NATURAL LANGUAGE PROCESSING GROUP
UNIVERSITY OF TURIN

TULE Documentation

SENTENCE REPRESENTATION

TULE Internal Report #4, February 2011

Author: Leonardo Lesmo

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF BOXES

LIST OF TABLES

1. INTRODUCTION

The Turin University Parser (TUP) includes a number of modules, that implement a standard distinction of tasks that must be accomplished in order to get the interpretation of a text. The main modules that are relevant for the present report are the tokenizer, the morphological analyser, the POS tagger, the parser and the semantic interpreter. The first three of them are mainly treated here as a single module. The text that undergoes the various processing stages is represented in various ways, according to the needs of the different modules and to the requirement that at some stages the results be human-readable. Some of these representation are internal to the modules, while others have been designed in order to store intermediate results into external files. This "blackboard" approach has the twofold goal of reducing, for large texts, the need to keep all data in memory, and of making the intermediate results available for inspection by humans.

This report describes the different representations. However, it must be observed that I do not describe here the features of the morphological/syntactic formalism (i.e. the various syntactic categories or the organization of the dependency links), since this information can be found in other reports (in particular, the Linguistic Notes).

Noneteheless, the task is not simple, since there are many ways that have been used to represent the structure; in particular:

a) The system-internal structure of the POS tagged sentences (i.e. the Lisp organization of the sequence of morphological data associated with the tokens composing the sentence)

b) The external structure of the POS tagged sentences (the one appearing in the output, human-readable, files)

c) The system-internal structure of the links that connect the tokens in the parsed sentence

d) The external representation of these links. This, together with the one of (b) above, correspond to the standard TUT Treebank format

e) A so-called "flat Attribute-Value Matrix" representation. This is isomorphic to (d), but includes information about the features (morphological and structural), which have been omitted in (d) in order to make it more readable

f) A real Attribut-Value Matrix representation, where the tree is represented as nesting of subtrees, instead of as separate lines whose parent is identified by the line number.

For each of them, this report describes the structure and lists a set of procedures useful to access the data and to navigate on the tree structure. Note that the tree is built in different phases so, in some cases it is only partially built. Consequently, at some stages the sentence must be viewed not as a tree but just as a sequence of lines. Note also that in this version of the report, I omit the representations that precede (a) above. It is clear that, before the POS tagger has completed its task, the input tokens are ambiguous under various respect. The complex description of the way this ambiguity is represented is delayed to future versions.

Finally, I want to observe that the various modules operate in sequence. It is obvious that this is not cognitively plausible, but, as the reader will realize, the process is complex enough even if we disregard the feedback that later modules can provide to earlier ones (e.g. semantics to syntax).

2. TEXTS AND SENTENCES

The goal of this section is to give an idea about the way the text analysis is organized, and how and where the different formats, that will be described in the following chapters enter into play. A short note about the segmentation of texts into sentences is also included.

The first thing to keep in mind is that TULE operations are associated with a *SYSTEM-CONTEXT*. This is a global variable that encodes the "mode of operation" of the system. Basically, there are two main contexts, i.e. TULE and ATLAS. The first one is related to parsing and Treebank development, the second one with full sentence interpretation (including semantic processing).

2.1. *SYSTEM-CONTEXT*=tule

The TULE context is used to support the development of the Turin University Treebank (TUT) and the refinements of the parser that are based on the tests made on TUT. These refinements are especially made in case TUT is extended with new data, so that new phenomena occur, that must be taken into account. In this way the parser coverage and precision is incrementally extended.

In all operations made in this context, the TUT formats described in Ch.3 are used. Actually, AVM formats (Ch.4) were defined when semantics entered into play, so they come after the original TUT-based organization. Nonetheless, there are two ways to provide the input, i.e. via a string or via a file. The first option is devoted to texts given by keyboard and are mostly used to demonstrate the behaviour of the parser. The second way is used for actual treebank development.

It is also possible to make some automatic comparison between the output of the parser and a manually annotated text. This is useful for the manual tuning and refinement of the parser. All of these operations are carried out on the TUT format of Ch.3.

In Fig.1 the various pipelines are depicted. The first of them (thin arrows) starts from an input string and produces a parsed text, i.e. one or more syntactic trees: the result is displayed on the screen or returned to the process which invoked the parser. The second possibility is to take the raw data from a file (.x in the figure, since any extension is allowed) in order to produce a .prs file. In the figure, the intermediate result .tb is shown. This is the result of the first three processes (tokenization, morphology, POS tagging), but do not include tree structures (see Ch.3). The process can be continued in case parser evaluation and refinement is the final goal. In this case, we start again from the .x file, but the .prs file is used by a process that compares the parsed text with the manual annotation of the same text. Of course, these data must be made available to the comparison in another file (.man in the figure).

Actually, some more files are produced as side effects during the process. I list them here, but they are not used as input by other processes in the pipeline:

.dis: This contains the POS disambiguation rules that have been used to select the POS of each word
.cmp: result of the comparison between the .prs and the .man. Any difference is marked by a comment in the involved line
.err: The total number of errors (split in some categories) after processing a given file
.sim: the sequence of lemmata associated with the word of the text, according with the POS tagger results.
.tag: A summary of the applied POS tagging rules, recording the number of successful application and the number of errors
.tball: The Morphosyntactic features associate with all possible readings of a word

Finally, note that the term "variable" (see the Legenda of fig.1) is intended to mean that the intermediate result in question is handled by an external procedure that stores the result of the previous module in the variable and provides the next module with that variable as an argument.

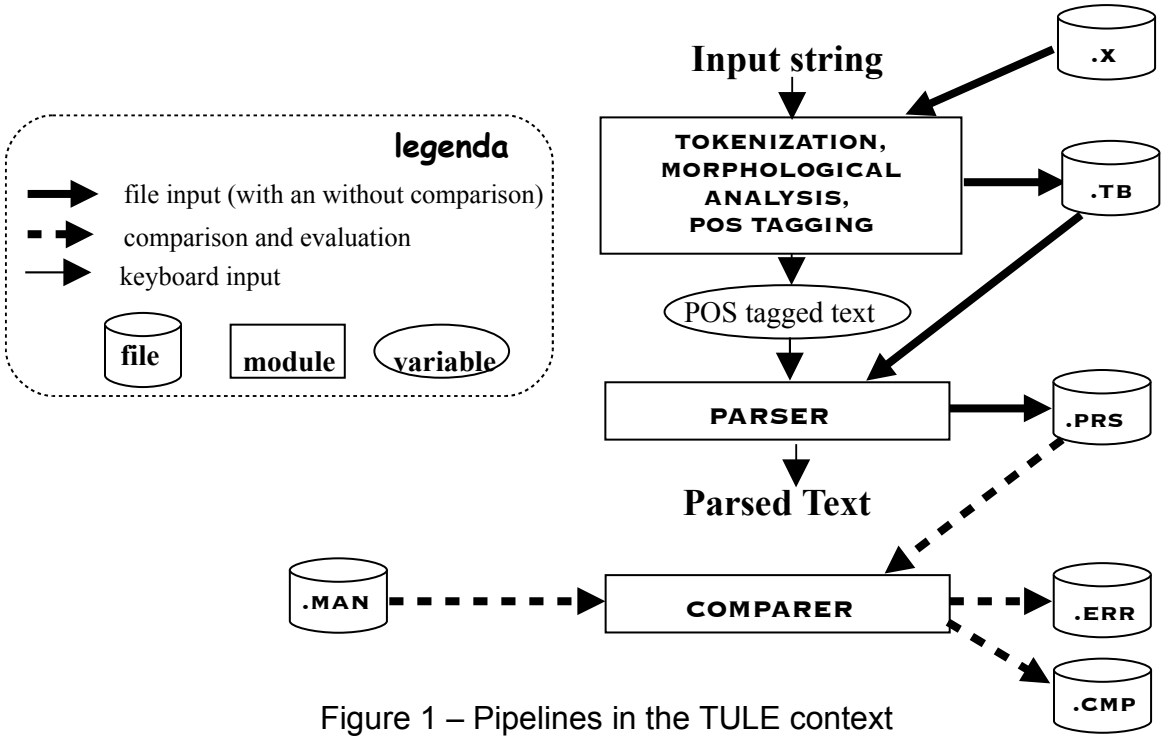

Figure 1 – Pipelines in the TULE context

2.2. *SYSTEM-CONTEXT*=atlas

The ATLAS context is used to face the problems associated with the development of the ATLAS project. The goal of this project is the translation from Italian to the Italian Sign Language of the deaf, but what is relevant here is that the translation is accomplished by means of a full ontology-based understanding of the input.

Originally, the idea was to work on single sentences, and the defined format was AVM, in order to cover easily the semantic annotation, by adding new features more freely. But after some time, we realized that full text analysis was required to carry out more extensive tests. Now, when the *atlas* context is entered, the system asks the user if the processing concerns a single sentence (which could, in principle, also be stored in a file) or a file that can include more than one sentence. The first mode is called "interactive", the second "file".

Thi visible effects are the following:

- In interactive mode, the different intermediate representation are shown on the screen. The amount of printing can be controlled by the user, but the final result is shown anyway.

  In the "interactive manual" mode, the sentence is given via keyboard and the screen visualization is the only result. In the "interactive with file" mode the screen visualization occurs as above, but the result of the analysis are also stored in files, with suitable extensions. The main reason for the existence of the "interactive manual" mode is that it takes as input a string, so it is more suitable for any kind of direct communication (e.g. as a server for interpretation).

- In the file mode, no screen output is provided: the results go into files; in particular, if the input file (whose name is asked to the user) is example.xxx (where xxx is any extension and can also be missing), the results are:

  a. example.tb (the same as for TUT)
  b. example.dis (the same as for TUT)
  c. example.tball (the same as for TUT)
  d. example.prs (the same as for TUT)
  e. example.sim (the same as for TUT)
  f. example.fav (representation in flat AVM: §4.1 and *Box 3*)[1]
  g. example.avm (representation in tree-structured AVM: §4.2 and *Box 4*)
  h. example.svm (representation in simplified tree-structured AVM: *Box 5*)

Internally, two different pipelines are used. In the interactive mode, the parser works directly on a flat AVM representation of the sentence, making the whole process more homogeneous. The TUT format is maintained just for the file outputs (if "interactive with file" is the case at hand). The actual result of the POS tagger is a sentence in flat AVM format, which is then passed to the parser. The result of the parser is then converted from flat AVM to tree-structured AVM and fed to the semantic interpreter. In other words, in the interactive mode the various stages are organized on the basis of assignment of values (different formats) to variables.

On the contrary, in the file mode the data are exchanged via files. This enables the system to handle large amounts of data. The ".tb" and ".prs" are produced in the usual way and the ".prs" is used as input for the semantic interpretation. The file content is translated into flat AVM and tree-structured AVM and the semantic interpretation is started up. All of this is synthesized in fig.2.

2.3. Text segmentation in sentences

This short paragraph has the goal to provide the reader with synthetic information about the way the text is split in different sentences. Although in most cases the solution is simple (e.g. in case a period, a question mark, etc. occurs at the end), in other cases different choices are possible. This may happen in case a colon or a semicolon occurs in the text. The basic idea that drove our search for a solution is that all sentences must have a single root in the parse tree. This enabled us to keep apart the cases in which what appears

[1] The only difference between the formats described in Ch.4 (actual internal results) and what appears in this and the next two files is that, in printing the files, the sentence headings are added for the sake of readability.

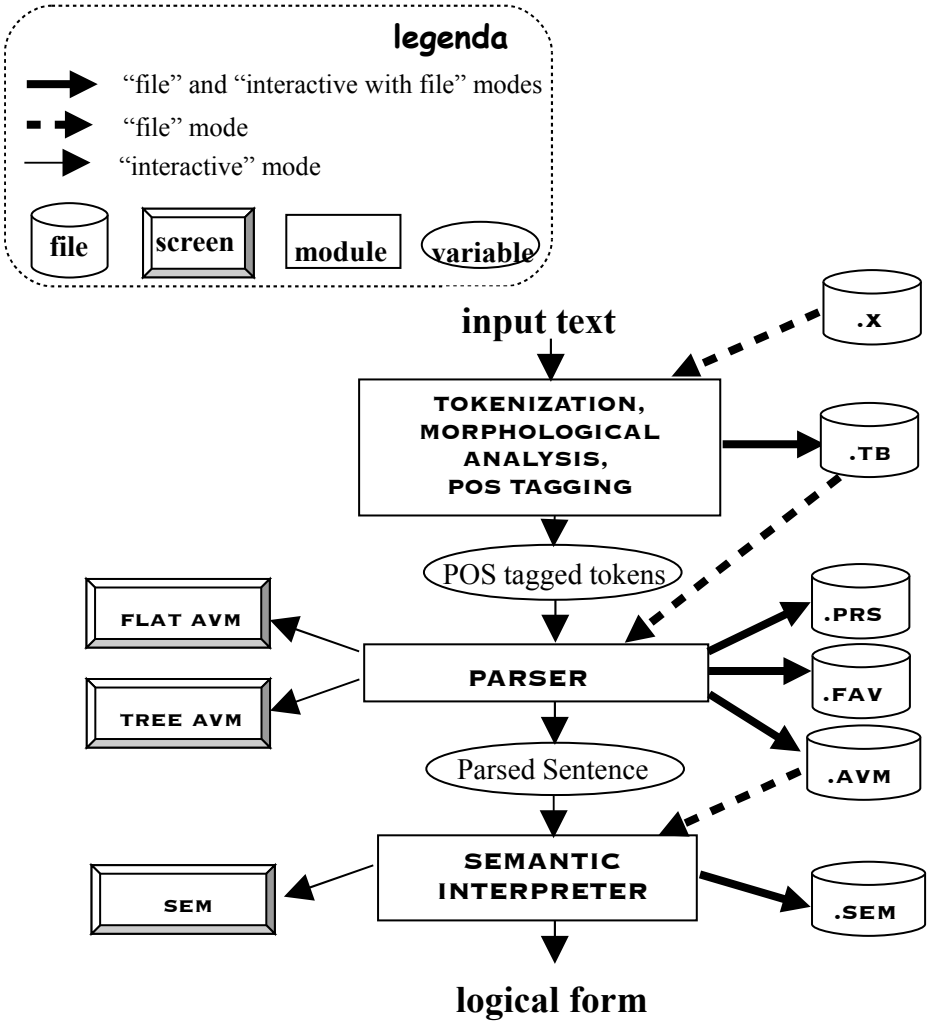

Figure 2 – Pipelines in the ATLAS context

in the two portions (before after, say, the colon) can be merged in a single tree, from cases where different trees are involved. Since this report addresses the representations and not text segmentation, I just report two examples that give an idea of the solution, without entering in greater details.

a) "The winners are:
   - The friend of Mike
   - The son of Jane
   - The organizer of the party"

   In this case, we assume a single sentence: the items in the list are taken as the subject of the verb and the dash is considered as a conjunction

b) "There are many things to do:
   - Mary must clean the dining room
   - Henry will wash the dishes

   In this second case, we assume three different sentences are involved, and the colon acts as a sentence terminator.

Finally, I want to note that this solution does not work for multiple sentences enclosed in parentheses. Since the parentheses are part of a larger sentence, in case more sentences

appear inside the parenthese we still have multiple roots. We do not have currently a solution for this. Suggestions are welcome.

3. THE TURIN UNIVERSITY TREEBANK (TUT) FORMATS

3.1. The external TUT format

In the TUT formats, the sentences are kept apart in two different ways, according to the representation being *External* or *Internal*. In *external* representation, the sentences are separated by *Heading lines*. Each of them is composed of a sequence of asterisks, the word *frase* (Italian for "sentence"), a form *TextId-N* (where *TextId* is a *Text Identifier* – any string – and *N* is a sentence number, i.e. an integer specifying the position of the sentence inside the text), and another sequence of asterisks (see Box 1).[2]

The actual data lines are given as text lines (UTF-8 encoding). There are two formats, one for input tokens and one for traces. The standard form is:

"3 partirà (PARTIRE VERB MAIN IND FUT INTRANS 3 SING) [0;TOP-VERB]"

Trace lines include a co-reference pointer of the form [Xs], where X is a line number and s is the trace type (not in the example of Box 1):

"15.10 t [14p] (AUTO NOUN COMMON F ALLVAL) [15;VERB-OBJ/VERB-SUBJ]

where line 14 is the one associated with the referent of the trace (i.e. the actual "auto" line). Note that the morphosyntactic infos in parentheses are redundant, being the same of the referent line: they are included just for readability purposes. Currently, there are three types of traces:

```
********** Frase EXAMPLE-1 **********
1 Domani (DOMANI ADV TIME) [3;ADVB-RMOD-TIME]
2 Maria (MARIA NOUN PROPER F SING ££NAME) [3;VERB-SUBJ]
3 partirà (PARTIRE VERB MAIN IND FUT INTRANS 3 SING) [0;TOP-VERB]
4 per (PER PREP MONO) [3;RMOD]
5 Roma (ROMA NOUN PROPER F SING ££CITY) [4;PREP-ARG]
6 . (#\. PUNCT) [3;END]

********** Frase EXAMPLE-2 **********
1 Prenderà (PRENDERE VERB MAIN IND FUT TRANS 3 SING) [0;TOP-VERB]
1.10 t [] (GENERIC-T PRON PERS ALLVAL ALLVAL ALLVAL) [1;VERB-SUBJ]
2 il (IL ART DEF M SING) [1;VERB-OBJ]
3 primo (PRIMO ADJ ORDIN M SING 1) [4;ADJC+ORDIN-RMOD]
4 treno (TRENO NOUN COMMON M SING) [2;DET+DEF-ARG]
5 del (DI PREP MONO) [4;PREP-RMOD]
5.1 del (IL ART DEF M SING) [5;PREP-ARG]
6 mattino (MATTINO NOUN COMMON M SING) [5.1;DET+DEF-ARG]
7 . (#\. PUNCT) [1;END]
```

Box 1 – *External* TUT format for a text including two sentences:
"Domani Maria parte per Roma. Prenderà il primo treno del mattino"

[2] The Text Identifier (EXAMPLE in Box 1) and the Initial Sentence Number (1 in Box 1) must be given in the input file (.x) in Figg.1 and 2

(Tomorrow, Maria will leave to Rome. She will take the first train of the morning)

- *f* : full traces, where the referent is the entire subtree of the token of the referenced line
- *p*: partial traces, where the referent is the entire subtree, excluding the portion including the trace. This is the case for reduced relative clauses.
- *w*: word traces, where the referent is the single referenced line

The square brackets may be empty (as in line 1.10 of the second sentence in *Box1*), when the referent is unknown or occurs in a different sentence. In the example, the referent is *Maria*, but we do not currently cope with inter-sentence co-references.

Each data line includes:

- The position of the token in sentence (line number or *index*). This can be an integer or a pair of integers separated by a dot (i.e. N or N.M). In *Box 1*, we see the two cases where non-atomic line numbers are used: traces (where the second element of the pair is an integer starting from 10), and compound items, as the Italian "preposizioni articolate" (preposition including an article, as "del"= di+il, i.e. "of the"; an English example is *cannot*). In this case the second component of the line identifier starts from 1.
- the input word or character (case-sensitive). This cannot be read by standard Lisp read macro, since it can be a special char (as the periods closing the two sentences of *Box 1*).
- a list of syntactic infos. This is in Lisp-readable format, so characters are preceded by \#

  The list includes:

  - The normalized form (e.g. infinite for verbs) of the word
  - The syntactic category
  - The syntactic subtype (sub-category)
  - Other infos depending on category and subtype (e.g. tense for verbs)
- A specification of the parent(s) included in brackets. The formats is [parentpos;label]. The governor of the current word is the token whose position is *parentpos*
- A comment, which must be preceded by one or more question marks;

  Ex. "??? This is a comment"

We repeat here that the morphosyntactic infos appearing in parentheses are associated with features whose name does not appear in the lines. This is easily solved by humans (who are the primary addressees of the representation), but make the quasi-positional notation a bit harder for the machine.

3.2. The internal TUT format

The represented data are the same as the ones described above, but they are complex LISP s-expressions, i.e. lists of lists. In particular, the text is represented as two separate and parallel lists: the first one includes the morphosyntactic information, while the second one includes the links, i.e. the data about the arcs connecting the tokens in the dependency tree (see *Box 2*). The main reason for keeping lines and links in different lists is mainly

historical. The first list is the output of the POS tagger, while the second one is filled by the parser. So, it seemed reasonable to keep the POS tagger output unchanged and to add a new list that contains the result of the parsing process. Unfortunately, this was correct before we started working on traces, but the presence of traces affect the POS tagger output, since some lines are added. However, changing the original decision would involve substantial modifications to a large portion of the parser, so we still keep the lists separate. The contents of the two lists are shown in Box 2.

```
Lines:
( ( (1 |Domani| (DOMANI ADV TIME) nil nil)
    (2 |Maria| (MARIA NOUN PROPER F SING ££NAME) nil nil)
    (3 |partirà| (PARTIRE VERB MAIN IND FUT INTRANS 3 SING) nil nil)
    (4 |per| (PER PREP MONO) nil nil)
    (5 |Roma| (ROMA NOUN PROPER F SING ££CITY) nil nil)
    (6 |.| (#\. PUNCT) nil nil)
  )
  ( (1 |Prenderà| (PRENDERE VERB MAIN IND FUT TRANS 3 SING) nil nil
    ( (1 10) |t| (GENERIC-T PRON PERS ALLVAL ALLVAL ALLVAL) empty nil)
    (2 |il| (IL ART DEF M SING) nil nil)
    (3 |primo| (PRIMO ADJ ORDIN M SING 1) nil nil)
    (4 |treno| (TRENO NOUN COMMON M SING) nil nil)
    (5 |del| (DI PREP MONO) nil nil)
    ( (5 1) |del| (IL ART DEF M SING) nil nil)
    (6 |mattino| (MATTINO NOUN COMMON M SING) nil nil)
    (7 |.| (#\. PUNCT) nil nil)
   )
)
Links:
( ( (3 ADVB-RMOD-TIME)
    (3 VERB-SUBJ)
    (0 TOP-VERB)
    (3 RMOD)
    (4 PREP-ARG)
    (3 END)
  )
  ( (0 TOP-VERB)
    (1 VERB-SUBJ)
    (1 VERB-OBJ)
    (4 ADJC+ORDIN-RMOD)
    (2 DET+DEF-ARG)
    (4 PREP-RMOD)
    (5 PREP-ARG)
    ( (5 1) DET+DEF-ARG)
    (1 END)
   )
)
```

Box 2 – *Internal* TUT format for a text including two sentences:
"Domani Maria parte per Roma. Prenderà il primo treno del mattino"
(Tomorrow, Maria will leave to Rome. She will take the first train of the morning)

Most data (in particular the morphosyntactic information) are similar to the external format (but the heading lines are not present in this representation). The pipes around the "input token" are used to force Lisp to maintain case information and to accept any sequence (as a period) as a valid Lisp "atom" (i.e. base expression). The two nil's at the end of the lines are associated with trace data and with possible comments. The defined functions that can be used to retrieve the data in this representation are reported in Appendix A.

4. THE ATTRIBUTE-VALUE MATRIX (AVM) FORMATS

The Attribute-Value Matrix (henceforth AVM) representation was introduced in order to make easier the inspection of the data. AVM is a rather standard representation, where structured data are organised in pairs <attribute, value>. A complex structure appears as follows:

< <Attribute1, Value1>, <Attribute2, Value2> .... <AttributeN, valueN> >

Since a Value can in its turn be an AVM, this representation is especially useful for representing trees. The advantage of AVM with respect to TUT is that the feature names appear explicitly, so that it is possible to get, for instance, the *PERSON* of a verb without knowing its position in the list of morphosyntactic values. However, as one can realize by having a look to the contents of *Box 3*, it is much heavier than the (external) TUT format.

4.1. The flat AVM format

In *Box 3*, I show what I call *flat* AVM format. It keeps the separation in lines, as the TUT formats do. Consequently, the tree is still represented by means of "pointer-to-parent" links. The line index is associated with the *POSIT* feature, and the input token with *FORM*. The rest of the data are grouped in four sublists: *SYN*, *SEM*, *TREE*, and *COREF*. The first one includes the morphosyntactic features, i.e.

- LEMMA
- CAT (the syntactic category)
- TYPE (the syntactic subcategory)
- GENDER
- NUMBER
- PERSON
- CASES (for pronouns: LSUBJ, LOBJ, OBL or more than one, ex. LSUBJ+LOBJ)
- MOOD
- TENSE
- TRANS (rough subcategorization of verbs: TRANS, INTRANS, REFL)
- V-DERIV (for nouns derived from verbs, the verb; for "failure", V-DERIV=fail)
- V-TRANS (for the verb from which the noun is derived: TRANS, INTRANS, REFL)
- LOCUTION (true if the token is part of a multi-word, absent or NIL otherwise)[3]
- CLITIC (for pronouns: true if the pronoun is a clitic, absent or NIL otherwise)

```
( ( ((POSIT 1) (FORM |Domani|)
      (SYN ((LEMMA DOMANI) (CAT ADV))) (SEM NIL)
      (TREE ((PARENT 3) (LABEL ADVB-RMOD-TIME))))
   ((POSIT 2) (FORM |Maria|)
```

[3] In general, the absence of a feature is equivalent to the specification that its value is NIL. In *Box 3*, I have included the NIL values just for the *SEM* feature, to recall that they will be filled by the semantic interpretation functions.

```
    (SYN ((LEMMA MARIA) (CAT NOUN) (TYPE PROPER) (GENDER F)(NUMBER ALLVAL)))
    (SEM ((SEMTYPE ££NAME)))
    (TREE ((PARENT 3) (LABEL VERB-SUBJ))))
 ((POSIT 3) (FORM |partirà|)
    (SYN ((LEMMA PARTIRE) (CAT VERB) (MOOD IND) (TENSE FUT) (TRANS INTRANS)
          (PERSON 3) (NUMBER SING))) (SEM NIL)
    (TREE ((PARENT 0) (LABEL TOP-VERB))))
 ((POSIT 4) (FORM |per|)
    (SYN ((LEMMA PER) (CAT PREP))) (SEM NIL) (TREE ((PARENT 3) (LABEL RMOD))))
 ((POSIT 5) (FORM |Roma|)
    (SYN ((LEMMA ROMA) (CAT NOUN) (TYPE PROPER) (GENDER F) (NUMBER ALLVAL)))
    (SEM ((SEMTYPE ££CITY)))
    (TREE ((PARENT 4) (LABEL PREP-ARG))))
 ((POSIT 6) (FORM \.)
    (SYN ((LEMMA #\.) (CAT PUNCT))) (SEM NIL) (TREE ((PARENT 3) (LABEL END))))
)
( ((POSIT 1) (FORM |Prenderà|)
    (SYN ((LEMMA PRENDERE) (CAT VERB) (MOOD IND) (TENSE FUT)
          (TRANS TRANS) (PERSON 3) (NUMBER SING))) (SEM NIL)
    (TREE ((PARENT 0) (LABEL TOP-VERB))))
 ((POSIT (1 10)) (FORM |t|)
    (SYN ((LEMMA GENERIC-T) (CAT PRON) (TYPE PERS) (GENDER ALLVAL)
          (NUMBER ALLVAL) (CASE ALLVAL))) (SEM NIL)
    (TREE ((PARENT 1) (LABEL VERB-SUBJ))) (COREF NIL))
 ((POSIT 2) (FORM |il|)
     (SYN ((LEMMA IL) (CAT ART) (TYPE DEF) (GENDER M) (NUMBER SING))) (SEM NIL)
     (TREE ((PARENT 1) (LABEL VERB-OBJ))))
 ((POSIT 3) (FORM |primo|)
    (SYN ((LEMMA PRIMO) (CAT ADJ) (TYPE ORDIN) (GENDER M) (NUMBER SING)))
          (SEM NIL) (TREE ((PARENT 4) (LABEL ADJC+ORDIN-RMOD))))
 ((POSIT 4) (FORM |treno|)
    (SYN ((LEMMA TRENO) (CAT NOUN) (TYPE COMMON) (GENDER M)
          (NUMBER SING)))
    (SEM NIL) (TREE ((PARENT 2) (LABEL DET+DEF-ARG))))
 ((POSIT 5) (FORM |del|)
    (SYN ((LEMMA DI) (CAT PREP))) (SEM NIL)
    (TREE ((PARENT 4) (LABEL PREP-RMOD))))
 ((POSIT (5 1)) (FORM |del|)
    (SYN ((LEMMA IL) (CAT ART) (TYPE DEF) (GENDER M) (NUMBER SING))) (SEM NIL)
    (TREE ((PARENT 5) (LABEL PREP-ARG))))
 ((POSIT 6) (FORM |mattino|)
    (SYN ((LEMMA MATTINO) (CAT NOUN) (TYPE COMMON) (GENDER M)
          (NUMBER SING))) (SEM NIL)
    (TREE ((PARENT (5 1)) (LABEL DET+DEF-ARG))))
 ((POSIT 7) (FORM \.)
    (SYN ((LEMMA #\.) (CAT PUNCT))) (SEM NIL)
    (TREE ((PARENT 1) (LABEL END))))
) )
```

Box 3 – *Flat AVM* format for a text including two sentences:
"Domani Maria parte per Roma. Prenderà il primo treno del mattino"
(Tomorrow, Maria will leave to Rome. She will take the first train of the morning)

The *SEM* sub-AVM includes:

- SEMTYPE (semantic class of proper names, as ££CITY for Roma)
- LEXMEAN (lexical meaning, expressed as a pointer to an ontology)

- IDENT (for proper names and some other items, reference to an individual in the KB)
- VALUE (for numbers, also when written in letters, the numerical value)
- YEAR (for abbreviated years, as '98, the actual value, i.e. 1998)
- DAYHOUR (for times expressed in the form 15:30 or 15:30:28; the hour - 15 in the example)
- DAYMINUTE (in the case above, the minute – 30 in the example)
- DAYSECOND (in the case above, just for the second form, the second – 28 in the example)

The *TREE* sub-AVM includes:
- PARENT
- LABEL

The *COREF* sub-AVM includes:
- LINE (the index of the referent line)
- CTYPE (f, p, w, for "full", "partial" or "word")

In Appendix A (§A.2), some functions for handling the flat AVM format are listed.

4.2. The tree-structured AVM format

This representation was included mainly to respect the standards of AVM. In particular, each parse tree is a single AVM with two top-level features, i.e. HEAD and DEPENDENTS. The *HEAD* is almost the same as a flat AVM line, with the exception that the *TREE* sub-feature has been replaced by a single-valued *LINK* feature. This depends on the fact that the *PARENT* feature is no more needed, because the parent is identified by the position of the AVM inside a larger AVM. Actually, also the *POSITION* feature could be disposed of, but it has been kept in order to make easier the retrieval of the referent of traces.

The value of the *DEPENDENTS* feature is a list of AVM's, one for each subtree governed by the head. Among them, a dummy dependent has been included, i.e.(#/#), which marks the linear position of the head within the sequence of dependents.

A final note concerns the movements on the tree. Traversing the tree downwards is very easy, since it is sufficient to retrieve the *DEPENDENTS* attribute. Conversely, it is not possible to move upwards on the sole basis of a given subtree, because the parent is the head of the larger tree including the subtree under analysis. So, when a movement up is needed, the function that implements the movement must be provided with the subtree for which the parent is being sought and with the full tree of the sentence: the search must start from the root, and the whole tree must be traversed until the subtree under analysis is found among the dependents.

In *Box 4*, I show the tree-structured AVM for our two example sentence. Even without looking at the example carefully, it is possible to notice that the printed form is much harder to inspect than the previous ones. Only indentation can provide some help, but given the amount of reported information, also indentation, for complex trees, is only partially useful. For this reason, in order to support the manual check of the tree-structured AVM representation, I have introduced a kind of "simplified" tree-structured AVM (see *Box 5*). It has not been included among the various formats described herein, because it is just for output, and that representation is not used by any procedure of TULE.

```
( (HEAD
   ((FORM |partirà|) (POSITION 3)
    (SYN  ( (LEMMA PARTIRE) (CAT VERB) (MOOD IND) (TENSE FUT) (TRANS INTRANS)
            (PERSON 3) (NUMBER SING)))
```

```
   (LINK TOP-VERB) (SEM NIL)))
 (DEPENDENTS
  (  ( (HEAD
       ((FORM |Domani|) (POSITION 1) (SYN ((LEMMA DOMANI) (CAT ADV)))
        (LINK ADVB-RMOD-TIME) (SEM NIL)))
      (DEPENDENTS ((#\#))))
   ( (HEAD
      ((FORM |Maria|) (POSITION 2)
       (SYN ((LEMMA MARIA) (CAT NOUN) (TYPE PROPER) (GENDER F) (NUMBER ALLVAL)))
       (LINK VERB-SUBJ) (SEM ((SEMTYPE ££NAME)))))
     (DEPENDENTS ((#\#))))
     (#\#)
     ( (HEAD
        ((FORM |per|) (POSITION 4) (SYN ((LEMMA PER) (CAT PREP))) (LINK RMOD) (SEM NIL)))
       (DEPENDENTS
        ((#\#)
        ( (HEAD
          ((FORM |Roma|) (POSITION 5)
           (SYN ((LEMMA ROMA) (CAT NOUN) (TYPE PROPER) (GENDER F) (NUMBER ALLVAL)))
           (LINK PREP-ARG) (SEM ((SEMTYPE ££CITY)))))
         (DEPENDENTS ((#\#)))))))
 ((HEAD
   ((FORM \.) (POSITION 6) (SYN ((LEMMA #\.) (CAT PUNCT))) (LINK END)
    (SEM NIL)))
  (DEPENDENTS ((#\#)))))))
((HEAD
 ((FORM |Prenderà|) (POSITION 1)
  (SYN ((LEMMA PRENDERE) (CAT VERB) (MOOD IND) (TENSE FUT) (TRANS TRANS)
          (PERSON 3) (NUMBER SING)))
  (LINK TOP-VERB) (SEM NIL)))
(DEPENDENTS
 ((#\#)
  ((HEAD
   ((FORM |t|) (POSITION (1 10))
    (SYN
     ((LEMMA GENERIC-T) (CAT PRON) (TYPE PERS) (GENDER ALLVAL)
      (NUMBER ALLVAL) (CASE ALLVAL)))
    (LINK VERB-SUBJ) (SEM NIL)))
  (DEPENDENTS ((#\#))))
  ((HEAD
   ((FORM |il|) (POSITION 2)
    (SYN ((LEMMA IL) (CAT ART) (TYPE DEF) (GENDER M) (NUMBER SING)))
    (LINK VERB-OBJ) (SEM NIL)))
   (DEPENDENTS
    ((#\#)
     ((HEAD
      ((FORM |treno|) (POSITION 4)
       (SYN
        ((LEMMA TRENO) (CAT NOUN) (TYPE COMMON) (GENDER M) (NUMBER SING)))
       (LINK DET+DEF-ARG) (SEM NIL)))
      (DEPENDENTS
       (((HEAD
         ((FORM |primo|) (POSITION 3)
          (SYN  ((LEMMA PRIMO) (CAT ADJ) (TYPE ORDIN) (GENDER M)  (NUMBER SING)))
          (LINK ADJC+ORDIN-RMOD) (SEM NIL)))
        (DEPENDENTS ((#\#))))
        (#\#)
        ((HEAD
          ((FORM |del|) (POSITION 5) (SYN ((LEMMA DI) (CAT PREP)))
```

```
        (LINK PREP-RMOD) (SEM NIL)))
      (DEPENDENTS
       ((#\#)
        ((HEAD
          ((FORM |del|) (POSITION (5 1))
           (SYN ((LEMMA IL) (CAT ART) (TYPE DEF) (GENDER M) (NUMBER SING)))
           (LINK PREP-ARG) (SEM NIL)))
         (DEPENDENTS
          ((#\#)
           ((HEAD
             ((FORM |mattino|) (POSITION 6)
              (SYN
               ((LEMMA MATTINO) (CAT NOUN) (TYPE COMMON) (GENDER M)  (NUMBER SING)))
              (LINK DET+DEF-ARG) (SEM NIL)))
            (DEPENDENTS ((#\#)))))))))))
  ((HEAD
    ((FORM \.) (POSITION 7) (SYN ((LEMMA #\.) (CAT PUNCT))) (LINK END) (SEM NIL)))
   (DEPENDENTS ((#\#)))))))
)
```

Box 4 – *Tree-structured AVM* format for a text including two sentences:
"Domani Maria parte per Roma. Prenderà il primo treno del mattino"
(Tomorrow, Maria will leave to Rome. She will take the first train of the morning)

```
************** Frase EXAMPLE-1 **************
   (Domani 1 DOMANI ADV NIL NIL ADVB-RMOD-TIME)
   (Maria 2 MARIA NOUN PROPER NIL VERB-SUBJ)
(partirà 3 PARTIRE VERB NIL NIL TOP-VERB)
   (per 4 PER PREP NIL NIL RMOD)
      (Roma 5 ROMA NOUN PROPER NIL PREP-ARG)
   (. 6 . PUNCT NIL NIL END)

************** Frase EXAMPLE-2 **************
(Prenderà 1 PRENDERE VERB NIL NIL TOP-VERB)
   (t (1 10) GENERIC-T PRON PERS NIL VERB-SUBJ)
   (il 2 IL ART DEF NIL VERB-OBJ)
        (primo 3 PRIMO ADJ ORDIN NIL ADJC+ORDIN-RMOD)
     (treno 4 TRENO NOUN COMMON NIL DET+DEF-ARG)
        (del 5 DI PREP NIL NIL PREP-RMOD)
          (del (5 1) IL ART DEF NIL PREP-ARG)
            (mattino 6 MATTINO NOUN COMMON NIL DET+DEF-ARG)
   (. 7 . PUNCT NIL NIL END)DEF-ARG]
```

Box 5 – Simplified *tree-structured AVM* format for a text including two sentences:
"Domani Maria parte per Roma. Prenderà il primo treno del mattino"
(Tomorrow, Maria will leave to Rome. She will take the first train of the morning)

APPENDIX A: Functions for working on the representations

Before listing the various functions, it must be noted that, for the access to the TUT internal format (A.1) and to the AVM format (A.2) there is a more generic form. For instance there are *get-newtb-categ* and *get-flatavm-categ*, that retrieve the syntactic category from either the TUT or the flat AVM formats. However, *get-synt-categ* accomplishes the same task: it

evaluates one or the other of the basic forms according to the value of the global variable *TREE-FORMAT* that takes as possible values “avm” and “tut”.

A.1 Functions for accessing the TUT format

The next two functions work on the external TUT format

- **is-sentence-heading (line firstline?)**
  INPUT: line (any string);
  firstline? (boolean): true if this is the first line of a file
  OUTPUT (boolean): t if “line” is a heading line, false otherwise)
- **interp-newtb-line (line &optional ok-no-treeinfo)**
  Converts an external TUT line into an internal TUT line
  INPUT: line (a string having the format of external TUT line – see chapter 2)
  Ok-no-treeinfo: boolean: true if the infos about tree structure can be absent
  OUTPUT (list): “line” expressed in internal TUT format (§1.2)

The following functions work on the internal TUT format, i.e. on the output of *interp-newtb-line* above

- **index-precedes** (ind1 ind2)
  INPUT: ind1 and ind2 (two line indices, each of which is an integer or a pair of integers)
  OUTPUT (boolean): true if index ind1 precedes index ind2.
- **index-prevline** (ind1 ind2)
  INPUT: ind1 and ind2 (two line indices, each of which is an integer or a pair of integers)
  OUTPUT (boolean): true if the token t1 with index ind1 comes immediately before the token t2 with index ind2. Actually, some sub-components of t1 could occur in between. Ex: “nei prati” (in the fields): “in” (first component of “nei”) has index X, “i” (second component of “nei” has index (X 1), “prati” has index X+1: (index-prevline X X+1) = t, though (X 1) is between them.
- **same-linumb** (ind1 ind2)
  INPUT: ind1 and ind2 (two line indices, each of which is an integer or a pair of integers)
  OUTPUT (boolean): true if the token t1 with index ind1 is the same token as t2 with index ind2 or if they are subcomponents of the same token
- **pick-prevword** (index lines)
  INPUT: index (a line index)
  lines (a list of lines)
- OUTPUT (a line): the line that immediately precedes the one with index “index”
- **find-dependents** (headline alllines alllinks)
  INPUT: headline (a line in internal TUT format)
  alllines (all the lines of the sentence in which headline is assumed to occur)
  alllinks (all the links to parent of that sentence)
  OUTPUT (list of lines): all lines dependent on headline (i.e. such that the pointer to parent is equal to the line index of headline)
- **find-first-aux** (word prev)
  INPUT: word (a line in internal TUT format; it is assumed to be a verbal line)
  prev (a list of lines that are assumed to precede “word”)
  OUTPUT (a TUT line): the line of the first auxiliary of a sequence of auxiliaries that are assumed to be the auxiliaries of “word”. Only adverbs and quote symbols can occur between the main and its auxiliaries. Of course, this function assumes that the auxiliaries precede the main verb. If the verb has no auxiliaries, its line is returned.
- **is-a-newtb-trace?** (wdata)
  INPUT: wdata (a line in internal TUT format)
  OUTPUT (boolean): true if wdata is a trace

- **same-locution** (wdata1 wdata2)
  INPUT: wdata1 and wdata2 (two lines in internal TUT format)
  OUTPUT (boolean): true if wdata1 and wdata2 are parts of the same multi-word
- **is-newtb-gender** (val)
  INPUT: val (any object)
  OUTPUT (boolean): true if val is one of F, M, N, ALLVAL
- **is-newtb-number** (val)
  INPUT: val (any object)
  OUTPUT (boolean): true if val is one of SING, PL, ALLVAL
- **is-newtb-person** (val)
  INPUT: val (any object)
  OUTPUT (boolean): true if val is one of 1, 2, 3, ALLVAL

The following functions retrieve the value of the morphosyntactic features from a line. For all of them the INPUT (wdata)a is a line in internal TUT format.

- **get-newtb-numb** (wdata): Returns the line index
- **get-newtb-linumb** (wdata): As the previous one, but if the index is non-atomic (e.g. (15 1)), it returns just the first part (15)
- **get-newtb-syntinfo** (wdata): Returns the list of morphosyntactic infos
- **get-newtb-inpword** (wdata): Returns the (case sensitive) input token
- **get-newtb-word** (wdata): Returns the lemma
- **get-newtb-categ** (wdata): Returns the syntactic category (Part of Speech)
- **get-newtb-type** (wdata): Returns the syntactic subcategory
- **get-newtb-number** (wdata): Returns the syntactic number (SING, PL, ALLVAL)
- **get-newtb-gender** (wdata): Returns the syntactic gender (F, M, N, ALLVAL)
- **get-newtb-person** (wdata): Returns the syntactic person (1, 2, 3, ALLVAL)
- **get-newtb-mood** (wdata): Returns the mood of verbs (IND, INFINITE, PARTICIPLE, ...)
- **get-newtb-tense** (wdata): Returns the tense of verbs (PRES, PAST, FUT, ...)
- **get-newtb-cases** (wdata): Returns the case of pronouns (LSUBJ, LOBJ, LSUBJ+LOBJ, ...)
- **get-newtb-treeinfo** (wdata): Returns the link information (e.g. (3 VERB-SUBJ))
- **get-newtb-corefinfo** (wdata): Returns the index of the referent of a trace
- **get-newtb-semtype** (wdata): Returns the semantic class of a proper name (e.g. £ £CITY for Rome)
- **get-newtb-subcat** (wdata): Returns the rough subcategorization class of a verb (TRANS, INTRANS, REFL)
- **get-newtb-value** (wdata): Returns the numeric value of a number (the LEMMA is an alphanumeric item)
- **get-condit-newtb-value** (wdata): As above, but for non-numbers it returns -1
- **get-newtb-year** (wdata): Returns the year for reduced year forms (e.g. '98 → 1998)
- **get-newtb-vderiv** (wdata): Returns the verb, if any, from which a noun is derived
- **get-newtb-vtrans** (wdata): Returns the rough subcategorization of that verb
- **get-newtb-parent** (wdata): Returns the index of the governor of a token
- **get-newtb-label** (wdata): Returns the label of the arc linking a token to its governor
- **get-newtb-dayhour** (wdata): Returns the day hour for daytimes given in the form hh:mm:ss or hh:mm (15:25:37 → 15)
- **get-newtb-dayminute** (wdata): Returns the minute for daytimes given in the form hh:mm:ss or hh:mm (15:25:37 → 25)
- **get-newtb-daysecond** (wdata): Returns the second for daytimes given in the form hh:mm:ss or hh:mm (15:25:37 → 37)
- **change-newtb-type** (wdata newtype): This changes the value of the syntactic subcategory feature for wdata; the new value is given in "newtype"

A.2 Functions for accessing the flat AVM format

- **same-avm-line** (line1 line2)
- **is-a-flatavm-trace?** (wdata)
  INPUT: wdata (a line in flat AVM format)
  OUTPUT (boolean): true if wdata is a trace
- **get-flatavm-feat-val** (avm attr)
  INPUT: avm (an Attribute-Value Matrix)
  attr (a feature name)
  OUTPUT (a Lisp object): the value of the feature "attr" in "avm"
- **replace-flatavm-feat-val** (avm attr newval)
  INPUT: avm (an Attribute-Value Matrix)
  attr (a feature name)
  newval (a feature value)
  OUTPUT (an avm): the new avm obtained by replacing the value of the feature "attr" in
  "avm" with the value "newval"

- **get-newtb-numb** (wdata): Returns the line index
- **get-flatavm-linumb** (wdata)
  As the previous one, but if the index is non-atomic (e.g.(15 1)), it returns just the first part (15)
- **get- flatavm -syntinfo** (wdata)
- **get- flatavm -inpword** (wdata)
- **get- flatavm -word** (wdata)
- **get- flatavm -categ** (wdata)
- **get- flatavm -type** (wdata)
- **change- flatavm -type** (wdata newtype)
- **get- flatavm -number** (wdata)
- **get- flatavm -gender** (wdata)
- **get- flatavm -person** (wdata)
- **get- flatavm -mood** (wdata)
- **get- flatavm -tense** (wdata)
- **get- flatavm -cases** (wdata)
- **get- flatavm -treeinfo** (wdata)
- **get- flatavm -corefinfo** (wdata)
- **get- flatavm -semtype** (wdata)
- **get- flatavm -subcat** (wdata)
- **get- flatavm -value** (wdata)
- **get-condit- flatavm -value** (wdata)
- **get- flatavm -year** (wdata)
- **get- flatavm -vderiv** (wdata)
- **get- flatavm -vtrans** (wdata)
- **get- flatavm -parent** (wdata)
- **get- flatavm -label** (wdata)
- **get- flatavm -dayhour** (wdata)
- **get- flatavm -dayminute** (wdata)
- **get- flatavm -daysecond** (wdata)
- **get- flatavm -seminfo** (wdata)
- **get- flatavm -lexmean** (wdata)

A.3 Functions common to internal TUT and flat AVM

As noted above, all *get-newtb-X* and *get-flatavm-X* have a more generic form *get-synt-X*. Here, I list all of them without comments. Before them, I list some functions that do not have a *newtb* or *flatavm* correspondence.

- **line-not-trace** (lines)

INPUT: lines (a list of lines in iinternal TUT or flat AVM format)
OUTPUT (line): the first line of "lines" that is not a trace (if any).

A.4 Functions for accessing the tree-structured AVM format

The list that follows is similar to the previous ones. Note, however, that the name of the functions that retrieve the feature values include a "head" part. This is because the input, in this case, is not a line, but a full tree (avmtree), so that the relevant features are the ones of the head of the tree. It is also worth observing that retrieving the parent of a node of the tree (i.e. moving upwards) is a bit more difficult, since "avmtree" does not include the parent of its head.

- **get-actavm-head** (avmtree)
  Returns the head of the tree, i.e. a list having a structure similar to the one of a "flat" AVM line (see §... for details)
- **get-actavm-dependents** (avmtree)
  Returns a tree set, i.e. a list of AVM trees, which includes all subtrees of "avmtree"
- **get-actavm-deps-with-traces** (avmtree)
  As above, but, if "avmtree" refers to a "full" trace, returns the subtrees of the referent of the trace
- **get-true-deplines** (lines)
  "lines" is assumed to be a list of dependents. The function removes from "lines" the (#\#) form, that marks the position of the head
- **get-actavm-headform** (avmtree)
- **get-actavm-headnumb** (avmtree)
- **get-actavm-headlinumb** (avmtree)
- **get-actavm-headcoref** (avmtree)
- **get-actavm-headsyn** (avmtree)
- **get-actavm-headsem** (avmtree)
- **get-actavm-headlink** (avmtree)
- **get-actavm-headcorefline** (avmtree)
- **get-actavm-headcoreftype** (avmtree)
- **get-actavm-headcateg** (avmtree)
- **get-actavm-headtype** (avmtree)
- **get-actavm-headgender** (avmtree)
- **get-actavm-headnumber** (avmtree)
- **get-actavm-headperson** (avmtree)
- **get-actavm-headmood** (avmtree)
- **get-actavm-headcase** (avmtree)
- **get-actavm-headlexmean** (avmtree)
- **get-actavm-ext-headlexmean** (avmtree)
  As above, but skips a possible English question or tense marker (will, do, ...), i.e. it returns the meaning of the governed verb
- **get-actavm-headvalue** (avmtree)
- **get-actavm-headlexident** (avmtree)
- **get-actavm-head** (avmtree)
- **is-a-actavm-trace?** (avmtree)
- **find-actavm-parent** (localtree treeset)
  returns the subtree of one of the trees in "treeset", which has, as an immediate dependent, "localtree"
- **find-coreferent** (linenumb treeset)
  returns the subtree of one of the trees in "treeset", which has, as an immediate dependent, "localtree"
- **find-actavm-descendant** (feat-path feat-vals deps)

- **get-sentential-object** (avmtree)
  Returns the subtree which is the VERB-OBJ of the head if the head of that subtree is a verb, or if it is a conjunction or a preposition governing a verb (in which case it returns the verbal subtree governed by the conjunction or the preposition). If the head is a modal, then it returns the VERB+MODAL-INDCOMPL of the head. It skips the possible English question or tense marker, but in this case avmtree must be semantically annotated.
- **get-standard-object** (avmtree)
  Returns the subtree which is the VERB-OBJ of the head. Handles some locution cases
- **get-preposition-arg** (avmtree)
  Returns the subtree which is the PREP-ARG of the head. Handles preposition continuations (as “insieme a” – together with).
-