))
        ((((#\. CAT PUNCT))))))
                             Ok Barbara general word (but with both a noun and
                             an adjective interpretation), followed by the
                             period.
```

An example of the lowest level ambiguity is given by some clitics. For instance, "porci" is composed by the verbal root "por" (resulting from an elision from "porre" – put) and "ci", which can be either a personal pronoun (us) or a locative pronoun (there); so, the ambiguity is "put us" or "put there"; furthermore, "porci" is also the plural form of "porco", i.e. pork:

```
        (((((PORRE CAT VERB CLASSE (7 ...) TRANSITIVE YES MOOD INFINITE ...))
           ((CI CAT PRON TYPE PERS ...)
            (CI CAT PRON TYPE LOC ...))))
         ((((PORCO CAT NOUN CLASSE (2) ...)))))
```

where the first interpretation refers to a verb+clitic, and the second one to the noun (porks).

A final step of processing leads to the exclusion of interpretations including unknown components (as the first two above) in case a fully known interpretation is available. Of course, the fully known interpretations are all kept as standard ambiguity, which may be resolved by the tagger.

## 3.3 The PoS tagger

The POS tagger is rule-based. It is based on the idea that, when a word is syntactically ambiguous, the ambiguity concerns a subset of the syntactic categories (PoS). So, it is possible to associate with such a subset a packet of rules devoted to that type of ambiguity. For instance, in the case of 'pesca' (peach vs. [s/he] fishes), the ambiguity is between *NOUN* and *VERB*, so that it is possible to apply the rules aiming at disambiguating between these two categories. Consequently, the tagger operations are trivial:

- determine the ambiguity on the basis of the result of the analyzer
- apply the rules of the packet associated with that ambiguity
- choose, among the interpretations of the word, the one (or the ones) of the category specified by the rule which succeeded.

The rules are in the file "*HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/TAGGER-KB-ALL/lexdisambr".

Comments:

- The rules in the same packet are NOT mutually exclusive. A precedence schema is applied, which is based on:
  a. An explicit priority value (CF in the rules; it can take three values: C - certain; A - Almost certain; U – Uncertain)
  b. The manual order in the packet.
- Different interpretations of the word may belong to the same PoS. In this case, an (intracategorial) ambiguity may remain. For some specific cases, there are packet of rules aimed at feature-based disambiguation (ex. *general-f-m* for gender of various categories, *noun-common-proper* for nouns, etc.)
- If no single choice is obtained, the final choice is random.

[6] Note that, at this level, the feature names (e.g. CAT for category, TYPE, etc. are still present. They are cancelled (i.e. not included in the final output) by the POS tagger.

The output of the tagger is not, as for the previous modules, an internal data structure, but it is stored in a file (same name and same directory as the input, but extension '.tb'). This output does not include the names of the features (CAT, GENDER, TENSE, ...), but only the values; this representation is less explicit but more readable. The content of this file has exactly the same format as the final output of the parser (see §2.1), but without the structural information (and without the traces). It must be noted that the tagger produces two more files:

1. extension '.tball': all lexical interpretations. The first interpretation is the one selected by the PoS tagger; the others are not ordered.
2. extension '.sim': sequence of citation forms (i.e. lemmata; this makes easier the search for all the occurrences of a given lemma).

## 3.4 The parser

The parser takes as input the '.tb' file produced by the PoS tagger. Note that the '.tball' file (including all the possible interpretations) is available, but is not currently used.

The parser works on single sentences (even if more sentences may appear in the file), according to the following schema:

- Remove from the sentence all material enclosed in parentheses
- Parse the sentence
- Parse the contents of the various parentheses
- Restore the parsed parentheses in the original position, trying to establish their connection to the context

To carry out the parsing (function *'parse-single-f'* in *"*HOME-DIR*/SYSTEM/ALLLANG/PROC-ALL/PARSER-PROC-ALL/chunk-parser"*) the following steps are applied:

I. Apply the non-verbal chunking rules
II. Connect the coordinations
III. Determine, for each verb, which are its dependents and assign them a grammatical function
IV. Link to the tree possible words remained unlinked.

### 3.4.1 Non-verbal chunking

This operation is carried out by means of a sequence of passes on the sentence. Each pass takes care of a chunking level: the levels are associated with the various syntactic categories, and their order is as follows (see the global variable *CHUNK-LEVELS* in *"*HOME-DIR*/SYSTEM/ALLLANG/PROC-ALL/PARSER-PROC-ALL/chunk-parser"*):

- *SPECIAL*
- *PUNCT*
- *ADV*
- *PREDET*
- *CONJ*
- *ENGLISH NOUN NOUN-ENG*
- *ADJ*
- *NUM*
- *NOUN*
- *PRON*
- *ART*
- *PREP*
- *VERB*

For each level, the 'chunking rules' associated with the category of that level are applied. Each group of rules looks, in the immediate context of a word of the given category, for its possible dependents. In other words, first we determine the chunks having as head an element of category *SPECIAL* (the only element that may be governed by a *SPECIAL* appears in the form 'legge 480/1992' - rule 480/1992 - where the *SPECIAL* '/' governs the year), then the ones having as head a *PUNCT*, and so on. When the antecedent of the rule holds, the link having as label the consequent of the rule is included in the parse tree.

The "*ENGLISH NOUN NOUN-ENG*" line refers to a specific packet of rules, to be applied only to English, having the goal of anticipating the analysis of noun-noun constructions.

Each group (level) of rules is organized in a tree-structure in order to enhance both readability and applicability. An example (a portion of the tree for the *ADJ* chunk) is the following:

```
ADJ
 |
 |--DEMONS-------|
 |               |---AFTER-------|
 |                               |-- <NOUN(AGREE), DET+DEF-ARG>
 |
 |--INDEF--------|
 |               |--FOLLOWS------|
 |                 (ADJ QUALIF*) |-- <NOUN(AGREE), DET+QUANTIF-ARG>
 |
 |--QUALIF-------|
                 |---BEFORE------|
                 |               |-- <ADV(COMPAR), ADVB+COMPAR-RMOD>
                 |               |-- <ADV(QUANT), ADVB+QUANTIF-RMOD>
                 |
                 |---AFTER-------|
                                 |-- <CONJ(COMPAR), COORD+COMPAR>
```

It must be read in the following way:
- The first level of the tree says that the underlying rules have to be applied just to *ADJ* of the specified subcategory (syntactic type). For instance, the last three rules apply to qualificative adjectives (*QUALIF*).
- The second level defines the scope and the direction of the search. The scope can be:
  - Immediate adjacency: *BEFORE* or *AFTER*, according to the required position of the dependent wrt the *ADJ* head.
  - Adjacency with possible intervening words (having some specified features): *PRECEDES* or *FOLLOWS*, according to the direction.
  - Adjacency with intervening chunk fragments: *CHUNK-PRECEDES* and *CHUNK-FOLLOWS*
- The third level includes the rules in the form *<conditions, label>*

For instance, the first rule of the tree in the figure above, can be read as:

*An adjective (ADJ) of subcategory 'demonstrative' (DEMONS) may govern a NOUN that immediately follows it (AFTER), if there is gender and number agreement (AGREE). In such a case, the label of the arc linking the NOUN to the ADJ must be DET+DEF-ARG.*

And for the second rule:

*An adjective (ADJ) of subcategory 'indefinite' (INDEF) may govern a NOUN that follows that adjective, provided that between the adjective and the noun appear only zero or more qualificative adjectives (FOLLOWS (ADJ QUALIF *)), if there is agreement; in this case, the label is DET+QUANTIF-ARG.*

An example of a rule using chunks (*CHUNK-PRECEDES* or *CHUNK-FOLLOWS*), is the following:

```
ART
 |
 |--DEF----------|
                 |-CHUNK-FOLLOWS-|
                                 |-- <NOUN(AGREE), DET+DEF-ARG>
```

It says that if a definite article is followed by elements that depend on a *NOUN* which agrees with the article, then that noun (the head of the chunk that follows) must be linked to the article via the relation *DET+DEF-ARG*. It assumes that the NP chunk has been already built via the application of other rules, and enables the parser to access the head of the chunk, independently of the intervening words.

The full set of rules, together with more details on them, is in "**HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/GRAMM-KB-ALL/parserules.dat*".

### 3.4.2 Coordination

The problem of coordination is faced in a heuristic way; the adopted method is:

1. If at the beginning, then look for a verb as a second conjunct; if not found, take any chunk head following the conjunction (note that chunks have already been built in the previous step)

2. Otherwise
   2.a – Find all possible second conjuncts
   They are assumed to be all elements that are still missing a link to a parent that follow the conjunction
   2.b – Check for the preceding words in order to find an optimal (according to some preferences) pair of first and second conjunct. The preferences are for conjuncts of the same category and (in the order):
   - adjacent to the conjunctions or
   - of *ART*, *ADJ*, or *NUM* categories, but with a threshold in their distance (currently set to 5 words) or
   - of PREP category, same preposition ("… for … and for …")
   - others of the same category

Of course, there are many exceptions to the requirement be of the same category. Some of them are accounted for in the current implementation, e.g.
-> the things you said and those said by me (second conjunct "those")
-> art. 132 and following (second conjunct = "following")
-> the opportunity of publishing or not (second conjunct = "no", i.e. a *PHRAS*)
-> telephones or other equivalent devices (second conjunct = "other", i.e. an *ADJ*)

In the case of verbs, there are preferences concerning the 'mood' of the conjuncts. Moreover, if the latter is an auxiliary, the first conjunct should also be such, and viceversa. Just in case a corresponding first conjunct is not found, the constraint on the auxiliaries is relaxed.

Finally, sequences of conjuncts are also looked for by the parser (as in "The girl, the boy, and the lady").

Note that these are just a few examples of the various heuristics used to assess the structure of the conjunctive fragment.

### 3.4.3 The verbal dependents

One of the most relevant components of the parser is the one that takes care of assigning the grammatical relations to the verbal dependents. This component is based on a module that determines the correspondence between the obligatory dependents (complements, i.e. verbal arguments) licensed by a given verb and the actual dependents selected in the sentence. In order to guarantee both a compact representation of the argument structures and an explicit representation of the possible surface realization of these structures, we introduced a mechanism of transformation; it works on basic argumental structures and automatically generates the possible surface realizations. Moreover, for representing some linguistic regularities, the base structures are encoded in an 'inheritance hierarchy', which enabled us not to define twice the surface realization of the same argument. For instance, the grammatical relation 'subject' is defined in a single node of the hierarchy (named *SUBJ-VERBS*) and its definition (where it is specified that a subject can be a noun, a verb in the infinite, etc.) is inherited by all descendants of *SUBJ-VERBS* (among which transitives, intransitives, modals, movement verbs, etc.).

Briefly, the information actually used by the parser to perform the match is produced in the following way:

```
    For each node NV in the hierarchy:
          NV (verbal class)
           |
           | Access to the definition
           V
          NVB (local argumental structure)
           |
           | Inheritance
           V
          NVC (full argumental structure)
           |
           | Transformations
           V
          SNV+ (set of possible surface realizations)
```

The operations described in the previous diagram are executed once (off-line); consequently, when the parser is applied, the correspondence NV --> SNV+ is already available. Moreover, in the case of the introduction of a new node (an operation that in this stage of development is more common than it could seem, since the nodes of the hierarchy encode also verbal locutions as 'tener conto' - keep into account -), it is sufficient to 'compile' the hierarchy, obtaining in this way all possible surface realizations of the new class.

Each verb is associated with a set of *NV* (verbal classes); so, given a verb, it is immediate to determine all possible surface realizations. The task of the matching module is to choose, among all possible realizations, the one which corresponds best to the set of actual verbal dependents. Beyond the fact that this is a rather complex task (consider the presence of adjuncts), it also depends on a previous step, not less complex, i.e. the selection of the actual dependents of the verb.

#### 3.4.3.1 Subcategorization

The subcategorization classes of Italian verbs are stored in
*"*HOME-DIR*/SYSTEM/ITALIAN/KB-ITA/SUBCAT-KB-ITA/verbclass-ita.dat".*
Currently (January 2008) 418 Italian verbs have associated classes (which are not assumed to be exhaustive). The total number of classes associated with the verbs is 895 (an average of 2.14 classes per verb). However, it must be noted that for each of the 4940 verbs present in the dictionary, a class is given. But in the dictionary the only possibilities are *TRANS* (transitive yes), *INTRANS* (transitive no), and *REFL* (transitive rifl), and it is not possible to associate more than one class to a given verb. So, we have that accurate data are available for 418 verbs, while for the remaining 4522 only approximate data are present.

For English, all 4700 (approx) verbs have a class (in *"*HOME-DIR*/SYSTEM/ENGLISH/KB-ENG/SUBCAT-KB-ENG/verbclass-eng.dat")*, but they have been extracted automatically by converting the WordNet subcategorization classes, so that, due to possible mismatches in the conversion, no warranty does exist about their accuracy.

The definition of the subcategorization classes is in the file

"**HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/SUBCAT-KB-ALL/subc-hier*".
Currently (January 2008) 123 general (language-independent) classes are defined, 23 of which are 'abstract classes', i.e. no verb is assigned to these classes, but thy are used just for inheritance. For Italian, 42 classes are defined for verbal locutions (e.g. "farsi avanti"), while only two of them for English ("have regard" and "take into account").

#### 3.4.3.2 Transformations

As stated above, what is used in the matching against the actual verbal dependents found in the sentence is not the base subcategorization frame defined by the verbal class, but the set of surface patterns obtained via transformations. Currently, 15 transformations are in use (see **HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/SUBCAT-KB-ALL/transf-def*). In the current version of the parser they are language-independent, though some of them are blocked for some languages.

It is not possible to describe here in any detail the transformation process. In general, each transformation is associated with a subset of the verbal classes; it specifies how one or more of the original surface realization have to be changed as a consequence of the transformation. This process results in a class as *TRANS* (transitive verbs) having more than 20 surface realizations, depending on its being in the base form, or having been passivized, or having undergone pro-drop, etc. Obviously, the number of realizations may overcome the one of the transformations, since more than one transformation can be applied to the same class at the same time (e.g. *TRANS+PASSIVIZATION+INFINITIVIZATION*).

These operations lead (from the 167 base classes) to about 2560 surface realizations (not all different, of course). The resulting transformed classes are stored in the file *"*HOME-DIR*/SYSTEM/ALLLANG/KB-ALL/SUBCAT-KB-ALL/newhier.out*"), which is the result of the 'compilation' of the knowledge about verbal subcategorization.

#### 3.4.3.3 Determining the verbal dependents

The sentence is read left-to-right: for each (non auxiliary) verb that is found, the search of its dependents is carried out, according to the following steps:

- a possible subject preceding the verb is looked for.
  It is any element preceding the verb of category *NOUN, ADJ, ART (DEITT, INDEF, INTERR, DEMONS), NUM, PRON* (the latter with *LSUBJ*, i.e. nominative, feature), which is not yet linked to the tree.
- determine if there are clitics preceding the verb.
  One or two elements having the feature *CLITIC*.
- find out all dependents that follow the verb.
  They are all the elements still unlinked, which are collected until a 'barrier' element is found. barrier elements can be 'exclusive' (in case the barrier element is not included among the complements), 'inclusive' (if it is included), or 'skipping' (if some material after the barrier can still act as a complement of the current verb). The exclusive barriers are:
  . The relative pronouns
  . Coordinating conjunctions whose first conjunct is the verb under analysis
  . Verbs not in the infinite or gerund tense, unless the current verb admits direct discourse
  . A clitic (not enclitics)
  . The negative adverb 'non' (it is usually linked to a following verb)
  . [for Italian] The preposition 'da' (by) followed by a pronoun and by a past participle; it is assumed to be the agent complement of a reduced relative ('il documento da lui firmato' - the document by him signed, i.e. the document signed by him)
  . a comma, if followed by an unmarked nominal chunk (assumed to be the subject of an incoming verb)

  . an unmarked nominal chunk, if the current verb is the verb of a reduced relative clause
  The inclusive barriers are:
  . A gerund or an infinite verb
  . A finite verb, in case the current verb admits direct discourse
  . A subordinative conjunction
  . Some doubly linked relative pronouns (currently 'chi' - who)
  The skipping barriers are:
  . reduced relatives (in case there is after them an unmarked nominal chunk, from which the search can continue)
- look for the dependents preceding the verb
  Finally, all elements preceding the verb are collected until a pre-barrier is reached; pre-barriers are:
  . a (non auxiliary) verb
  . a relative pronoun
  . a subordinative conjuction

The set of elements collected in these four steps constitutes the set of dependents of the verb (both complements and adjuncts).

The only exception is given by unmarked elements separated by commas. If the sequence is longer than two elements, we assume that they are not all dependents on the verb, but only the first is, while the others are coordinated elements.

#### 3.4.3.4 The match

For this operation, we have at disposal:
- A verb
- Its dependents, i.e. the result of the previous step
- A set of possible surface argumental realizations (see the sections on subcategorization and transformations).

Now, a first substep aims at reducing the set of surface realization to be taken into account. This is obtained by observing the form of the verb (passive, infinite, with or without clitics) and selecting only the realizations compatible with this form (e.g. all realizations including in their sequence of derivation the *PASSIVIZATION* transformation can be excluded if the verb is not passive).

In the second substep, we cut off duplicate surface realizations (e.g. both *INFINITIVIZATION* and *NULL-SUBJ* produce the deletion of the subject, so the surface realizations are identical and it is useless to carry out the match twice. Furthermore, since the base matching procedure returns a single result for each realization, all realizations including both a subject and an object are duplicated (exchanging the order of VERB-SUBJ and VERB-OBJ; so that both matches can in principle be obtained; see below).

The match proper, between a single surface realization and the set of dependents must take into account the possible presence of both complements and adjuncts. Since this is a 'robust' parser, and no backtracking step is included for allowing to choose a different set of dependents, the match 'must' succeed. So, no argument really is obligatory, and the match can succeed even with missing arguments. The only exception are the verbal locutions, where the argument identifying the locution, labelled as *EMPTYCOMPL*, must appear: of course, the locution 'keep into account' cannot be hypothesized if 'into account' is missing. Then, among all the possible matches, the best one is obtained, according to criteria that are described below. The single match is based on an 'internal' preference: a match with an argument is preferred to a match with an adjunct.

At the end of all single matches, we have a set of results, composed of one match for each possible surface realization available for the verb. Now, the different hypotheses are compared applying a larger set of preference criteria.

They are the following (applied in this order):

1. A label of *VERB-ARG*LOCUT* type appears in the result: if there is a verbal locution associated with the verb, and the element that characterizes the locution (e.g. 'into account' in 'keep into account') is present, then that interpretation is chosen.
2. The label *EMPTYCOMPL* appears in the result (if there is a pseudo-reflexive interpretation as accorger-si - remark - it is preferred)
3. The subject is not assigned to a temporal expression (ex. "Questa sera arriva Martina" – This evening arrives Martina, i.e. Martina arrives)
4. The *VERB+MODAL-INDCOMPL* is assigned (i.e. the verbal argument of a modal has bee found
5. [for English] The subject does not follow the verb
6. The label *PRON-RMOD-LOC+METAPH* appears in the result (this label is used for clitics as 'CI sono' - There are)
7. The label *VERB-SUBJ+IMPERS* is not assigned to an enclitic (the cases of enclitic impersonal subject are rare)
8. The label *VERB-SUBJ+IMPERS* is assigned and the verb has preference for impersonals wrt reflexives
9. There are less adjuncts
10. The verb is imperative, and the label *VERB-SUBJ* does not appear in the result
11. There are less unassigned arguments
12. Less transformations have been applied
13. There is no competition between VERB-SUBJ and VERB-OBJ (there is just one unmarked case) and VERB-OBJ is assigned (pro-drop is preferred if applied to the subject)
14. There is competition between VERB-SUBJ and VERB-OBJ, but VERB-SUBJ is assigned to a dependent preceding the verb, while VERB-OBJ follows the verb (standard order).
15. There is competition between INDOBJ and OBJ, and the verb is a special predicative verb (as "dire" - tell, "spiegare" - explain, ...)
16. The VERB-OBJ precedes the VERB-PREDCOMPL-OBJ (if both are present).

### 3.4.4 The elements that are still unlinked

If there are words of the sentence which, at this stage, are still lacking the pointer to the parent (unlinked), some heuristic rules are applied to find a solution. These rules are applied in two steps (the rules of the first step are 'preferred' to the rules of the second step. The rules depend on the PoS of the word to attach.

First step:

| | |
|---|---|
| *ADJ* --> | An adjective is linked to the nearest preceding or following noun |
| *ADV* --> | An adverb is linked to the nearest preceding or following verb |
| *NOUN* --> | as ADV |
| *PRON* --> | If it is not a doubly linked relative pronoun (ex. 'chi' - 'who -) as ADV, otherwise it is linked to the first verb that follows the relative clause to which the pronoun belongs. |
| *PREP* --> | If it does not precede a relative pronoun (ex. 'di cui' - of which - 'per il quale' - for whom -), as ADV. Otherwise, it is linked to the first verb that follows it (hopefully, the main verb of the relative clause) |
| *CONJ* --> | A conjunction is linked to the nearest preceding verb |
| *INTERJ* –> | Attach it to the nearest verb |

Second step:

| | |
|---|---|
| *PREP* --> | It is linked to the nearest preceding noun |
| *ADV* --> | It is linked to non-verbal categories |
| *CONJ* --> | If it is a subordinative conjunction, it is linked to the first noun that precedes it, and that may govern it (i.e. skipping relative clauses and other subordinates). |

# 4. Other facilities

Answering *3 (Other Functions)* to the initial request of the system *(Which of the options you want to follow?*), a further request appears, concerning special functions available to the user.

The first of them (a) enables the user to check the presence in the dictionary of one or more words. S/he is requested to *"Write the words to analyze. Finish with a !"*. So any sequence of word forms may be written, and the system performs the morphological analysis on them. Note that this facility is not very flexible. So, if any of the words is not recognized, the system signals it, and then, if asked to continue, prints the result of the analysis just for the words preceding the unknown one. The output format is the one of the analyzer (§3.2), so it includes all feature names.

## 4.1 Evaluation

With the second answer (b) it is possible to count the number of errors of the Tagger and of the Parser. Of course, the evaluation is made comparing the output of the parser (file '.prs') with a file which is assumed to be correct (it must have the same name and extension '.man').

So, the process that produces the evaluation is the following:
I.   Carry out the standard parsing (answer 1 to the initial request)
II.  Copy the resulting 'XXX.prs' file into 'XXX.man'
III. Check and correct manually XXX.man
IV.  Carry out the parsing+comparison process (answer *3 and then b* to the initial request)

Of course, since XXX.man is assumed to be correct, the parser can be refined and re-evaluated without repeating the steps I, II, III.

The result of the comparison is contained in three new files:
- 'XXX.cmp'; This contains the actual result of the comparison. It is a copy of the manually correct file, where in each line where a disagreement has been found wrt the result of the parser, it has been added the sequence:
    *??? >>> error description*
  So, it is possible to check where the parser made an error. Note that the errors include the ones coming from the tagger, which are identified by:
    *??? >>> DIFFERENT SYNTINFOS*

- 'XXX.tag'; it summarizes the behaviour of the PoS Tagger, reporting, for each rule that has been applied in the analysis of the file the number of correct results and the number of errors. These data appear in a simple teble of the following form:

```
***********************************
       Rule Name           ok err
***********************************
 ADJ-ADV-NOUNR03E           1  0
 ADJ-ADV-PRONR02            3  0
 ADJ-NOUN-VERBR05           3  1
     …                      …  …
```

- 'XXX.err'; it contains a table synthesizing the result of the comparison. An example is reported below:

```
 --------------------------------------------------------------
  SENT   |lines|trace|punct** err **lkerr|trerr|puerr|syerr| %err |
 --------|-----|-----|-----**-----**-----|-----|-----|-----|------|
   1-  5 |   59|    2|    5**    4**    3|    1|    0|    1|  6.78|
   6- 10 |  116|    2|   15**   27**   25|    3|    6|    2| 23.28|
  11- 15 |   98|    6|   16**   12**    9|   12|    8|    3| 12.24|
  16- 20 |  221|   12|   17**   46**   43|   17|   11|    3| 20.81|
  21- 25 |  141|    8|   15**   23**   21|    6|    5|    2| 16.31|
  26- 30 |   94|    4|    6**   19**   14|   10|    2|    5| 20.21|
  31- 35 |  193|    2|   12**   25**   25|    4|    5|    0| 12.95|
  36- 40 |  194|   10|   14**   39**   33|   12|    6|    6| 20.10|
  41- 45 |  139|    5|   14**   23**   20|   10|    7|    3| 16.55|
  46- 50 |   65|    0|    5**    9**    8|    0|    1|    1| 13.85|
 --------|-----|-----|-----**-----**-----|-----|-----|-----|------|
  TOTAL  | 1320|   51|  119**  227**  201|   75|   51|   26| 17.20|
 --------------------------------------------------------------
```

where:

a. SENT is a sequential numbering of the sentences in the file (note that it always begins with 1, that is it is not related with the 'official' numbering defined by *INITIAL-SENT-NUMBER* (see §1.2). Note also that, in order to have compact tables, each line of the table corresponds to a group of 5 sentences.
b. lines: The number of lines (words, punctuation marks, etc.) of the 5 sentences.
c. trace: the total number of traces appearing in these sentences
d. punct: the number of punctuation marks (and text markers)
e. err: the total number of errors made by the parser (except the trace errors)
f. lkerr: the linking errors, i.e. the errors related to a wrong attachment to the parent (wrong parent, or wrong arc label)
g. trerr: errors in the traces. Note that the value in 'trerr' may be higher than the one in 'trace'. In fact the parser may have inserted some extra traces. Each added trace (which is not counted in 'trace') is a 'trerr'
h. puerr: errors in the linking of the punctuation marks
i. syerr: tagging errors (i.e. errors concerning the syntactic data in the line). It must be noted that when an error of this type occurs, a possible linking error in the same line is disregarded.
l. %err: The value obtained by dividing 'err' by ('lines' minus 'traces'), i.e. the percent of errors with respect to the number of lines, disregarding the traces.

The third answer (c), i.e. "*parsing evaluation on automatically tagged data*", carries out the same operations as the one explained above (with the same outputs), but without carrying out the morphological analysis and the PoS tagging steps. In other words, the operations start from an (already given) '.tb' file. Consequently, it is possible to correct manually the tagger output of a previous parsing attempt (i.e. '.tb') and apply this option, in order to determine the number of errors that are independent of the tagger errors.

## 4.2 PoS Tagger refinement

The fourth option (d), i.e. "*collect tagging rule statistics*", just ask the system to make the sum of the tagging statistics (files 'XXX.tag', see above) obtained in the standard evaluation (answer (b)). The data present in various '.tag' files are summed up, in order to get complete information about the behaviour of the tagging rules.

Since the goal is to get data from several files, the option of providing a single file name as input is not available. So, if one wants to execute this facility, it is necessary to initialize the "**HOME-DIR*/DATI/tagcorpus.dat*" file (see §1.2).