NATURAL LANGUAGE PROCESSING GROUP
UNIVERSITY OF TURIN

TULE
Documentation

TOKENIZATION AND MORPHOLOGICAL ANALYSIS

TULE Internal Report #2, January 2009
Author: Leonardo Lesmo

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF BOXES

LIST OF TABLES

1. INTRODUCTION

The morphological analyser takes as input a text and produces a sequence of words (each of which is possibly ambiguous).

It uses various knowledge sources, among which a dictionary and a suffix table. However, the analyser takes also care of the tokenization of the input text; this is the first step of the process and is carried out by means of an automaton that keeps apart various types of entries, as general words, proper names, dates, numbers, and so on.

The tokenizer returns tokens (which are possibly ambiguous with respect to the tokenizer categories). Each token is examined by the morphological analyser, that, in its turn, returns word hypotheses. While the tokens are just sequences of characters with an associated token category, the output includes lexical elements, that carry information about lexical category (Part of Speech) and feature values (as gender, number, tense, according to the specified category). A detailed description of the features is provided in http://www.di.unito.it/~tutreeb/ (Documents: Syntactic Categories).

The overall architecture of these first two steps of analysis is shown in Fig.1. In this report, we describe the two main processes appearing in Fig.1, paying special attention to the "Token Automaton" and to the "Suffixes" managed by the morphological analyser.

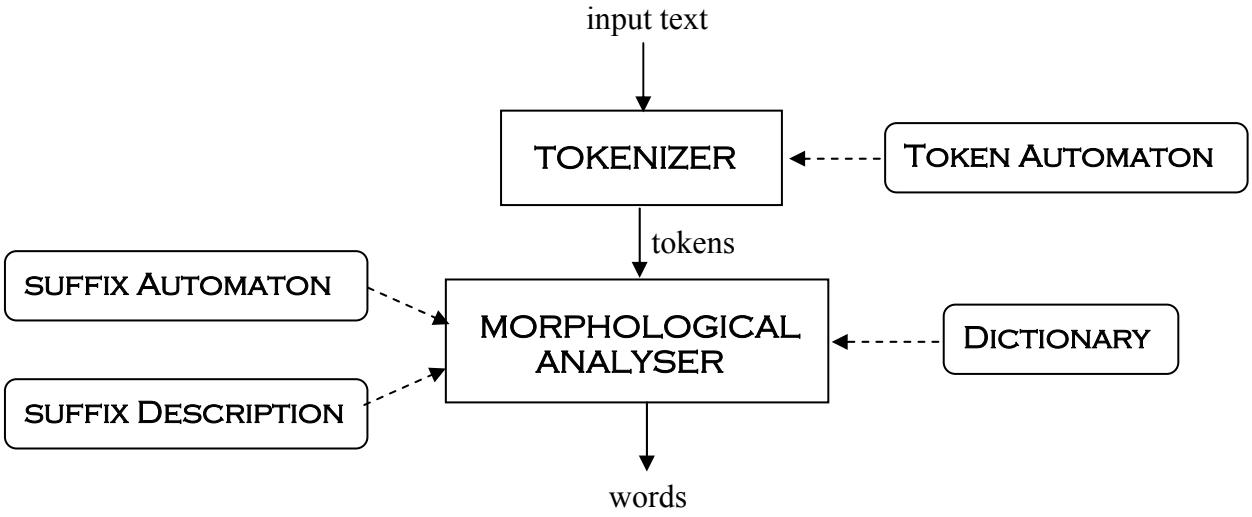


Figure 1 – From the input string to the word hypotheses

2. THE TOKENIZER

The input of the tokenizer is a character string. It can be given either in a file or via a keyboard. A suitable way to interact with the system is via the GUI downloadable from …. The text in a file can include any sequence of characters (ASCII, see Appendix A), but of course a reasonable output will be obtained just in case it refers to a piece of text[1].

[1] No provision is made for handling tables or figures. The file is a txt file.

2.1. The output of the tokenizer

The expected output of the tokenizer is a sequence of "token hypotheses". Each hypothesis belongs to a "tokenizer category". The defined cetegories are[2]:

- GW (general word): any standard word, excluding proper names
- NOMEP: proper names (NOME Proprio)
- SIGLA: abbreviations, codes, initials, etc.
- NUMBER : numbers (in digits)
- DATE : dates (in a standard forms, as 18/01/2009)
- SEGNOINTER : punctuation marks (SEGNO INTERpunzione)
- SPECIAL: other symbols
- PARAGRAPH-NUMB: chapter and paragraph numbers, in dot notation (as 3.1.3.)
- YEAR: just for forms as '03 (standing for 2003)

Of course, there can be some ambiguity; for instance, in case of items in first position in a sentence, (at least in case of Italian and English), if the word admits a general word interpretation (as the name Mark) it could be either a name or a lemma of some other category. In this case, the tokenizer must emit both hypotheses, and will be a task of the POS tagger to choose the best one[3]. It must be clear that the tokenizer categories have nothing to do with the syntactic categories (see Appendix B), though in some cases there can be a correspondence (as in the case of dates).

The format of the output is:

(seq-interp$_1$ seq-interp$_2$ … seq-interp$_n$)

Where each *seq-interp$_i$* is a sequence of tokens, having the form:

(characters token-category)

"*characters*" is the list of machine codes (UTF-8, currently, see Appendix A) of the characters composing the token and "*token-category*" is among the ones listed above. In Box 1, we report the tokenizer output corresponding to the string "Barbara.". In Italian (but this output is language-independent), it can be a proper name (as Barbara in English) followed by a period, or an adjective (or noun: barbarian) followed by a period, or a SIGLA (e.g. an abbreviation, as "Fig."). Note that the GW interpretation has been de-capitalized, i.e. the upper case initial has been replaced by its lower case counterpart.

((((98 97 114 98 97 114 97) GW)	** *first seq-interp*
((46) SEGNOINTER))	
(((66 97 114 98 97 114 97) NOMEP)	** *second seq-interp*
((46) SEGNOINTER))	
(((66 97 114 98 97 114 97 46) SIGLA)))	** *third seq-interp*

Box 1 – Tokenizer output for the string "Barbara."

2.2. The tokenizer arcs

The tokenizer is implemented as a deterministic finite-state automaton. Each input character causes a transition from state S_{start} to state S_{end} (of course it may be that $S_{start} = S_{end}$). During the transition, it may emit an output. This is obtained by buffering the characters following the

[2] Some names come from Italian words. This reflects the first implementation of TULE, which, about 15 years ago, was devoted to Italian.

[3] This is a bit imprecise: the tokenizer cannot know whether there is or not a GW interpretation, since it has no access to the dictionary: also in case the word is "John", it issues both the GW and the NOMEP hypotheses.

previous output: a new output picks one or more of the characters in the buffer, and associates them with a tokenizer category. The used characters are removed from the buffer, and the analysis proceeds, starting from S_{end}. A single output operation can issue more than one token, but the number of consumed characters must be the same (see the example of "Barbara." in Box 1.).

There are two final states: *s-eos* and *s-eof*. The first one specifies that the tokenizer believes that a sentence has been completed (End Of Sentence), the second one says that there are no more characters to analyse (End Of File).

The type of operation (output) to be made is encoded in the tokenizer arcs. A tokenizer arc may include up to four components, i.e.:

- the arc label, specifying when the arc can be traversed. Since the automaton is deterministic, the labels of the arc exiting a state are mutually exclusive. The ***else*** label has the usual interpretation.
- the next state (i.e. S_{end})
- the "pause" symbol (exclamation mark). This specifies that the automaton must proceed in its analysis starting from a character left in the buffer, instead from the next input.
- the actual output operation

The first two components are mandatory, while the last two are optional. The specification of the output operation may include one or more of the following:

- *(FLUSH rem)*: All characters in the buffer, except the last *rem* (an integer) are removed from the buffer. The removed characters are not used (used mainly for deleting one or more spaces)
- *(tok-categ rem)*: as above, but the removed characters are output, and they are assigned the tokenizer category *tok-categ*.
- *(tok-categ rem = default)*: as above, but the characters taken from the buffer are thrown away, and the sequence of characters "*default*" is output in their place.
- *(tok-categ rem + append),* where *append* is a list of characters: as above but the "append" characters are appended at the end of the ones taken form the buffer, instead of replacing them
- *(tok-categ @ char-code)*: a *tok-categ* hypothesis is emitted. The characters associated with it are the characters of the buffer up to (but not including) the first occurrence of *char-code* (an UTF-8 character code).
- *(tok-categ $ char-code)*: as above, but the *char-code* character is included in the output
- *(tok-categ - char-code)*: as @, but here the output characters are all equal to *char-code*, so the extraction of characters form the buffer stops when the first character different from char-code is found in the buffer

A number of examples of the first two types appear in the figures that follow, with the first item, i.e. FLUSH or *tok-categ*, suitably encoded to keep the size of the figures manageable (the codes are described in Fig.2). For the same reason, examples of the other types do not appear in the figures, but their occurrence is explained in the text.

2.3. The arc labels

The input characters are assumed to belong to the extended ASCII UTF-8 character set named iso-latin-1, which is reported in Appendix A. They are grouped in various subsets (char classes), whose list appears in Table 1.

Of course, the different subclasses are not mutually exclusive. Notice that the tokenizer automaton is common to the various languages currently handled by the analyser. For instance, it includes special paths for compact forms (isn't, you're, …) in English, for initial reversed question mark (¿) in Spanish, for middle dot (·) in Catalan, and for a special transliteration of Hindi, where long vowels are represented as capitalized letters (so that they can occur inside a word).

ARC NAME	DEF	CHARS
neutral	9, 10, 13, 32	Tab, Newline, Ret, Space
minlet	97-122, 223-246, 248-255	a-z, ß, à, á, â, ã, ä, å, æ, ç, è, é, ê, ë, ì, í, î, ï, ð, ñ, ò, ó, ô, õ, ö, ø, ù, ú, û, ü, ý, þ, ÿ
cap-el	76	L (UE Directive Codes)
di	100	d (for I'd)
e	101	e (for you're)
el	108	l (for catalan, after a ·)
di-em-er-es-v	100, 109, 114, 115, 118	m (for I'm), r (for you're), s (genitives and it's), v (for I've), d (for I'd)
ti	116	t (for don't, wouldn't, …)
caplet	65-90, 192-214, 216-222	A-Z, À, Á, Â, Ã, Ä, Å, Æ, Ç, È, É, Ê, Ë, Ì, Í, Î, Ï, Ð, Ñ, Ò, Ó, Ô, Õ, Ö, Ø, Ù, Ú, Û, Ü, Þ
period	46	.
min	60	<
mag	62	>
min-min	171	«
mag-mag	187	»
escape	27	Esc
comma	44	,
minus	45	-
slash	47	/
digit	48-57	0-9
zero	48	0
one	49	1
three	51	3
one-two	49, 50	1, 2
four-to-nine	52-57	4, 5, 6, 7, 8, 9
two-to-nine	50-57	2, 3, 4, 5, 6, 7, 8, 9
one-to-nine	49-57	1, 2, 3, 4, 5, 6, 7, 8, 9
zero-to-two	48, 49, 50	0, 1, 2
accent	96	` (grave)
apostrophe	39	'
at	64	@
ampersand	38	&
closed-par	41	)
nth	176	°
mid-dot	183	· (Catalan)
underscore	95	_
s-termin	33, 46, 58, 59, 63	! . : ; ? (sentence terminator)
w-termin	34, 40, 41, 44, 45, 60, 62, 91, 93, 171, 187	" ( ) , - < > [ ] « » (word terminator)
special	35, 36, 42, 43, 61, 92, 94, 123, 124, 125, 126	# $ * + = \ ^ { | } ~
sig-char	38, 45, 46, 95	& - . _
sig-char-not-period	38, 45, 95,	& - _
s-term-not-period	33, 58, 59, 63	! : ; ?
sent-q-start	191	¿ (Spanish)
inter-not-s-termin	34, 40, 41, 44, 45, 91, 93	" ( ) , - [ ]

Table 1 – Arc labels of the tokenizer automaton

The various groupings of digits are used to cover various expressions for dates, and for various forms of real numbers, but more details about the various items that are covered by the automaton

(and about the role of the different labels) can be found below, where the structure of the automaton is analyzed in depth.

2.4. The Start State (*s-0*)

The state *s-0* is entered at the beginning of the tokenization process, being positioned on the first character of the input string and is often re-entered when an output has been emitted. In some cases, the current state remains *s-0* (i.e. S_{end} = s-0) after the analysis of a new character; this happens for instance when a space is encountered (*neutral*), or when a « symbol is found (*min-min*). In the first case, no output is produced (the space is "flushed"), while in the latter a *SEGNOINTER* token is output (***0**, in the figures[4]). In many cases, however, a state is entered, associated with the analysis of specific types of tokens. A *minlet* activates the hypothesis of a general (standard) word (state *s-gw-0*), a capitalized letter makes the automaton pass in state *s-gwp-0* (general word or proper name), a *period* activates the analysis of possible period sequences (*s-dot-0*), a digit the analysis of numbers (states from *s-n-0-a* to *s-n-0-d*), and similarly for the accent and apostrophes. Most of these items are covered in the following sections.

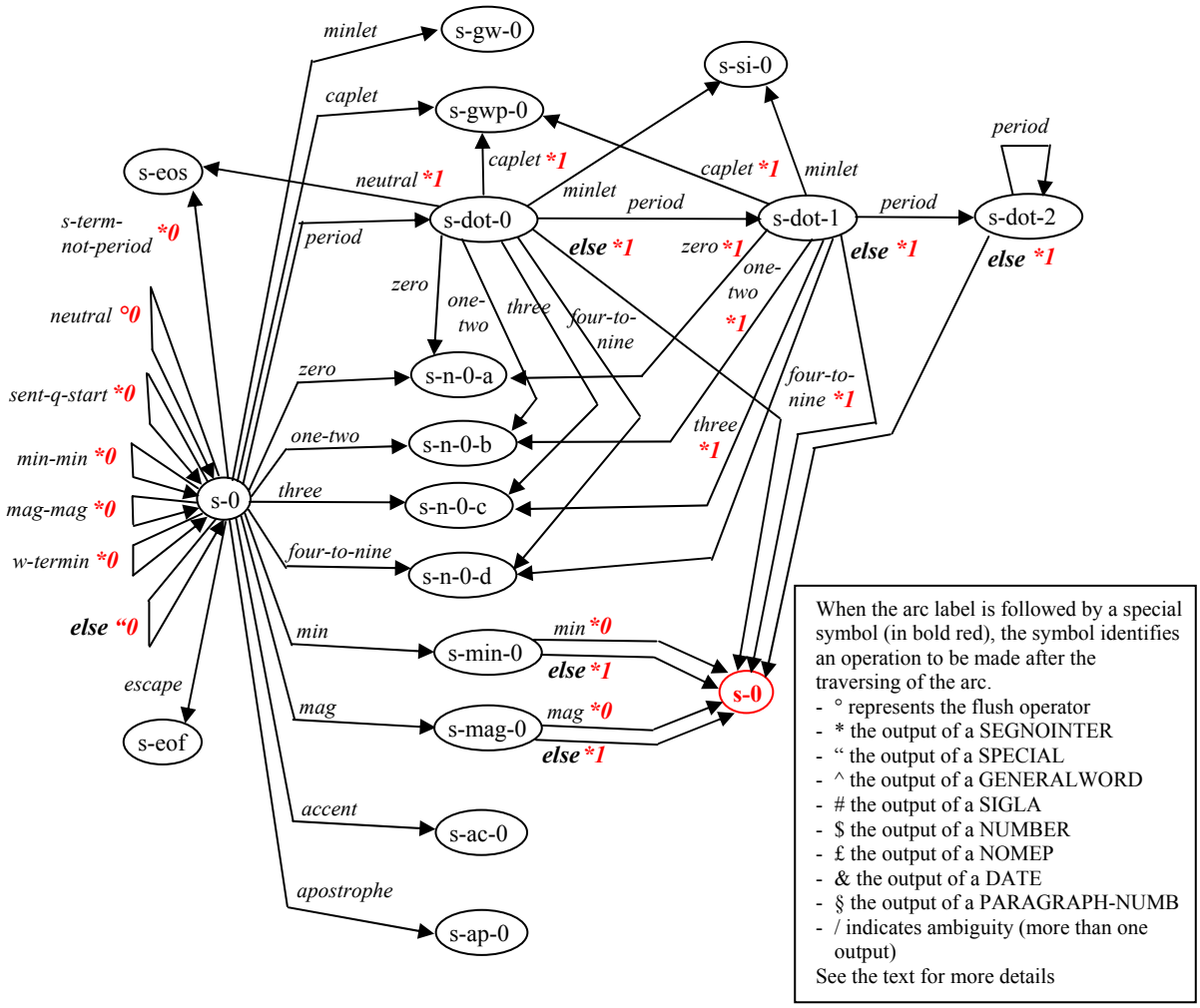


Figure 2 – The tokenizer automaton; first part, with the start symbol (s-0) and including the treatment of dots

[4] Remember that the integer after the operation code specifies how many character must be left in the buffer. So, **0* means that the set of characters in the buffer must be assigned the category *SEGNOINTER*, while **1* means that the characters but the last one are a SEGNOINTERP, but the last one remains in the buffer.

In Fig.2, we specify all possible states reachable from *s-0*, and we also include the analysis of dot sequences and of less-than (*min*) and greater-than (*mag*) symbols. The latter are rather simple: the only provision is that a sequence of two > or two < is interpreted as a single symbol (» or «, respectively).

Slightly more complex is the analysis of dots: a single period can be interpreted as the end of a sentence (*s-eos*) or, if followed by a digit, it may be taken as part of a number (states from *s-n-0-a* to *s-n-0-d*). A *caplet* activates the hypothesis of a new word (assuming that there is a missing space), while a *minlet* activates a *SIGLA* (*s-si-0*). A sequence of two or more dots is interpreted as a single token (suspension or continuation dots).

In case an *escape* character is found, the tokenizer interprets it as an *end-of-file* marker (and enters the state *s-eof*).

2.5. General Words (*s-gw*)

General words are the standard lexical items that can be found in a dictionary, excluding proper names and abbreviations (and also various initials or codes: all of these are encompassed by *SIGLA*, see §2.1). A general word hypothesis is activated by a lower case letter (see Fig.2) and remains active while other lower case letters are found. This is implemented by the loop on the state *s-gw-1*.

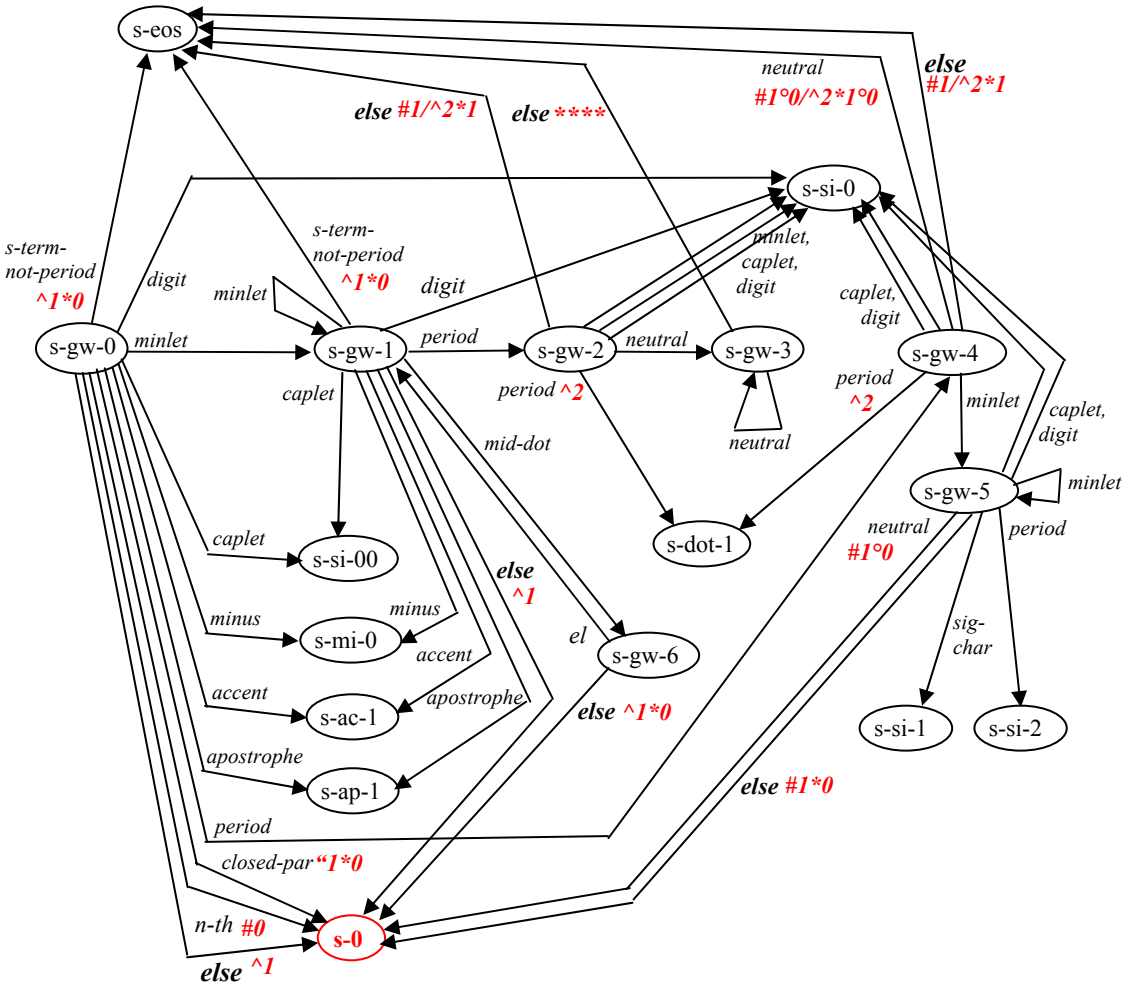


Figure 3 – The tokenizer automaton; second part, general words (*s-gw-i*). For the **** output from *s-gw-3*, see the text

The standard exit (with the recognition of a word) is associated with the ***else*** arcs exiting from states *s-gw-0* and *s-gw-1*. If two or more *minlet*'s are followed by a period, then this may be the initial part of a *SIGLA* or a completion of a sentence (this is accounted for by state *s-gw-2*). For all sentence terminators different from a period (*s-term-not-period*, as colons or question marks) the *s-eos* state is entered, after the emission of two tokens: a general word and a punctuation mark (*SEGNOINTER*).

State *s-gw-3* is entered after a sequence of *minlet*'s, a period and a *space* (or other *neutral*'s, as tabs). It is used just to flush the sequence of spaces. However, when this sequence is completed, the output must be expressed in a rather complex way, saying that either it is a *SIGLA* (including the period) or a general word (excluding the period) followed by a single period. In Fig.3, it has not been encoded in the arc, in order to keep the size of the figure itself manageable. Its actual form is:

((SIGLA $ 46) (FLUSH 1))
((GW @ 46) (SEGNOINTER - 46) (FLUSH 1))))

Note that *(FLUSH 1)* means that the last character (the one that caused the ***else*** transition) must remain in the buffer. In this case, the "!" (exclamation mark) is present on the arc to indicate that this remaining character, and not the next one in input, must be used to proceed in the analysis.

The case of a single *minlet* followed by a period is covered by state *s-gw-4*: if another period follows, a general word is output, and the hypothesis of a dot sequence is entered; otherwise a *sigla* is hypothesized, but its analysis is entered at different stages, according to the sequence found until now (states *s-si-0*, *s-si-1*, and *s-si-3* in the figure), possibly passing through *s-gw-5*.

State *s-gw-6* is used just to account for the Catalan sequence "l·l" (as in *col·lectió – collection*).

Note that in case of hyphens (*minus*), accents and apostrophes, no output is emitted, since this is delayed to a next step (to account for single hyphenated words, for stressed words and for elision).

2.6. General Words or Proper Names (*s-gwp*)

The part of the automaton devoted to the treatment of string segments ambiguous between general words and proper names is not substantially different from the one of general words. State *s-gwp-0* is entered after an upper case letter, and a sequence of following lower case letters is accepted (loop on *s-gwp-1*) as well as a sequence of upper case letters (loop on *s-gwp-2*).

The outputs are more complex than in GW, since most of them must account for the ambiguity between general words and proper names, and in some cases a SIGLA hypothesis must also be emitted.

Also in this case, a "complex" output appears in the figure; the "real" output operation is reported below, without further comments, since the case is very similar to the one of *s-gw-3* above.

((*SIGLA* $ 46) (*FLUSH* 1))
((*SIGLA* @ 46) (*SEGNOINTER* - 46) (*FLUSH* 1))
((NOMEP @ 46) (*SEGNOINTER* - 46) (*FLUSH* 1))
((GW @ 46) (*SEGNOINTER* - 46) (*FLUSH* 1))))

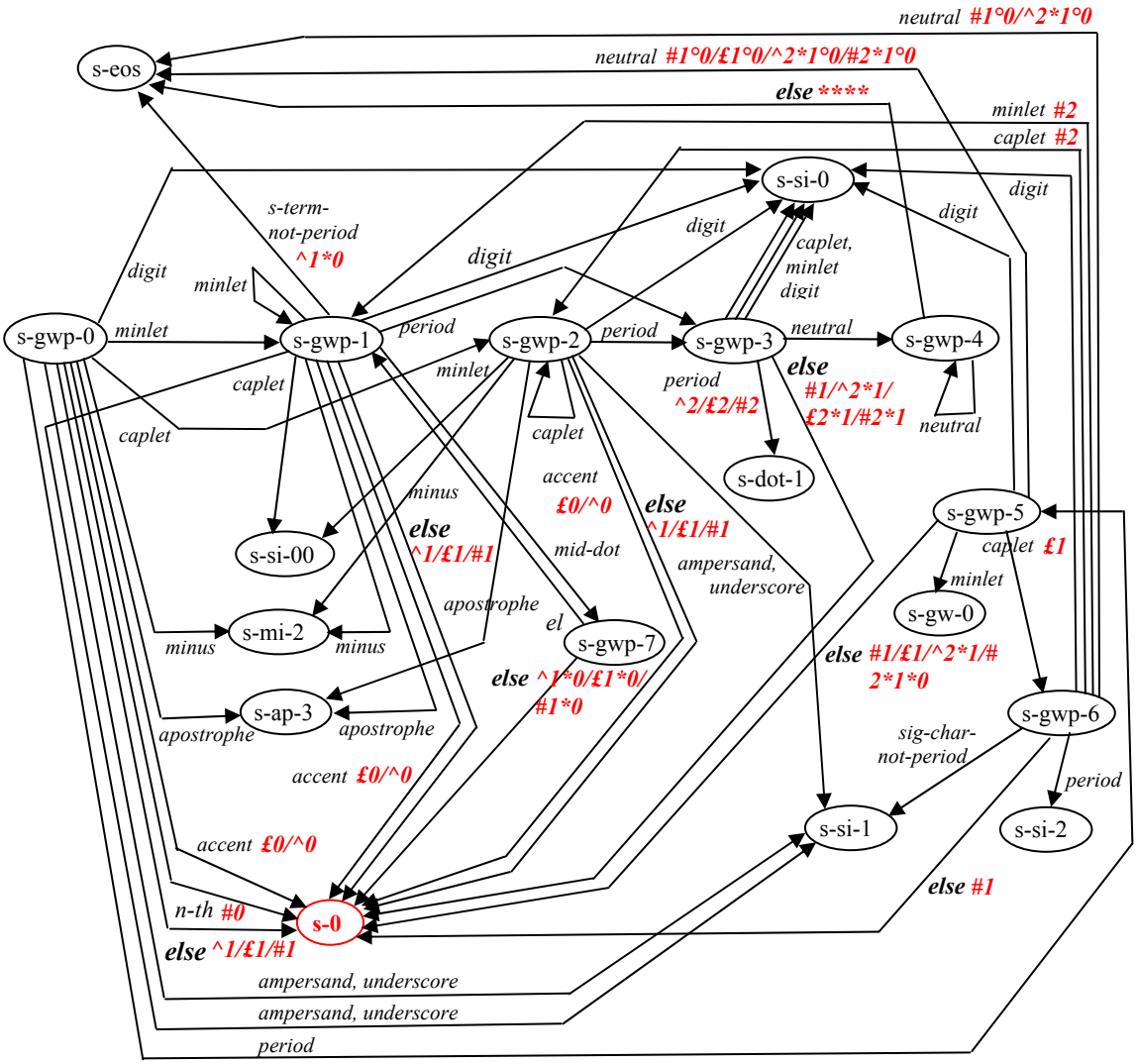


Figure 4 – The tokenizer automaton; third part, general words or proper names (*s-gwp-i*). For the **** output from *s-gwp-4*, see the text

2.7. Numbers (s-n)

There are four entry points for the portion of the automaton referring to numbers:

- *s-n-0-a*: entered after an initial zero
- *s-n-0-b*: entered after the digit one or the digit two
- *s-n-0-c*: entered after the digit three
- *s-n-0-d*: entered after any other digit

This is due to the need of covering dates in the form "8/10/2008". Any slash after a single digit different from zero, or after a pair of digits in the range 01-31 causes a transition to state *s-d-0*, which is a state in the "dates" part of the automaton (§2.9).

If, on the contrary, a period or a comma are found after one, two or three digits, then a number with thousands separation and/or with decimals is hypothesized. In particular, state *s-n-6* refers to a sequence of one to three digits followed by a period, and *s-n-7* refers to a such a sequence followed by a comma.

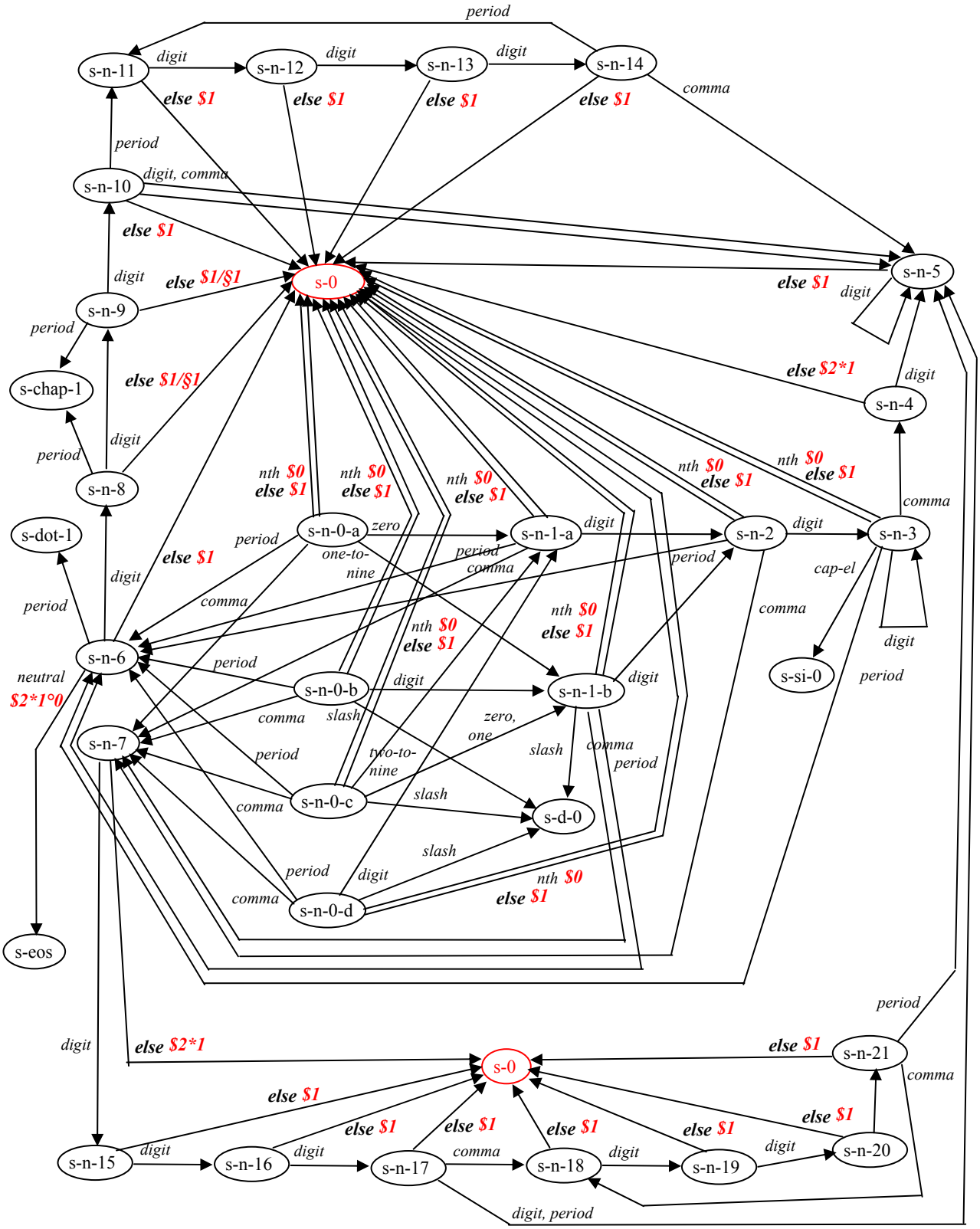


Figure 5 – The tokenizer automaton; fourth part: numbers

The states from *s-n-8* to *s-n-14* refer to numbers with periods for thousands separation (i.e. in the form 3.127, 125.712.241). Note that in this case, all digit sequences (except for the first one) must include exactly three digits. If the period is followed by more than three digits, it is assumed that the period acts as a separator for decimals (3.1415). The same happens if a comma is found in place of any of the following periods (2.715,37 or 31.432.785,4433). In this case, state *s-n-5* is entered.

2.8. Accents and apostrophes (s-ac, s-ap)

By "accent", we mean the grave accent, by "apostrophe" the small vertical stroke; no provision is made for separate handling of the acute accent. This is due to the initial flavour towards Italian. In Italian, both acute and grave accent are normally used, but very few people are able to keep them apart. What happens is that either the stressed char is used (e.g. "*è*"), or the accent is written separately, but in this case the apostrophe is most often used in place of the accent, while it is less common to use the grave accent. I do not remember of any case of an acute accent standing alone.

For apostrophes, the situation is made more complex by the fact that the same character is used as a real apostrophe (to signal some form of elision) or as a quotation mark (open and closed).

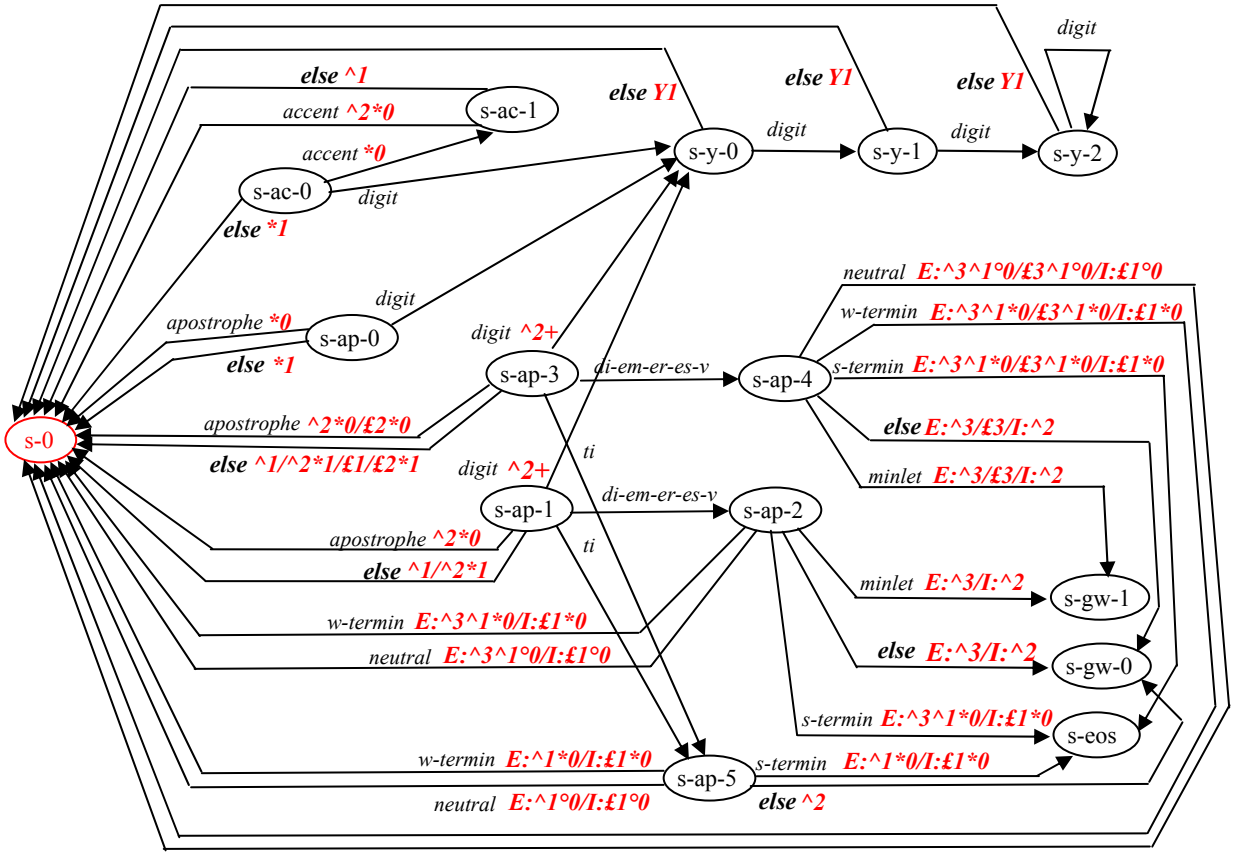


Figure 6 – The tokenizer automaton; fifth part: accents and apostrophes

2.9. Dates (s-d)

Dates are much simpler than the previous items. The only form that is currently recognized is DD/MM/YYYY. State *s-d-0* is entered from some states of the number part of the automaton (*s-n-0-b, s-n-1-b, s-n-0-c and s-n-0-d*, see fig.5) when a slash is encountered. This corresponds to the completion of the day part of the date: from *s-n-0-b* after a 1 or 2, from *s-n-1-b* after a 1x , 2x, 30 or 31, from *s-n-0-c* after a 3, from *s-n-0-d* after a digit other than 1, 2 or 3. State *s-d-3* refers to the month part: 01 to 09 (passing through *s-d-1*), 10, 11 or 12 (passing through *s-d-2*). In case a 1 to 9

without zero is found, state *s-d-3* is skipped, passing directly to *s-d-4* (after the slash). Note that if this sequence is not respected, a non-date output is produced. For instance, from *s-d-3* (e.g. after "3/10") any character different from a slash produces an output *NUMBER+ SPECIAL+NUMBER*.

The correct *DATE* output is obtained from states *s-d-6*, *s-d-7* and *s-d-8*, though the year part is accepted if it includes any number of digits greater or equal to 2.

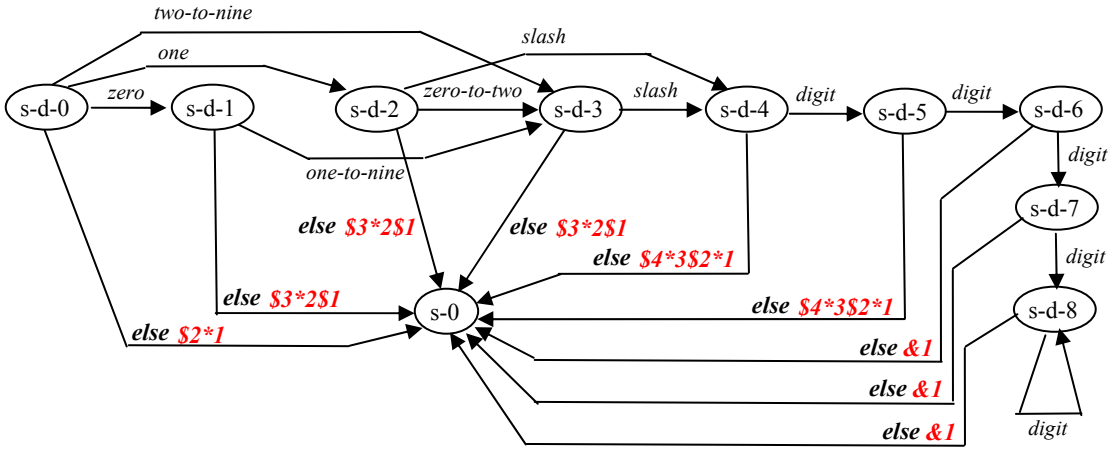


Figure 7 – The tokenizer automaton; sixth part: dates

2.10. The minus-hyphen sign (s-mi)

The need for a special treatment of hyphens is due to the presence in the dictionary of hyphenated words (e.g. week-end). In case such a sequence occurs, two different hypotheses must be produced, the first one referring to two distinct words connected by an hyphen, the second one concerning the single hyphenated word.

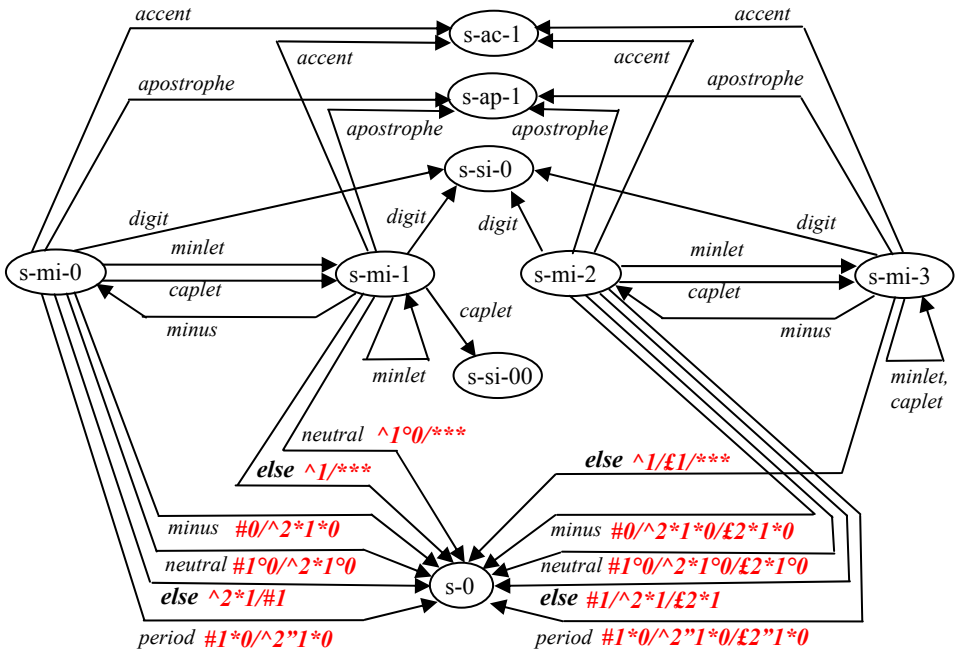


Figure 8 – The tokenizer automaton; seventh part: minus-hyphen

This is reflected in the various outputs. For instance, *s-mi-0* is entered after a sequence of lower case letters, and a hyphen; a new letter leads to state *s-mi-1*, and from this a *neutral* (identifying the end of the word, produces an output shown in Fig.8 as *^1°0/****. We have now scanned a sequence as *xxxx-yyyyN* (where N stands for neutral); the first output (before the slash) specifies that the characters before the last one (i.e. *xxxx-yyyy*) must be output as a GW and that the last character (*N*) must be flushed. The complex expression after the slash (second hypothesis) is:

((GW @ 45) (SEGNOINTER - 45) (GW @ 45) (SEGNOINTER - 45) (GW 1) (FLUSH 0)))

This means:

- Output as GW all characters up to (but not including) 45 (i.e. "-"). This produces (xxxx GW).
- Output as SEGNOINTER all characters equal to 45. This produces (- SEGNOINTER)
- Output as GW all characters up to (but not including) 45 (i.e. "-"). This produces (yyyy GW).
- Output as SEGNOINTER all characters equal to 45. This produces (- SEGNOINTER)
- Flush until the end of the buffer (removing N).

The third and fourth operation have no effect if no 45 character appears in the buffer. This enables the automaton to handle up to a maximum of three words separated by hyphens.

Other situations are treated analogously; in particular, state *s-mi-2* is associated with cases where the first letter is capitalized (e.g. *Xxxx-yyyy*).

2.11. SIGLA's (s-si)

The tokenizer category SIGLA encompasses any sequence of letters, digits, and other characters belonging to the class *sig-char* (ampersand, period, hyphen and underscore). Of course they must be mixed in such a way to exclude other interpretations (e.g. not just small letters, otherwise the hypothesis is GW, not just digits, then period, then digits, otherwise the hypothesis is NUMBER, and so on). The portion of the automaton in Fig.9 is rather simple and does not require special comments, except a remind that state *s-si-00* was introduced to account for the mixing of lower and upper case letters appearing in our Hindi transliteration, that *s-si-0* is associated with any partially recognized SIGLA sequence, *s-si-1* is entered after ampersand, hyphen or underscore, and *s-si-2* is reached after any sequence of letters including two or more periods.

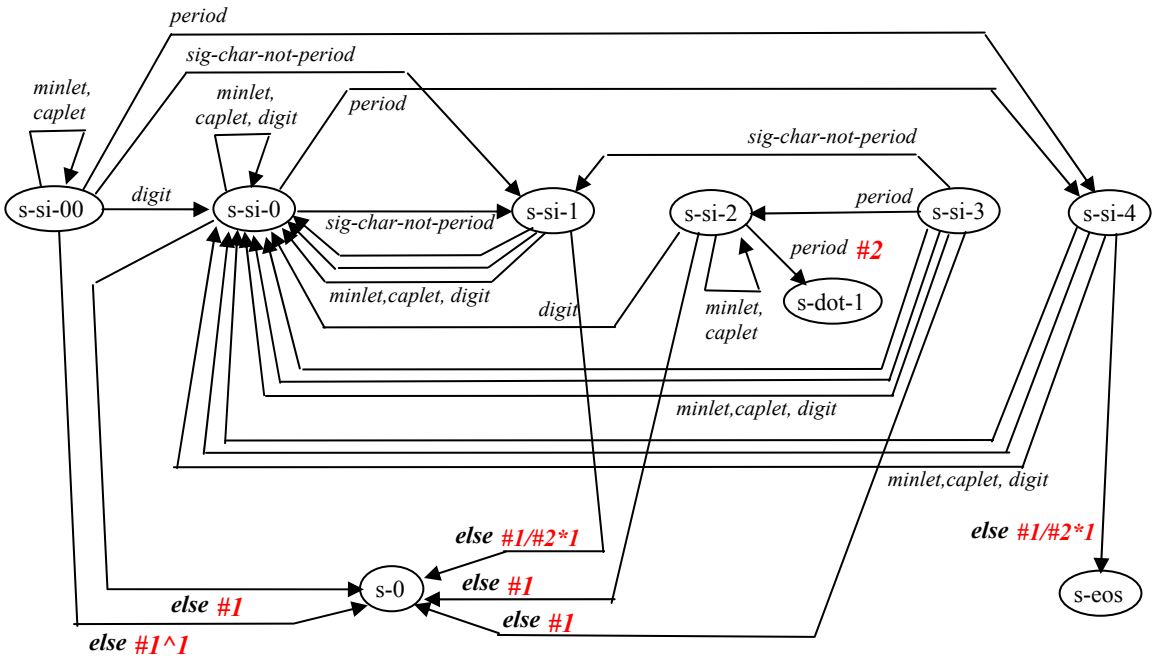


Figure 9 – The tokenizer automaton; eighth part: SIGLA's

2.12. Chapter and Paragraph Numbers (s-chap)

This section refers to dotted sequences of numbers, usually used to index book chapters or paragraphs. They have been taken apart both because their internal structure is different from the one of numbers and because their syntactic role in a text is particular.

State *s-chap-1* is reached from *s-n-8* and *s-n-9* (where a number with dots was being analysed) in case less than three digits occur between two dots. This is based on the assumption that the number of section is less than one hundred. After this, any sequence of digit and periods is accepted. Note that a separator after the second period makes the automaton issue also the hypothesis of a dotted number followed by a period (*$2*1*).

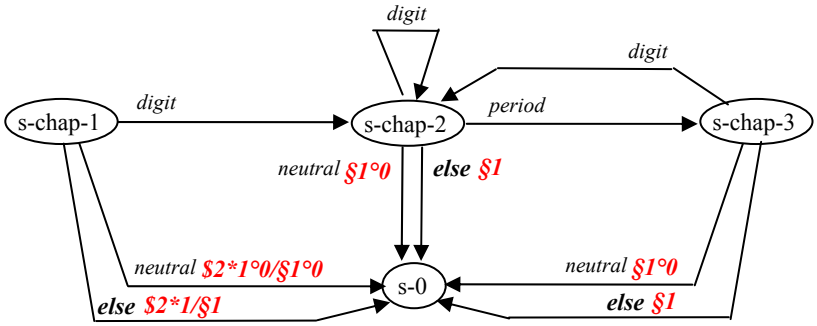


Figure 10 – The tokenizer automaton; ninth part: Chapter and Paragraph Numbers

3. FROM TOKENIZER OUTPUT TO WORD HYPOTHESES

This part of the process has the goal of passing from token hypotheses to word hypotheses. Here, the term "word" is used in a rather loose sense, including actual lexical items (lemmata), proper names, dates, numbers, etc., according to the syntactic categories (POS) foreseen by the parser (see Appendix B).

The knowledge sources used in this process are (see Fig.1) a dictionary (actually, more than one), a suffix automaton and some information about suffixes. In the current implementation, prefixes are not accounted for, so that the more general term "affix" is constantly replaced by "suffix".

Since this knowledge, differently from the one about tokenization, is language-dependent, I will make a general presentation of the approach, but the tables and examples have to be intended as related to the particular language.

Contrarily to what we have seen for the tokenizer automaton, the suffix automata are generated automatically (at least in part, as we will see) from a declarative description of the possible suffixes. The first paragraph of this section is concerned with such "automatic compilation" process.

3.1. Generation of the Suffix Data Structures

The various suffix tables associated with the different languages include some codes whose goal is to keep the tables smaller and, according to my view, more readable. A first piece of knowledge required for the pre-compilation phase is the description of these codes and of the format of the input table. The architecture of this compilation module is presented in Fig.11.

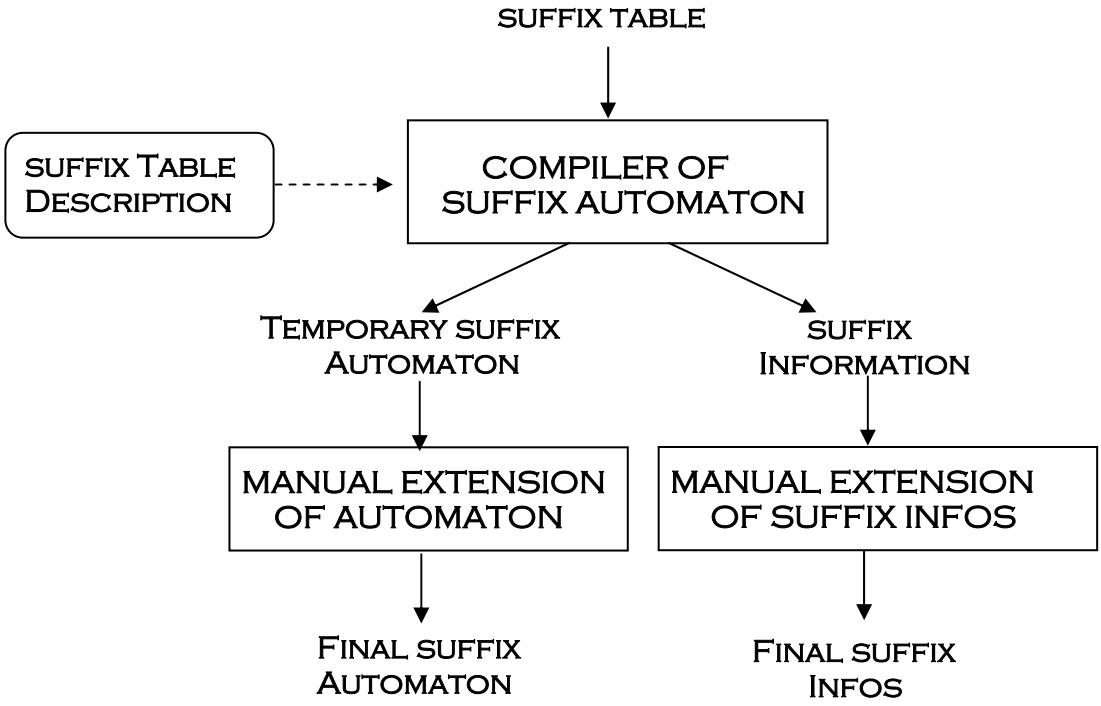


Figure 11 – Production of the KS used by the Morphological Analyser

I will use, as a first example of *Suffix Table Description* the one for Italian. It is a bit more complex than the one for English, that will be described immediately after, so that it enables me to emphasize some of the features of the description.

Let's start with adjectives, which are reasonably simple. In Italian, adjectives can be inflected for gender and number and not for case, so that *bello* (nice) can have the following forms[5]:

- *bello* (masculine, singular)
- *bella* (feminine singular)
- *belli* (masculine plural)
- *belle* (feminine plural)

The dictionary is based on root-suffix separation, so that in the dictionary there is the *root* of the word, and not the so-called *citation form*. The entry for *bello* is the following (see Appendix C for a brief overview of the dictionaries used by the morphological analyser):

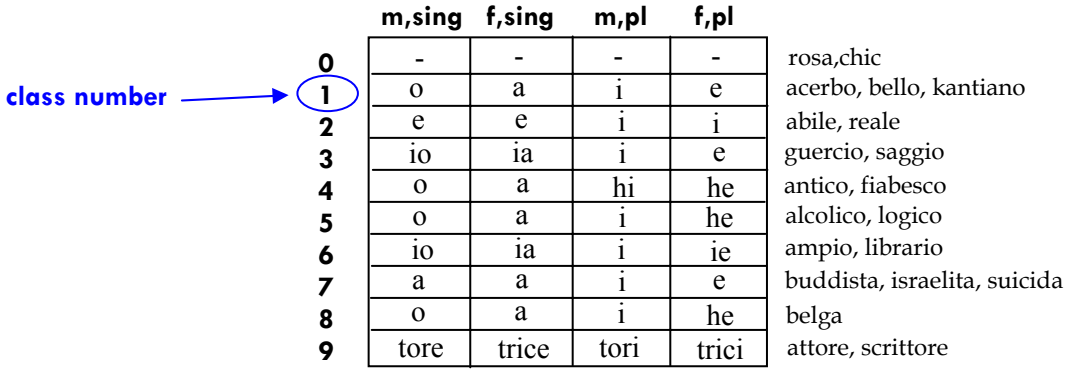
(bell ((bello cat adj classe (1))))

where the access is via *bell*, and the associated infos are *bello* (lemma), "*cat adj*" (syntactic category), "*classe (1)*". The latter specifies that the accepted suffixes are *o, a, i, e* (according to the variations listed above), so what is needed is a specification that:

1. The first adjectival class (*classe (1)*) refers to the list of suffixes *(o a i e)*
2. The list refers, in order, to *((m sing) (f sing) (m pl) (f pl))*, where *m* stands for masculine, *f* for feminine, *sing* for singular and *pl* for plural

The part of (input) suffix table referring to adjectives is reported in Fig.12. Since classes are, by default, indexed via increasing integers, what is missing is a specification of the meaning of the different columns.

[5] There are other forms, as "begli" (masculine plural) before vowels or other beginnings ("begli occhi", "begli sforzi", …) or "bei" ("bei cani"). These are accounted for as invariant (supplementary) forms. For instance, the entry for "begli" is: *(begli ((bello cat adj classe (0) gender m number pl)))*



	m,sing	f,sing	m,pl	f,pl	
0	-	-	-	-	rosa,chic
1	o	a	i	e	acerbo, bello, kantiano
2	e	e	i	i	abile, reale
3	io	ia	i	e	guercio, saggio
4	o	a	hi	he	antico, fiabesco
5	o	a	i	he	alcolico, logico
6	io	ia	i	ie	ampio, librario
7	a	a	i	e	buddista, israelita, suicida
8	o	a	i	he	belga
9	tore	trice	tori	trici	attore, scrittore

Figure 12 – Structure of the input Suffix Table (Italian Adjectives)

The description of the required pieces of information is given in the form exemplified in Table 2

cases	nil
genders	m f
numbers	sing pl
gender-numbers	(m sing) (f sing) (m pl) (f pl)

Table 2 – Description of the suffix data for Italian adjectives

The first three lines define the values for the three relevant parameters: *case* is not encoded in Italian morphology (*nil*); the genders can be masculine (*m*) or feminine (*f),* but, for instance, not *neuter*; the number can be singular (*sing*) or plural (*pl*). The fourth line specifies the values associated to the four columns in the suffix table (Fig.12). The Italian examples can be compared to the analogous tables for English (Fig.13 and Table 3).

In English, class 0 is reserved for comparatives and superlatives (as *clumsier* and *clumsiest*)[6]. Class 1 is for invariables, where the root coincides with the lemma (ex. *coarse*): the value *@empt* refers to the null suffix. The other classes refer to adjectives with specific suffixes. Note that class 7 is described in two lines: this accounts for adjectives that accept two types of suffixes, as *numer-*, which can produce both *numeric* and *numerical*. In principle, most classes could be dispensed with, by simply specifying that adjectives are invariable. However, the classes are also used for the derivation of adverbs from adjectives, so that from class 3 it is possible to derive the adverb *admir-ably* and from class 5 *vol-ubly*.

Finally, note that the table has six columns: this is due to the existence of the *neuter* gender for English. Of course, this has to be told to the analyser somewhere. And in fact the description table for English (Table 3) specifies that the possible values for the *gender* parameter are "*m, f, n*".

Both in Italian and in English, no cases are provided. In many languages, however, *case* is a relevant feature. So, the tables shown in Figg. 12 and 13 should actually be seen as just one slice of a three-dimensional table (which, for Italian and English, includes a single slice).

It must be observed that the separate specification of the range of values for the different parameters is useful in the production of the output tables, which act as a kind of *reversed index* on suffixes. What is specified there are all places in the table where a given suffix occurs. For Italian adjectives, for instance, the output table says that "e" occurs in class 1 for feminine plural, in class 2 for masculine singular and for feminine singular, in class 3 for feminine plural and in class 7 for

[6] This is a provisional solution. Class 0 records the presence of comparatives and superlatives, but it should be replaced by a derivational process as the ones described in §3.2.1.1.

feminine plural. However, in the resulting table, the specification for class 2 is given as "*allval sing*", where *allval* is a term that unifies with all values. This compaction is possible only because masculine and feminine are all possible values for the *gender* attribute.

	m,sing	f,sing	n,sing	m,pl	f,pl	n,pl	
0	-	-	-	-	-	-	clumsier, clumsiest
1	@empt	@empt	@empt	@empt	@empt	@empt	coarse, bayesian
2	y	y	y	y	y	y	empty, evolutionary
3	able	able	able	able	able	able	admirable, movable
4	ible	ible	ible	ible	ible	ible	eligible, extensible
5	uble	uble	uble	uble	uble	uble	soluble, voluble
6	ic	ic	ic	ic	ic	ic	terrific
7	ic	ic	ic	ic	ic	ic	numeric, syntactic
7	ical	ical	ical	ical	ical	ical	numerical, syntactical

Figure 13 – Structure of the input Suffix Table (English Adjectives)

cases	nil
genders	m f n
numbers	sing pl
gender-numbers	(m sing) (f sing) (n sing) (m pl) (f pl) (n pl)

Table 3 – Description of the suffix data for English adjectives

The input tables include data about the various inflected categories of the language. For Italian and English, they provide suffixes for adjectives, nouns, verbs and some data about adverbs. Pronouns are missing because they have a rather irregular morphology and the regular part is obtained from the adjective tables. Adverbs are not inflected, but the tables also encode some *derivation* information, as the fact that an adverb can be derived from an adjective or a noun from a verb (see §3.2.1.1). The tables for verbs are considerably more complex than the ones for adjectives and nouns, since they must encode also the person and tense information.

We report in Box 2, the actual format of the table for three of the nine defined classes for English verbs. With respect to irregular verbs (class 2), it can be observed that all entries are defined, also the ones associated with the irregular forms. They are used in the dictionary entries in the following way (example of "to eat"):

(ate ((eat cat verb classe (2 (b f)))))
(eat ((eat cat verb classe (2 (a e h l m n)))))
(eaten ((eat cat verb classe (2 (i)))))

where the class code is expanded with specific mood-tense codes, i.e. the ones associated with the different lines of the table (see the comments on the lines of Box 2). Of course, also the description of the verbal part of the table is more complex than the one for adjectives. It is reported in Table 4. Note that the number of parentheses varies; this depends on the fact that the participles are inflected for gender and number, so that they are assumed to behave as adjectives and nouns, i.e. there could also be a *case* marking, not present for English. On the contrary, for the other tenses, the possible case marking is encoded in a different way. More details can be found in Appendix D, in connection with the Hindi verbal suffix tables.

The compiling process produces two outputs, an inverse index on the suffixes, specifying with which combination of category and parameters a given suffix is associated (see Box 3), and an automaton that enables to scan a word from the end, producing all hypotheses of root+suffix combinations (see Fig.14). The work of the morphological analyser is to scan the word and, for each recognized suffix, to retrieve from the inverse index the data about parameter values related with each suffix. This is described in more depth in the next section.

```
; *** regular (walk, ...)
   1  ((@empt    @empt   s       @empt  @empt  @empt)     ; indicative present (a)
       (ed       ed      ed      ed     ed     ed)       ; indicative past (b)
       (@empt    @empt   @empt   @empt  @empt  @empt)     ; subjunctive present (e)
       (ed       ed      ed      ed     ed     ed)       ; subjunctive past (f)
       ((ing     ing     ing     ing    ing    ing))     ; participle present (h)
       ((ed      ed      ed      ed     ed     ed))      ; participle past (i)
       (ing)                                             ; gerund (l)
       (@empt)                                           ; infinite (m)
       (nil      @empt   @empt   @empt  @empt  @empt))    ; imperative (n)
; *** irregular (buy, eat, ...)
   2  ((@empt    @empt   s       @empt  @empt  @empt)     ; indicative present (a)
       (@empt    @empt   @empt   @empt  @empt  @empt)     ; indicative past (b)
       (@empt    @empt   @empt   @empt  @empt  @empt)     ; subjunctive present (e)
       (@empt    @empt   @empt   @empt  @empt  @empt)     ; subjunctive past (f)
       ((@empt   @empt   @empt   @empt  @empt  @empt))    ; participle present (h)
       ((@empt   @empt   @empt   @empt  @empt  @empt))    ; participle past (i)
       (@empt)                                           ; gerund (l)
       (@empt)                                           ; infinite (m)
       (nil      @empt   @empt   @empt  @empt  @empt))    ; imperative (n)
; *** have
   3  ((ve      ve      s       ve     ve     ve)       ; indicative present (a)
       (d        d       d       d      d      d)        ; indicative past (b)
       (ve       ve      ve      ve     ve     ve)       ; subjunctive present (e)
       (d        d       d       d      d      d)        ; subjunctive past (f)
       ((ving    ving    ving    ving   ving   ving))    ; participle present (h)
       ((d       d       d       d      d      d))       ; participle past (i)
       (ving)                                            ; gerund (l)
       (ve)                                              ; infinite (m)
       (nil      ve      ve      ve     ve     ve)       ; imperative (n)
```

Box 2 – Actual format of the suffix table for three verbal classes

Cases	nil
Genders	m f n
numbers	sing pl
Persons	1 2 3
Moods	ind subjunctive participle gerund infinite imperative
Tenses	pres past
moods-tenses-def	a (ind pres) (person number) (1 sing) (2 sing) (3 sing) (1 pl) (2 pl) (3 pl)
	b (ind past) (person number) (1 sing) (2 sing) (3 sing) (1 pl) (2 pl) (3 pl)
	e (subjunctive pres) (person number) (1 sing) (2 sing) (3 sing) (1 pl) (2 pl) (3 pl)
	f (subjunctive past) (person number) (1 sing) (2 sing) (3 sing) (1 pl) (2 pl) (3 pl)
	h (participle pres) (gender number) (m sing) (f sing) (n sing) (m pl) (f pl) (n pl)
	i (participle past) (gender number) (m sing) (f sing) (n sing) (m pl) (f pl) (n pl)
	l (gerund pres) nil nil
	m (infinite pres) nil nil
	n (imperative pres) (person number) (1 sing) (2 sing) (3 sing) (1 pl) (2 pl) (3 pl)

Table 4 – Description of the suffix data for English verbs

```
S   (NOUN ( 1 (GENDER F NUMBER PL)
               2 (GENDER M NUMBER PL)
               3 (GENDER N NUMBER PL) )
       VERB   ( 1 (A (3) )
               2 (A (3) )
               3 (A (3) ) ) )
ES  (NOUN (4 (GENDER F NUMBER PL)
               5 (GENDER M NUMBER PL)
               6 (GENDER N NUMBER PL) )
       VERB   ( 7 (A (3) )
               8 (A (3) ) ) )
```

Box 3 – A portion of the inverted index generated from the suffix tables

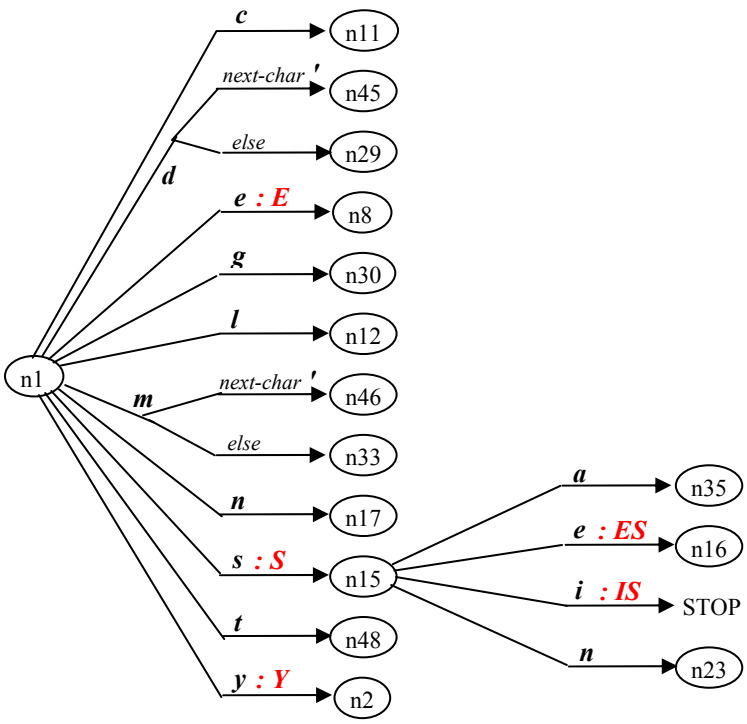


Figure 14 – A portion of the suffix automaton generated from the suffix tables

By inspecting Fig.14, it is possible to see an example of the manual intervention that modifies the automaton produced automatically. The "*d*" and the "*m*" arcs from *n1* lead to choice points which have been added manually. They involve a lookahead (or lookback, since we are moving backward from the end of the word) on the input string. The "*d*" case refers to expression as "*I'd*" (I would) and the "*m*" case to "*I'm*" (I am). These are handled in states *n45* and *n46*, which have been created manually, while *n29* and *n33* are the states generated automatically from the suffix tables.

3.2. Morphological analysis

The term "morphological analysis" is used here to encompass a wide coverage process, i.e. the one that takes the output of the tokenizer automaton and emits hypotheses about items (usually words) belonging to one or more of the syntactic categories listed in Appendix B. In most cases (i.e. for dictionary entries) this involves a true morphological analysis, but in other cases, it

requires just dictionary look-up (as for proper names and abbreviations) or no effort at all (as for punctuation marks). The activity to perform is determined by the *tokenizer category* (see 2.1).

In the next sections, I present the operations performed in correspondence with the various tokenizer categories, omitting *SEGNOINTERP* (punctuation) and *SPECIAL* (special symbols), for which nothing is done.

With respect to various possible ambiguities, it is possible that some choice has to be made at this stage. This problem is addressed in Section 3.3.

3.2.1. GW (General Word)

The first step in the analysis of GW is the extraction of the possible suffixes. In order to make the process clear, I present an example. Let's suppose that the result (or one of the results, in case of ambiguity) of the tokenizer is the following:

(((97 99 116 114 101 115 115 101 115) gw))

Where the sequence of numbers is the one of the character codes of *actresses*. On the basis of the automaton in Fig.14, the arc labelled with *s: S* (code 115 is *s*) is traversed. This corresponds to the emission of the *actresse+s* hypothesis and to the access to node *n15*. From this node, the *e: ES* (code 101 is *e*) arc is activated, and the *actress+es* hypothesis is formed. The next state is *n16*, but from there no new hypothesis can be made (it is used for *–ies* endings).

After that, the inverse index (Box 2) is accessed, and the null suffix hypothesis is also attempted[7]. What is obtained after this step is:

actresses+nil /
actresse+s *(noun (1 (gender f number pl) 2 (gender m number pl) 3 (gender n number pl)) verb (1 (a (3)) 2 (a (3)) 3 (a (3)))*
actress+es *(noun (4 (gender f number pl) 5 (gender m number pl) 6 (gender n number pl)) verb (7 (a (3)) 8 (a (3)))*

Now, the dictionary is accessed to look for a match. Neither the root *actresses*, nor the root *actresse* appear in the dictionary. On the contrary, for *actress*, we get:

(actress ((actress cat noun classe (4))))

The match is found, since in both the result of the extraction of suffixes and in the dictionary there is a *noun* item of morphological class 4. Consequently, the final result is:

actresses ((actress cat noun classe (4) gender f number pl))

or, in the compact form occurring in the parser output and in the Treebank:

actresses (actress noun f pl)

3.2.1.1. Morphological derivations

Some general words are not present explicitly in the dictionaries, but are obtained as derivations from other entries. This applies to adverbs (*frequent* → *frequently*) and nouns (*negotiate* → *negotiation*). In these cases, the match is implemented procedurally, by checking the correspondence between the morphological class proposed by the automaton and the one stored in the dictionary, which, in these cases, do not coincide (see below).

[7] This is different from the *@empt* null suffix encoded in tables. *@empt* is used for the null suffix of morphologically variant items. The default hypothesis mentioned in the text refers to *invariables*, i.e. items that feature no morphological variations. They are identified in various ways:

a) The value of the *classe* feature in the dictionary is *0* (but, in English, this is reserved for comparatives and superlatives)
b) The syntactic category is in the list of *invariant categories* (adv, conj, num, interj, phras, prep)
c) They appear in a separate dictionary of *invariable* words

The first case concerns the derivation of nouns from verbs. The verb in question must be marked in the dictionary as enabling the derivation. This is made via the feature *nom*, that can appear in the verbal entries. The possible values are:

Italian: **o** (*abbracciare → abbraccio [embrace → embrace]*), **zione** (*abbreviare → abbreviazione [abbreviate → abbreviation]*), **mento** (*addestrare → addestramento [train → training]*), **zion-ment** (*ammonire → ammonizione, ammonimento [admonish → admonition, admonishment]*)

English: **ion** (*aggregate → aggregation*)

Beyond the possibility of the derivation, the actual form the suffix may assume depends on the verbal class. For instance the verb *abbracc-iare* (morphological class 5) produces *abbracc-io*, while the verb *aiut-are* (*[help]* morphological class 1) produces *aiut-o*. As said above, the correspondence is checked procedurally (function *ana_noun*) and takes advantage from the verbal class obtained from the dictionary and the suffix class for nouns obtained from the suffix tables (see classes 30-38 in the full definition of the Italian nominal classes appearing in Appendix D). For instance, the verbal class 5 (*abbracc-*, from the dictionary) can only match with the noun class 37 (*-io*, from the suffix table). The process is the same for English, with the exception that verbs of morphological class 5 (*idealize, legalize*) are not explicitly marked in the lexicon, since the possibility of performing the derivation (*idealization, legalization*) is assumed by default.

The second case refers to the derivation of nouns from adjectives. This is currently implemented only for Italian, but this process could in principle be applied also to English. In this case, some adjectives in the dictionary include the feature "*nom ita*" that licenses derivations of the form *ambiguo → ambiguità* (*ambiguous → ambiguity*). The process is the same as for verbs (nominal suffix classes 40 and 41 in Appendix D).

In Italian, the process of morphological derivation is applied also for obtaining the *superlative* forms of adjectives (*bello → bellissimo [nice → very nice]*). The process is the same as the one outlined above for other derivations. No extra information is present in the dictionary, since the assumption is that this derivation can be applied to all adjectives. However, also in this case, two morphological codes are needed to keep apart the *–issimo* suffix (*bell-o → bell-issimo*) from *–hissimo* (*antic-o → antic-hissimo*). As observed in Footnote 6, this should also be applied to English, but the current implementation does not cover this case.

Finally, derivations are used to obtain adverbs from adjectives, in the same way as the other derivational process described above.

3.2.1.2. Numbers expressed in letters

The analyser of general words takes care of another possibility, i.e. that the input string expresses a number written in letters instead of in digits, as *three, thirtytwo*, *twohundredninetynine*, *twentyeighth,* etc. The recognition of these expressions, and the production of the actual numeric value associated with them is accomplished by a separate module which, again, takes advantage of an automaton that encodes the various forms these descriptions may assume.

3.2.2. YEAR

The tokenizer category YEAR refers to strings as '07, '954, '91. This is just completed to compute the actual numerical value for the year in question. Since at this stage there cannot be any contextual analysis, some defaults are used.

- 'xxxx: this is a partially incorrect expression; the apostrophe is ignored, and the resulting value is xxxx
- 'xyy: if x is zero, then the value is 20yy, otherwise it is 1xyy
- 'xy: if x is zero or one, then the value is 20xy, otherwise it is 19xy

3.2.3. DATE

This tokenizer category covers strings of the form *day/month/year*, where *day* and *month* can be one or two digits and *year* can be from two to four digits. The analysis of dates simply takes apart the various parts of the string, by assigning to the attributes *day*, *month* and *year* of the resulting syntactic item suitable numerical values. The year value is obtained in a way similar to the one described in the previous paragraph.

3.2.4. PARAGRAPH-NUMBER

This is analogous to the previous one. Here, the separator is the period instead of the slash, and the different sub-components do not refer to specific items. The result is the assignment to the *value* attribute of the resulting *index* syntactic item of a list of numbers which are the indices of the various subparagraphs: "3.2.11.4" → *value = (3 2 11 4)*

3.2.5. NUMBER

Numbers undergo two different processes depending on the last character. If it is "°", then the number is interpreted as an ordinal adjective, otherwise it is a true number, which can be integer or real. In case of reals, some conversion is made to account for various commas and/or periods. Some examples follow (on the left of the arrow there are character strings, on the right binary numbers):

"3.1415" → 3.1415
"127,715" → 127.715
"3.111,174" → 3111.174
"1,234,567.89" → 1234567.89

Both in cases of ordinals and in case of cardinals, the obtained number is associated to the *value* parameter of the syntactic item[8].

"3.1415" → *"3.1415" (("3.1415" cat num value 3.1415))*
"173°" → *"173°" (("173°" cat adj type ordin gender allval number allval value 173))*

3.2.6. NOMEP (Proper Names)

Since proper names are assumed to be morphologically invariable, they are only looked for in the corresponding dictionary. The only exception is for initials (as "W." and "A." in "W. A. Mozart"): in this case, it is assumed by default (and this "default" choice is specified in the result) that the pair <Capital letter, period> is a proper name.

However, the *default* specification is also used for all words with a capitalized initial. This happens in case there is a *NOMEP* tokenizer hypothesis, which has no match with a name stored in the dictionary. This hypothesis is handled subsequently by the POS tagger, which takes into account the fact that this is only a default.

[8] The string notation adopted here is only partially correct and reflects some peculiarities of the LISP implementation. The actual output consists in *atom names* (atoms are internal LISP data structures) where strings appear in the text above and in LISP numbers (a subclass of *atom*) where there is no string notation. In the printed form of the TUT Treebank, the first string occurrence (the one outside the parentheses, representing the actual input) is without the string quotes, the second one (usually associated with the *lemma*) is enclosed in pipes, while the value occurs as shown (without any marker). Also, in TUT, the double parentheses are replaced by single ones, since the output is obtained after the POS Tagger intervention, which removes ambiguities. The actual TUT lines for the two examples in the text would be:

3.1415 (|3.1415| num 3.1415)
173° (173° adj ordin allval allval 173)

3.3. Ambiguities from tokens and lexicon

There are cases where the search for an item in the dictionaries fails. But since a single tokenizer output may include more than one item (e.g. in case of hyphenated terms, "*term1-term2*", both the hypothesis *((term1-term2 GW))* and the hypothesis *((term1 GW) (- SEGNOINTER) (term2 GW))* are made) it is also possible that some part of the item are unknown (e.g. *term1*) while other are present in the lexicon (e.g. *term2*). Furthermore, a capitalized word can be a known proper name (e.g. *Marianne*) or a default proper name (e.g. *Skrupp*). And if it is at the beginning of a sentence, the non-proper interpretation is also possible.

These cases are taken apart at the end of the previous step (§3.2). The choice is made on the basis of preferences, and its main goal is to exclude unknown items, but a secondary goal is to decide if the capitalized initial has to be maintained or changed into lowercase.

3.4. Multi-word expressions (locutions)

The multi-word expressions covered by the current implementation are of three different types:

a) *FIXED* locutions: sequence of words that are morphologically invariant, in fixed order, without intervening items. Examples are "*by means of*", "*if and only if*" (English), "*a bruciapelo*", "*in preda a*" (Italian)
b) *FLEXIBLE* locutions: sequence of words that may be inflected, but occur in fixed order without intervening words: "*drinking water*", "*interest rate*" (English), "*associazione a delinquere*", "*calcio d'angolo*" (Italian)
c) *VERBAL* locutions: verbs with specific locutions as complements: "*have regard*", "*take into account*" (English), "*rendersi conto*", "*mettere in pratica*" (Italian).

Before seeing how they are coped with, it must be remarked that multi-word expressions cover also proper names (e.g. "*Los Angeles*", "*Banca d'Italia*"), which are, of course *FIXED*, and that hyphenated items, if the hyphen is omitted, must be encoded as multi-word expression. In other words, *left-handed* is a standard dictionary entry, but, if the writing "*left handed*" has to be accepted, it must be specified to be a possible multi-word.

In the following, I will not discuss *VERBAL locutions*, since their analysis is strictly related to the one of verbal subcategorization frames (see Tule Report #1). It must only be observed that the governing verb may take all standard inflected forms associated with its morphological class, and that the "locution" complement can occur everywhere in the sentence (provided that the parser is able to attach it to the right verb). Finally, note that also compound English verbal forms (as "*pick up*", "*take away*") are covered, though in a way slightly different from verbal locutions.

Both *FIXED* and *FLEXIBLE* locutions are recorded in the same dictionary, which is different from the standard dictionary. Each locution is represented as a small automaton, whose nodes have the following format:

```
word ((lemma   CAT syntactic_category
               TYPE syntactic_type
               LOCUT YES
               LOCTYPE FIXED
               ROOT node_ident
               PREV prev_node_ident
               NEXT next_node_ident)
```

The basic attributes of the node are the same as for a standard dictionary entry (*lemma, CAT, TYPE*); *LOCUT* and *LOCTYPE* specify that this is a special entry (i.e. one concerning a multi-word expression); the structure of the automaton is encoded in the *ROOT, PREV* and *NEXT* features, but it is supported also by the *lemma*. In fact, for mnemonic reasons, the node identifiers (*node_ident, prev_node_ident, next_node_ident*) coincide with words. Let's take as an example the full representation of "*by means of*", reported in Box 4.

by ((by_and_out cat prep type poli locut yes root by loctype fixed prev nil next and)
(by_means_of cat prep type poli locut yes root by loctype fixed prev nil next means)
(by_no_means cat adv type neg locut yes root by loctype fixed prev nil next no)
(by_no_way cat adv type neg locut yes root by loctype fixed prev nil next no))
means ***((by_means_of cat prep type poli locut yes root means loctype fixed prev by next of)***
(by_no_means cat adv type neg locut yes root means loctype fixed prev no next nil))
of ((of_course cat adv type manner locut yes root of loctype fixed prev nil next course)
(by_means_of cat prep type poli locut yes root of loctype fixed prev means next nil)
…..)

Box 4 – Representation of the multi-word expression "by means of"

The first word of the locution (by) has *root=by,* and *next=means*, while the second word has *root=means* and *prev=by*. It is clear that the "*by*" root is shared by all other expression where the word "by" occurs. So, the actual processing requires that the *next* of one node (word) coincides with the *root* of the next node (word), but also that the two lemmas (*by_means_of*) are the same. This prevents from accepting as a single multi-word a sequence as "*by no means of*".

Actually, the node identifiers can be any symbol. As stated above, the use of the word itself has just the goal of increasing readability. There is, however, a case where this cannot be done, that is when a word occurs twice in the locution (e.g. *as soon as*). In these cases, the two occurrences must be taken apart: we have adopted the convention that the node associated with the various occurrence are labelled as "word-i" (*as1* and *as2* in "*as soon as*").

APPENDIX A - Iso-latin-1 (ISO/IEC 8859-1) Character encoding

The table below specifies the character codes currently recognized by the tokenizer automaton and used in various modules of the morphological analyser. The table enables the reader to extract both the hexadecimal value (by justaposing the digits in bold of the corresponding line and column) or the decimal value (by summing the numbers enclosed in parentheses in the line and column). For instance, the ampersand (&) has hexadecimal code *26*, which corresponds to the decimal value *38*.

	-0 (0)	**-1** (1)	**-2** (2)	**-3** (3)	**-4** (4)	**-5** (5)	**-6** (6)	**-7** (7)	**-8** (8)	**-9** (9)	**-A** (10)	**-B** (11)	**-C** (12)	**-D** (13)	**-E** (14)	**-F** (15)
0- (0)	NULL						ACKN		BACK	HTAB	LF	VTAB		RET		
1- (16)									CANC			ESC	EOF			
2- (32)	SPACE	**!**	**"**	**#**	**$**	**%**	**&**	**'**	**(**	**)**	*****	**+**	**,**	**-**	**.**	**/**
3- (48)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**:**	**;**	**<**	**=**	**>**	**?**
4- (64)	**@**	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**
5- (80)	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**	**[**	****	**]**	**^**	**_**
6- (96)	**`**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**
7- (112)	**p**	**q**	**r**	**s**	**t**	**u**	**v**	**w**	**x**	**y**	**z**	**{**	**|**	**}**	**~**	DEL
8- (128)																
9- (144)																
A- (160)			**¢**	**£**				**§**		**©**		**«**				
B- (176)	**°**				**´**			**·**				**»**				**¿**
C- (192)	**À**	**Á**	**Â**	**Ã**	**Ä**	**Å**	**Æ**	**Ç**	**È**	**É**	**Ê**	**Ë**	**Ì**	**Í**	**Î**	**Ï**
D- (208)	**Ð**	**Ñ**	**Ò**	**Ó**	**Ô**	**Õ**	**Ö**	**×**	**Ø**	**Ù**	**Ú**	**Û**	**Ü**	**Ý**	**Þ**	**ß**
E- (224)	**à**	**á**	**â**	**ã**	**ä**	**å**	**æ**	**ç**	**è**	**é**	**ê**	**ë**	**ì**	**í**	**î**	**ï**
F- (240)	**ð**	**ñ**	**ò**	**ó**	**ô**	**õ**	**ö**	**÷**	**ø**	**ù**	**ú**	**û**	**ü**	**ý**	**þ**	**ÿ**

Ð, ð: Eth (Icelandic)
Þ, þ: Thorn (Icelandic)
ß: Sharp S (German)
· (B7, middle dot) is the Catalan symbol separating two l's

APPENDIX B – Syntactic categories in the TULE/TUT annotation scheme

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)
11. *PREDET* (predeterminers)
12. *PREP* (prepositions)
13. *PRON* (pronouns)
14. *PUNCT* (punctuation)
15. *SPECIAL* (special symbols)
16. *VERB* (verbs)
17. INDEX (Chapter/paragraph numbers)

APPENDIX C – Dictionaries and Data Structures

The following list refers to the actual files osed by the TULE parser, in particular for the processes described in the present report. All actual data can be found in the corresponding directories of the downloadable system. Each entry of the list includes the name of the file, a possible paragraph number in parentheses, referring to the paragraph of this report where the item is analysed, and, after a colon, the name of the directory where the file is located. The home directory is provided by the path ending with the *SYSTEM* subdirectory. Where *LANGUAGE, LANG or lang* occurs in the name, then the KB is language-dependent and the actual position is determined by substituting the language identifier in the directory names. The identifiers are:
LANGUAGE: ITALIAN, ENGLISH, HINDI, CATALAN, SPANISH
LANG: ITA, ENG, HIN, CAT, SPA
lang: ita, eng, hin, ca, sp

Tokenization:
- token-autom.lisp (§2.4-2.10): ALLLANG/KB-ALL/MORPHO-KB-ALL

Morphology:
- suff-tab-*lang*.dat (§3.1): *LANGUAGE* /KB-*LANG* /MORPHO-KB-*LANG*
- network-*lang*.dat (§3.1): *LANGUAGE* /KB-*LANG* /MORPHO-KB-*LANG*
- numbautom-*lang*.lisp (§3.2.1.2): *LANGUAGE* /KB-*LANG* /MORPHO-KB-*LANG*
  (not available for Spanish)
- enclitinfo-ita.dat: ITALIAN/KB-ITA/MORPHO-KB-ITA
  This records some data about Italian enclitics (also for Catalan and Spanish, not for English, not covered for Hindi)

Lexicon:
- dictionary-*lang*.dat: *LANGUAGE*/KB-*LANG*/DICTION-KB-*LANG*
- dictionary-funct-*lang*.dat: *LANGUAGE*/KB-*LANG*/DICTION-KB-*LANG*

It includes some invariable items (some pronouns, prepositions, conjunctions, etc.). It must be merged with the standard dictionary. This work has only been made for Italian, for which, in fact, this dictionary does not exist.

- locutions-*lang*.dat (§3.4): *LANGUAGE* /KB-*LANG*/DICTION-KB-*LANG*
- proper-names-all.dat: ALLLANG /KB-ALL/DICTION-KB-ALL
- proper-names-*lang*.dat: *LANGUAGE* /KB-*LANG*/DICTION-KB-*LANG*
- proper-comp-all.dat (§3.4): ALLLANG /KB-ALL/DICTION-KB-ALL
- proper-comp-*lang*.dat(§3.4): *LANGUAGE* /KB-*LANG*/DICTION-KB-*LANG*
- invariabili-*lang*.dat: *LANGUAGE* /KB-*LANG*/DICTION-KB-*LANG*
- prep_art-*lang*.dat: *LANGUAGE* /KB-*LANG*/DICTION-KB-*LANG (Italian, Catalan)*
- sigle-*lang*.dat: *LANGUAGE* /KB-*LANG*/DICTION-KB-*LANG*

APPENDIX D – Suffix Tables for Italian, English, and Hindi

ITALIAN

adv

```
( 1 (mente)
  2 (amente)
  3 (iamente)
  4 (emente) )
```

adj

```
      m,sing f,sing  m,pl   f,pl
( 0  ((nil    nil    nil    nil ))          ; *** rosa, bel, chic, gran (115)
  1  ((o      a      i      e   ))          ; *** acerbo, bello, kantiano (2509)
  2  ((e      e      i      i   ))          ; *** abile, mordace, reale (1826)
  3  ((io     ia     i      e   ))          ; *** guercio, malvagio, saggio (25)
  4  ((o      a      hi     he  ))          ; *** antico, fiabesco, lungo (122)
  5  ((o      a      i      he  ))          ; *** alcolico, logico, sismico (984)
  6  ((io     ia     i      ie  ))          ; *** ampio, librario, rivoluzionario (276)
  7  ((a      a      i      e   ))          ; *** buddista, israelita, suicida (58)
  8  ((a      a      i      he  ))          ; *** belga (1)
  9  ((tore   trice  tori   trici ))        ; *** conduttore, regolatore (5)
  20 ((issimo issima issimi issime ))       ; *** superlatives (bell-issimo, grand-issimo)
  21 ((hissimo hissima hissimi hissime))    ; *** superlatives (antic-hissimo)
)
```

noun

```
( 0  ((nil    nil    nil    nil ))          ; *** album, località, mais (1649)
  1  ((nil    a      nil    e   ))          ; *** acqua, memoria, salsa (4147)
  2  ((o      nil    i      nil ))          ; *** chiodo, parametro, suffisso (3563)
  3  ((o      a      i      e   ))          ; *** bambino, filosofo, maestro (333)
  4  ((nil    a      nil    he  ))          ; *** albicocca, grammatica, riga (357)
  5  ((o      nil    hi     nil ))          ; *** albergo, lago, tricheco (224)
  6  ((io     nil    i      nil ))          ; *** cappuccio, avvoltoio, cerchio (922)
  7  ((nil    ia     nil    e   ))          ; *** buccia, faccia, pioggia (97)
  8  ((e      nil    nil    i   ))          ; *** (0)
  9  ((a      a      i      e   ))          ; *** acrobata, congressista, omicida (359)
  10 ((e      nil    i      nil ))          ; *** abete, quintale, torrone (1112)
  11 ((nil    e      nil    i   ))          ; *** deduzione, abitudine, riunione (1015)
  12 ((a      nil    hi     nil ))          ; *** duca, gerarca, patriarca (6)
  13 ((nil    ie     nil    i   ))          ; *** moglie, superficie (2)
  14 ((o      nil    nil    a   ))          ; *** braccio, centinaio, miglio (9)
  15 ((a      nil    i      nil ))          ; *** assioma, barista, schema (127)
  16 ((a      a      hi     he  ))          ; *** collega, oligarca, stratega (4)
  17 ((e      e      i      i   ))          ; *** cantante, cliente, martire (158)
  18 ((e      a      i      e   ))          ; *** buffone, padrone, panettiere (123)
  19 ((o      a      i      he  ))          ; *** amico, biologo, monaco (67)
  20 ((o      a      hi     he  ))          ; *** cuoco, mago, profugo (21)
  21 ((tore   trice  tori   trici ))        ; *** attore, direttore, pescatore (403)
     ; *** The next classes (31, 32, 33, 34, 35, 36, 37) do not appear in the
     ;     dictionary. Entries of these classes are generated automatically by
     ;     the morphological analyser for nouns obtained via verbal nominalization
  30 ((nil    azione nil    azioni ))       ; *** accettazione, adulazione, rifondazione
  31 ((nil    izione nil    izioni ))       ; *** abolizione, deglutizione, elargizione
  32 ((nil    iazione nil   iazioni ))      ; *** conciliazione, dissociazione, variazione
  33 ((amento nil    amenti nil ))          ; *** accantonamento, pedinamento, turbamento
  34 ((imento nil    imenti nil ))          ; *** abbattimento, pentimento, stordimento
  35 ((iamento nil   iamenti nil ))         ; *** incoraggiamento
```

```
      36 ((o      nil    i      nil ))          ; *** addebito, sviluppo, crollo
      37 ((io     nil    i      nil ))          ; *** abbraccio, annuncio, elogio
      38 ((o      nil    hi     nil ))          ; *** blocco, tocco
         ; *** The next classes are as the ones above, but obtained via adjectival
         ;     nominalization
      40 ((nil     ità    nil   ità ))          ; *** agilità, metodicità, reperibilità
      41 ((nil     ietà   nil   ietà ))         ; *** aleatorietà, arbitrarietà, involontarietà
   )
verb
   (1 (     ; *** amare, lavorare, zappare (2468)
       (o     i     a     iamo    ate    ano)                ; (a) ind.pres.
       (avo   avi   ava   avamo   avate  avano)              ; (b) ind.imperf.
       (ai    asti  ò     ammo    aste   arono)             ; (c) pass.rem.
       (erò   erai  erà   eremo   erete  eranno)            ; (d) future
       (i     i     i     iamo    iate   ino)               ; (e) conj.pres.
       (assi  assi  asse  assimo  aste   assero)            ; (f) conj.imperf.
       (erei  eresti erebbe eremmo ereste erebbero)         ; (g) condiz.
       ((ante  ante  anti  anti))                           ; (h) partic.pres.
       ((ato   ata   ati   ate))                            ; (i) partic.pass.
       (ando)                                               ; (l) gerund
       (are)                                                ; (m) infinite
       (nil   a     i     iamo    ate    ino))             ; (n) imperative
    2 (       ; *** combattere, precedere, temere (43)
       (o     i     e     iamo    ete    ono)              ; (a) ind.pres.
       (evo   evi   eva   evamo   evate  evano)            ; (b) ind.imperf.
       ((ei etti) esti (è ette) emmo este (erono ettero))  ; (c) pass.rem.
       (erò   erai  erà   eremo   erete  eranno)            ; (d) future
       (a     a     a     iamo    iate   ano)              ; (e) conj.pres.
       (essi  essi  esse  essimo  este   essero)           ; (f) conj.imperf.
       (erei  eresti erebbe eremmo ereste erebbero)         ; (g) condiz.
       ((ente  ente  enti  enti))                           ; (h) partic.pres.
       ((uto   uta   uti   ute))                            ; (i) partic.pass.
       (endo)                                               ; (l) gerund
       (ere)                                                ; (m) infinite
       (nil   i     a     iamo    ete    ano))             ; (n) imperative
    3 (       ; *** avvertire, partire, servire (35)
       (o     i     e     iamo    ite    ono)              ; (a) ind.pres.
       (ivo   ivi   iva   ivamo   ivate  ivano)            ; (b) ind.imperf.
       (ii    isti  ì     immo    iste   irono)            ; (c) pass.rem.
       (irò   irai  irà   iremo   irete  iranno)           ; (d) future
       (a     a     a     iamo    iate   ano)              ; (e) conj.pres.
       (issi  issi  isse  issimo  iste   issero)           ; (f) conj.imperf.
       (irei  iresti irebbe iremmo ireste irebbero)        ; (g) condiz.
       ((ente  ente  enti  enti))                           ; (h) partic.pres.
       ((ito   ita   iti   ite))                            ; (i) partic.pass.
       (endo)                                               ; (l) gerund
       (ire)                                                ; (m) infinite
       (nil   i     a     iamo    ite    ano))             ; (n) imperative

    4 (       ; *** allagare, cercare, stroncare (424)
       (o     hi    a     hiamo   ate    ano)              ; (a) ind.pres.
       (avo   avi   ava   avamo   avate  avano)            ; (b) ind.imperf.
       (ai    asti  ò     ammo    aste   arono)            ; (c) pass.rem.
       (herò  herai herà  heremo  herete heranno)          ; (d) future
       (hi    hi    hi    hiamo   hiate  hino)            ; (e) conj.pres.
       (assi  assi  asse  assimo  aste   assero)           ; (f) conj.imperf.
       (herei  heresti herebbe heremmo hereste herebbero)  ; (g) condiz.
       ((ante  ante  anti  anti))                           ; (h) partic.pres.
       ((ato   ata   ati   ate))                            ; (i) partic.pass.
       (ando)                                               ; (l) gerund
       (are)                                                ; (m) infinite
       (nil   a     hi     hiamo    ate    hino))          ; (n) imperative
    5 (       ; *** annunciare, baciare, sbocciare (255)
       (io    i     ia    iamo    iate    iano)            ; (a) ind.pres.
       (iavo  iavi  iava  iavamo  iavate  iavano)          ; (b) ind.imperf.
       (iai   iasti iò    iammo   iaste   iarono)          ; (c) pass.rem.
       (erò   erai  erà   eremo   erete   eranno)          ; (d) future
       (i     i     i     iamo    iate    ino)             ; (e) conj.pres.
       (iassi iassi iasse iassimo iaste   iassero)         ; (f) conj.imperf.
       (erei  eresti erebbe eremmo ereste erebbero)        ; (g) condiz.
       ((iante iante ianti ianti))                          ; (h) partic.pres.
       ((iato  iata  iati  iate))                           ; (i) partic.pass.
       (iando)                                              ; (l) gerund
       (iare)                                               ; (m) infinite
       (nil   ia    i     iamo    iate    ino))            ; (n) imperative
    6 (       ; *** dare, dire, fare (some forms) (41)
       (o     ai    à     iamo    ate    anno)             ; (a) ind.pres.
       (avo   avi   ava   avamo   avate  avano)            ; (b) ind.imperf.
```

```
    (i     esti  e     emmo    este   ero)                ; (c) pass.rem.
    (arò   arai  arà   aremo   arete  aranno)             ; (d) future
    (ia    ia    ia    iamo    iate   iano)              ; (e) conj.pres.
    (essi  essi  esse  essimo  este   essero)            ; (f) conj.imperf.
    (arei  aresti arebbe aremmo areste arebbero)         ; (g) condiz.
    ((ante  ante  anti  anti))                           ; (h) partic.pres.
    ((ato   ata   ati   ate))                            ; (i) partic.pass.
    (ando)                                               ; (l) gerund
    (are)                                                ; (m) infinite
    (nil   (ai à) ia    iamo   ate    iano))             ; (n) imperative

 7 (       ; *** addurre, estrarre, porre (some forms) (313)
    (o     i     e     iamo    ete    ono)                ; (a) ind.pres.
    (evo   evi   eva   evamo   evate  evano)             ; (b) ind.imperf.
    (i     esti  e     emmo    este   ero)                ; (c) pass.rem.
    (rò    rai   rà    remo    rete   ranno)              ; (d) future
    (a     a     a     iamo    iate   ano)                ; (e) conj.pres.
    (essi  essi  esse  essimo  este   essero)            ; (f) conj.imperf.
    (rei   resti rebbe remmo   reste  rebbero)           ; (g) condiz.
    ((ente  ente  enti  enti))                           ; (h) partic.pres.
    ((uto   uta   uti   ute))                            ; (i) partic.pass.
    (endo)                                               ; (l) gerund
    (re)                                                 ; (m) infinite
    (nil   i     a     iamo    ete    ano))              ; (n) imperative
 8 (       ; *** accendere, chiudere, leggere (some forms) (1027)
    (o     i     e     iamo    ete    ono)                ; (a) ind.pres.
    (evo   evi   eva   evamo   evate  evano)             ; (b) ind.imperf.
    (i     esti  e     emmo    este   ero)                ; (c) pass.rem.
    (erò   erai  erà   eremo   erete  eranno)            ; (d) future
    (a     a     a     iamo    iate   ano)                ; (e) conj.pres.
    (essi  essi  esse  essimo  este   essero)            ; (f) conj.imperf.
    (erei  eresti erebbe eremmo ereste erebbero)         ; (g) condiz.
    ((ente  ente  enti  enti))                           ; (h) partic.pres.
    ((o     a     i     e))                              ; (i) partic.pass.
    (endo)                                               ; (l) gerund
    (ere)                                                ; (m) infinite
    (nil   i     a     iamo    ete    ano))              ; (n) imperative
 9 (       ; *** bollire, inserire, sfoltire (363)
    (isco  isci  isce  iamo    ite    iscono)             ; (a) ind.pres.
    (ivo   ivi   iva   ivamo   ivate  ivano)             ; (b) ind.imperf.
    (ii    isti  ì     immo    iste   irono)              ; (c) pass.rem.
    (irò   irai  irà   iremo   irete  iranno)            ; (d) future
    (isca  isca  isca  iamo    iate   iscano)            ; (e) conj.pres.
    (issi  issi  isse  issimo  iste   issero)            ; (f) conj.imperf.
    (irei  iresti irebbe iremmo ireste irebbero)         ; (g) condiz.
    ((ente  ente  enti  enti))                           ; (h) partic.pres.
    ((ito   ita   iti   ite))                            ; (i) partic.pass.
    (endo)                                               ; (l) gerund
    (ire)                                                ; (m) infinite
    (nil   isci  isca  iamo    ite    iscano))           ; (n) imperative
10 (       ; *** essere, sapere (some forms) (2)
    (ono   ei    nil   iamo    iete   ono)                ; (a) ind.pres.
    (o     i     a     avamo   avate  ano)                ; (b) ind.imperf.
    (i     sti   nil   mmo     ste    rono)               ; (c) pass.rem.
    (arò   arai  arà   aremo  arete  aranno)             ; (d) future
    (ia    ia    ia    iamo    iate   iano)              ; (e) conj.pres.
    (ssi   ssi   sse   ssimo   ste    ssero)             ; (f) conj.imperf.
    (arei  aresti arebbe aremmo areste arebbero)         ; (g) condiz.
    ((ente  ente  enti  enti))                           ; (h) partic.pres.
    ((ato   ata   ati   ate))                            ; (i) partic.pass.
    (endo)                                               ; (l) gerund
    (ere)                                                ; (m) infinite
    (nil   ii    ia    iamo    ate    iano))             ; (n) imperative

11 (       ; *** abbagliare, ammaliare, scambiare (275)
    (io    i     ia    iamo    iate    iano)             ; (a) ind.pres.
    (iavo  iavi  iava  iavamo  iavate  iavano)           ; (b) ind.imperf.
    (iai   iasti iò    iammo   iaste   iarono)           ; (c) pass.rem.
    (ierò  ierai ierà  ieremo  ierete  ieranno)          ; (d) future
    (i     i     i     iamo    iate    ino)              ; (e) conj.pres.
    (iassi iassi iasse iassimo iaste   iassero)          ; (f) conj.imperf.
    (ierei  ieresti ierebbe ieremmo iereste ierebbero)   ; (g) condiz.
    ((iante iante ianti ianti))                          ; (h) partic.pres.
    ((iato  iata  iati  iate))                           ; (i) partic.pass.
    (iando)                                              ; (l) gerund
    (iare)                                               ; (m) infinite
    (nil   ia    i     iamo    iate    ino))             ; (n) imperative
```

```
 12 (       ; *** dire (1) Introduced for the imperative "di'"
     (o     i     e     iamo    ite     ono)                ; (a) ind.pres.
     (evo   evi   eva   evamo   evate   evano)              ; (b) ind.imperf.
     (i     esti  e     emmo    este    ero)                ; (c) pass.rem.
     (irò   irai  irà   iremo   irete   iranno)             ; (d) future
     (a     a     a     iamo    iate    ano)                ; (e) conj.pres.
     (essi  essi  esse  essimo  este    essero)             ; (f) conj.imperf.
     (irei  iresti irebbe iremmo ireste irebbero)           ; (g) condiz.
     ((ente  ente  enti  enti))                             ; (h) partic.pres.
     ((o     a     i     e))                                ; (i) partic.pass.
     (endo)                                                 ; (l) gerund
     (ire)                                                  ; (m) infinite
     (nil    ì     a     iamo    ite     ano))              ; (n) imperative
```

ENGLISH

adv

```
  ( 1 (ly)
    2 (ily)
    3 (ably)
    4 (ibly)
    5 (ubly)
    6 (ically)
    7 (y)  )
```

adj

```
     ; *** classes 2, 3, 4, 5 are used to generate the corresponding adverbs
        m,sing f,sing  n,sing m,pl   f,pl   n,pl
  ( 0 (((nil    nil    nil    nil    nil    nil))     ; *** crazier, craziest (1186)
    1  ((@empt  @empt  @empt  @empt  @empt  @empt)    ; *** bayesian, candy-scented (11497)
    2  ((y      y      y      y      y      y)        ; *** empty, stationary (1517)
    3  ((able   able   able   able   able   able)     ; *** admirable, movable (876)
    4  ((ible   ible   ible   ible   ible   ible)     ; *** eligible, extensible (164)
    5  ((uble   uble   uble   uble   uble   uble)     ; *** soluble, voluble (12)
    6  ((ic     ic     ic     ic     ic     ic)       ; *** alcoholic, terrific (1365)
    7  ((ic     ic     ic     ic     ic     ic)       ; *** aesthetic, numeric (310)
    7  ((ical   ical   ical   ical   ical   ical)     ;      same as above
```

noun

```
 (0 (nil   nil   nil   nil   nil   nil)   ; ***
  1 (nil   @empt nil   nil   s     nil)   ; *** feminine
  2 (@empt nil   nil   s     nil   nil)   ; *** masculine
  3 (nil   nil   @empt nil   nil   s)     ; *** neuter
  4 (nil   @empt nil   nil   es    nil)   ; *** feminine
  5 (@empt nil   nil   es    nil   nil)   ; *** masculine
  6 (nil   nil   @empt nil   nil   es)    ; *** neuter
  7 (nil   y     nil   nil   ies   nil)   ; *** feminine
  8 (y     nil   nil   ies   nil   nil)   ; *** masculine
  9 (nil   nil   y     nil   nil   ies)   ; *** neuter
  30 (nil  nil   ization nil nil izations)  ; *** neuter
  31 (nil  nil   ion   nil   nil   ions)  ; *** neuter
  ; also the next are covered by class 30:
  ;             (nil nil isation nil nil isations)
  ; this is obtained by changing s into z into the suffix network
 )
```

verb

```
 (0 ()      ; ***
  1 (       ; *** regular (love, walk, ...)
     (@empt  @empt s     @empt   @empt  @empt)             ; (a) ind.pres.
     (ed     ed    ed    ed      ed     ed)               ; (b) ind.past
     (@empt  @empt @empt @empt   @empt  @empt)             ; (e) subj.pres.
     (ed     ed    ed    ed      ed     ed)               ; (f) subj.past
     ((ing   ing   ing   ing     ing    ing))             ; (h) partic.pres.
     ((ed    ed    ed    ed      ed     ed))              ; (i) partic.past
     (ing)                                                ; (l) gerund
     (@empt)                                              ; (m) infinite
     (nil    @empt @empt @empt   @empt  @empt))           ; (n) imperative
  2 (       ; *** irregular (buy, eat, ...)
     (@empt  @empt s     @empt   @empt  @empt)             ; (a) ind.pres.
     (@empt  @empt @empt @empt   @empt  @empt)             ; (b) ind.past
     (@empt  @empt @empt @empt   @empt  @empt)             ; (e) subj.pres.
     (@empt  @empt @empt @empt   @empt  @empt)             ; (f) subj.imperf.
     ((@empt @empt @empt @empt   @empt  @empt))            ; (h) partic.pres
     ((@empt @empt @empt @empt   @empt  @empt))            ; (i) partic.pass.
     (@empt)                                              ; (l) gerund
     (@empt)                                              ; (m) infinite
     (nil    @empt @empt @empt   @empt  @empt))           ; (n) imperative
  3 (       ; *** have
```

```
   (ve     ve    s     ve      ve     ve)                ; (a) ind.pres.
   (d      d     d     d       d      d)                 ; (b) ind.past
   (ve     ve    ve    ve      ve     ve)                ; (e) subj.pres.
   (d      d     d     d       d      d)                 ; (f) subj.past
   ((ving  ving  ving  ving    ving   ving))             ; (h) partic.pres.
   ((d     d     d     d       d      d))                ; (i) partic.pass.
   (ving)                                                ; (l) gerund
   (ve)                                                  ; (m) infinite
   (nil    ve    ve    ve      ve     ve))               ; (n) imperative
4 (       ; *** be
   (am     are   is    are     are    are)               ; (a) ind.pres.
   (was    were  was   were    were   were)              ; (b) ind.past
   (be     be    be    be      be     be)                ; (e) subj.pres.
   (were   were  were  were    were   were)              ; (f) subj.past
   ((being being being being   being  being))           ; (h) partic.pres.
   ((been  been  been  been    been   been))            ; (i) partic.past
   (being)                                               ; (l) gerund
   (be)                                                  ; (m) infinite
   (nil    be    be    be      be     be))               ; (n) imperative
5 (       ; *** harmonize, harmonise (all below, accepted with "s" in place of "z")
   (ize    ize   izes  ize     ize    ize)               ; (a) ind.pres.
   (ized   ized  ized  ized    ized   ized)              ; (b) ind.past
   (ize    ize   ize   ize     ize    ize)               ; (e) subj.pres.
   (ized   ized  ized  ized    ized   ized)              ; (f) subj.past
   ((izing izing izing izing   izing  izing))            ; (h) partic.pres.
   ((ized  ized  ized  ized    ized   ized))             ; (i) partic.past
   (izing)                                               ; (l) gerund
   (ize)                                                 ; (m) infinite
   (nil    ize   ize   ize     ize    ize))              ; (n) imperative
6 (       ; *** aerify, amplify
   (y      y     ies   y       y      y)                 ; (a) ind.pres.
   (ied    ied   ied   ied     ied    ied)               ; (b) ind.past
   (y      y     y     y       y      y)                 ; (e) subj.pres.
   (ied    ied   ied   ied     ied    ied)               ; (f) subj.past
   ((ying  ying  ying  ying    ying   ying))             ; (h) partic.pres.
   ((ied   ied   ied   ied     ied    ied))              ; (i) partic.past
   (ying)                                                ; (l) gerund
   (y)                                                   ; (m) infinite
   (nil    y     y     y       y      y))                ; (n) imperative

7 (       ; *** accuse, activate
   (e      e     es    e       e      e)                 ; (a) ind.pres.
   (ed     ed    ed    ed      ed     ed)                ; (b) ind.past
   (e      e     e     e       e      e)                 ; (e) subj.pres.
   (ed     ed    ed    ed      ed     ed)                ; (f) subj.past
   ((ing   ing   ing   ing     ing    ing))              ; (h) partic.pres.
   ((ed    ed    ed    ed      ed     ed))               ; (i) partic.past
   (ing)                                                 ; (l) gerund
   (e)                                                   ; (m) infinite
   (nil    e     e     e       e      e))                ; (n) imperative
8 (       ; encompass, possess
   (@empt  @empt es    @empt   @empt  @empt)             ; (a) ind.pres.
   (ed     ed    ed    ed      ed     ed)                ; (b) ind.past
   (@empt  @empt @empt @empt   @empt  @empt)             ; (e) subj.pres.
   (ed     ed    ed    ed      ed     ed)                ; (f) subj.past
   ((ing   ing   ing   ing     ing    ing))              ; (h) partic.pres.
   ((ed    ed    ed    ed      ed     ed))               ; (i) partic.past
   (ing)                                                 ; (l) gerund
   (@empt)                                               ; (m) infinite
   (nil    e     e     e       e      e))                ; (n) imperative
9 (       ; *** modals: can, may, will
   (@empt  @empt @empt @empt   @empt  @empt)             ; (a) ind.pres.
   (nil    nil   nil   nil     nil    nil)               ; (b) ind.past
   (@empt  @empt @empt @empt   @empt  @empt)             ; (e) subj.pres.
   (nil    nil   nil   nil     nil    nil)               ; (f) subj.past
   ((nil   nil   nil   nil     nil    nil))              ; (h) partic.pres.
   ((nil   nil   nil   nil     nil    nil))              ; (i) partic.past
   (nil)                                                 ; (l) gerund
   (@empt)                                               ; (m) infinite
   (nil    @empt @empt @empt   @empt  @empt))            ; (n) imperative
```

HINDI

In all adjectival and nominal tables, the first row of each class refers to the *direct* case, the second one to the *oblique* case.

adj

```
            m,sing f,sing  m,pl   f,pl
       ( 0  ((nil    nil    nil    nil)
             (nil    nil    nil    nil))
         1  ((@empt  @empt  @empt  @empt)                ; *** sarala (simple), amIra ()
             (@empt  @empt  @empt  @empt))
         2  ((A      I      e      I)                    ; *** usak-A (her/his)
             (e      I      e      I))
         3  ((A      A      e      i)                    ; *** kAl-A (black)
             (e      I      e      i))
         4  ((e      I      e      i)                    ; *** acc-A (good)
             (e      I      e      i))
         5  ((a      a      a      a)                    ; *** anek-a (many)
             (oM     oM     oM     oM))
       )
```

noun

```
       ( 1  ((A      I      e      iyAz)                 ; *** laDak-A (boy-girl)
             (e      I      oM     iyoM))                ;     both masculine and feminine
         2  ((@empt  nil    @empt  nil)                  ; *** kroXa- (resentment), pAnI- ()
             (@empt  nil    @empt  nil))
         3  ((@empt  nil    @empt  nil)                  ; *** rAjA- (),BARAvix- ()
             (@empt  nil    oM     nil))
         4  ((A      nil    e      nil)                  ; *** kamr-A (room)
             (e      nil    oM     nil))
         5  ((A      nil    e      nil)                  ; *** loh-A (iron)
             (e      nil    e      nil))
         6  ((a      nil    a      nil)                  ; *** Karc-a ()
             (e      nil    oM     nil))
         7  ((a      nil    a      nil)                  ; *** Gar-a (home)
             (a      nil    oM     nil))
         8  ((Az     nil    Ez     nil)                  ; *** ku-Az (well)
             (Az     nil    oM     nil))
         9  ((i      nil    i      nil)                  ; *** kav-i ()
             (i      nil    iYoM   nil))
         10 ((I      nil    I      nil)                  ; *** Axam-I ()
             (I      nil    iYoM   nil))
         11 ((U      nil    U      nil)                  ; *** Al-U ()
             (U      nil    uoM    nil))
         12 ((Uz     nil    Uz     nil)                  ; *** geh-Uz ()
             (Uz     nil    uzoM   nil))
         13 ((nil    @empt  nil    @empt)                ; *** BIda- (), IrRyA- (), SAnwi- (),
             (nil    @empt  nil    @empt))               ;     vAyu- (), bAlU- (), sarasoM- ()
         14 ((nil    a      nil    a)                    ; *** rAw-a ()
             (nil    eM     nil    oM))                  ; *** ASA- ()
         15 ((nil    A      nil    A)                    ; *** gudiy-A ()
             (nil    Az     nil    oM))
         16 ((nil    @empt  nil    @empt)
             (nil    (yeM ez) nil  oM))
         17 ((nil    @empt  nil    @empt)                ; *** Apawwi- ()
             (nil    yAz    nil    yoM))
         18 ((nil    @empt  nil    @empt)                ; *** qwu- ()
             (nil    ez     nil    oM))
         19 ((nil    U      nil    U)                    ; *** bah-U ()
             (nil    (uyeM uez) nil uoM))
         20 ((nil    @empt  nil    @empt)                ; *** lO- ()
             (nil    ez     nil    voM))
         21 ((nil    @empt  nil    @empt)                ; *** mAz- ()
             (nil    yeM    nil    oM))
         22 ((nil    @empt  nil    @empt)                ; *** pot-o ()
             (nil    ez     nil    oM))
       )
```

verb

```
    ; *** each line refers to one mood-tense combination of a given paradigm
    ;     It is included in double parentheses, to account for the possibility that it
    ;     be case-inflected. If no case is involved, a single sub-parenthesis occurs
   (1 (                              ; *** KA- (eat)
       ((yA           (e ye)         (I yI)         (IM yIM)))     ; (a) base past       (yA)
       ((UzgA         (egA yegA)     (iegA iyegA)   (egA yegA)     ; (b) base future m sing
         (ezge yeMge) oge            (iegA iyegA)   (ezge yeMge)   ;     base future m pl
         UzgI         (egI yegI)     (iegA iyegA)   (egI yegI)     ;     base future f sing
         (ezgI yeMgI) ogI            (iegA iyegA)   (ezgI yeMgI))) ;     base future f pl
       ((Uz           (e ye)         (ie iye)       (e ye)         ; (c) subj sing
```

```
     (ez yeM|)     o             (ie iye)       (ez yeM)))     ;     subj pl
   ((kara))                                                    ; (h) participle pres
   ((wA            we            wI              wIM))         ; (i) participle past (wA)
   ((nA))                                                      ; (l) infinitive pres
   ((nA            ne            nI              nIM)))        ; (m) infinitive past (nA)
2 (                              ; *** k-ara ()
  ((iyA           (ie iye)       I              IM))           ; (a) base past       (yA)
   ((arUzgA       aregA          (ariegA IjiyegA) aregA        ; (b) base future m sing
     areMge       aroge          (ariegA IjiyegA) areMge       ;     base future m pl
     arUzgI       aregI          (ariegA IjiyegA) aregI        ;     base future f sing
     areMgI       arogI          (ariegA IjiyegA) areMgI))     ;     base future f pl
   ((arUz         are            (arie Ijiye)   are            ; (c) subj sing
     areM         aro            (arie Ijiye)   areM))         ;     subj pl
   ((arara))                                                   ; (h) participle pres
   ((arawA        arawe          arawI          arawIM))       ; (i) participle past (wA)
   ((aranA))                                                   ; (l) infinitive pres
   ((aranA        arane          aranI          aranIM|)))     ; (m) infinitive past (nA)
3 (                              ; *** l-e ()
   ((iyA          (ie iye)       I              IM))           ; (a) base past       (yA)
   ((UzgA         egA            (IjiegA IjiyegA) egA          ; (b) base future m sing
     eMge         oge            (IjiegA IjiyegA) eMge         ;     base future m pl
     UzgI         egI            (IjiegA IjiyegA) egI          ;     base future f sing
     eMgI         ogI            (IjiegA IjiyegA) eMgI))       ;     base future f pl
   ((Uz           e              (Ijie Ijiye)   e              ; (c) subj sing
     eM           o              (Ijie Ijiye)   eM))           ;     subj pl
   ((ekara))                                                   ; (h) participle pres
   ((ewA          ewe            ewI            ewIM))         ; (i) participle past (wA)
   ((enA))                                                     ; (l) infinitive pres
   ((enA          ene            enI            enIM))         ; (m) infinitive past (nA)
4 (                              ; *** h-o ()
   ((uA           (ue uye)       (uI uyI)       (uIM uyIM)))   ; (a) base past       (yA)
   ((oUzgA        ogA            (oiegA oiyegA) ogA            ; (b) base future m sing
     oMge         oge            (oiegA oiyegA) oMge           ;     base future m pl
     oUzgI        ogI            (oiegA oiyegA) ogI            ;     base future f sing
     oMgI         oogI           (oiegA oiyegA) oMgI))         ;     base future f pl
   ((oUz          o              (oie oiye)     o              ; (c) subj sing
     (oez oyeM)   oo             (oie oiye)     (oez oyeM)))   ;     subj pl
   ((okara))                                                   ; (h) participle pres
   ((owA          owe            owI            owIM))         ; (i) participle past (wA)
   ((onA))                                                     ; (l) infinitive pres
   ((onA          one            onI            onIM)))        ; (m) infinitive past (nA)
5 (                              ; *** p-I ()
   ((iyA          (ie iye)       I              IM))           ; (a) base past       (yA)
   (((iUzgA iyUzgA) (iegA iyegA)(IjiegA IjiyegA)(iega iyeMgA) ; (b) base future m sing
     (iezge iyeMge) ioge         (IjiegA IjiyegA)(iezge iyeMge);     base future m pl
     (iyUzgI iuzgI) (iegI iyegI)(IjiegA IjiyegA)(iegI iyegI)  ;     base future f sing
     (iezgI iyeMgI) iogI         (IjiegA IjiyegA)(iezgI iyeMgI)));   base future f pl
   ((iUz          (ie iye)       (Ijie Ijiye)   (ie iye)       ; (c) subj sing
    (iez iyeM)    io             (Ijie Ijiye)   (iez iyeM)))   ;     subj pl
   ((ikara))                                                   ; (h) participle pres
   ((iwA          iwe            iwI            iwIM))         ; (i) participle past (wA)
   ((inA))                                                     ; (l) infinitive pres
   ((inA          ine            inI            inIM)))        ; (m) infinitive past (nA)
6 (                              ; *** C-U ()
   ((uA           (ue uye)       (uI uyI)       (uIM uyiM)))   ; (a) base past       (yA)
   ((uUzgA        (uegA uyegA)   (uiegA uiyegA) (uegA uyegA)   ; (b) base future m sing
     (uezge uyeMge) uoge         (uiegA uiyegA) (uezge uyeMge) ;     base future m pl
     uUzgI        (uegI uyegI)   (uiegA uiyegA) (uegI uyegI)   ;     base future f sing
     (uezgI uyeMgI) uogI         (uiegA uiyegA) (uezgI uyeMgI)));    base future f pl
   ((uUz          (ue uye)       (uie uiye)     (ue uye)       ; (c) subj sing
     (uez uyeM)   uo             (uie uiye)     (uez uyeM)))   ;     subj pl
   ((ukara))                                                   ; (h) participle pres
   ((uwA          uwe            uwI            uwIM))         ; (i) participle past (wA)
   ((unA))                                                     ; (l) infinitive pres
   ((unA          une            unI            unIM)))        ; (m) infinitive past (nA)
7 (                              ; *** uT-a ()
   ((A            e              I              IM))           ; (a) base past       (yA)
   ((UzgA         egA            (iegA iyegA)   egA            ; (b) base future m sing
     eMge         oge            (iegA iyegA)   eMge           ;     base future m pl
     UzgI         egI            (iegA iyegA)   egI            ;     base future f sing
     eMgI         ogI            (iegA iyegA)   eMgI))         ;     base future f pl
   ((Uz           e              (ie iye)       e              ; (d) subj sing
     eM           o              (ie iye)       eM))           ;     subj pl
   ((kara))                                                    ; (h) participle pres
   ((wA           we             wI             wIM))          ; (i) participle past (wA)
   ((nA))                                                      ; (l) infinitive pres
   ((nA           ne             nI             nIM)))         ; (m) infinitive past (nA)
8 (                              ; *** so- ()
```

```
((yA           (e ye)         (I yI)         (IM yIM)))     ; (a) base past       (yA)
((UzgA         (egA yegA)     (iegA iyegA)   (ega yegA)     ; (b) base future m sing
  yeMge        oge            (iegA iyegA)   yeMge          ;     base future m pl
  UzgI         yegI           (iegA iyegA)   yegI           ;     base future f sing
  yeMgI        ogI            (iegA iyegA)   yeMgI))        ;     base future f pl
((Uz           @empt          (ie iye)       e              ; (c) subj sing
  (ez yeM)     o              (ie iye)       (ez yeM)))     ;     subj pl
((kara))                                                   ; (h) participle pres
((wA           we             wI             wIM))          ; (i) participle past (wA)
((nA))                                                     ; (l) infinitive pres
((nA           ne             nI             nIM)))         ; (m) infinitive past  nA)
```