MACO: Morphological Analyzer Corpus-Oriented

Sofía Acebo        Alicia Ageno
Salvador Climent  Javier Farreres
Lluís Padró         Francesc Ribas
Horacio Rodríguez  Oscar Soler

Report LSI-94-30-R
MACO: Morphological Analyser
Corpus-Oriented

Sofía Acebo*, Alicia Ageno, Salvador Climent,
Javier Farreres¹, Lluís Padró, Francesc Ribas¹,
Horacio Rodríguez, Oscar Soler*.

Departament de Llenguatges i Sistemes Informàtics.
Universitat Politècnica de Catalunya.
Pau Gargallo 5, 08028 Barcelona
horacio@lsi.upc.es

(*) Universitat de Barcelona.

¹ These researchers have been supported by a grant of the Departament d’Ensenyament de la
Generalitat de Catalunya.
MACO: Morphological analyser Corpus-Oriented

Sofía Acebo, Alicia Ageno, Salvador Climent, Javier Farreres, Lluís Padró, Francesc Ribas, Horacio Rodríguez, Oscar Soler.

Abstract

This document describes the organisation, functional capabilities and performance of MACO, a tool for morphological analysis of free text that allows the integration, in a cooperative mode, of different already existing linguistic Knowledge Sources.

1. Introduction.

The present document describes MACO, a tool for morphological analysis of corpora. MACO (Morphological analyser Corpus-Oriented) has been designed to attach as much morphological information as possible (of course the part of speech but also other information, depending on the linguistic source) to every word in the input text.

MACO has been conceived and designed as a general purpose morphological tool although the current implementation of the system (and the involved Data Sources) is devoted to the morphological analysis of Spanish texts.

MACO has been built, within Acquilex II Project, as a part of Working Part 4.3 (Corpus Tools) in order to allow an easy and robust way of attaching initial sets of part of speech tags to every word in the input text. Morphological analysis can be considered as a first step for most corpus-based processes (like pos-tagging, semantic-tagging, syntactic-tagging, and so on) and experiments. [Moreno 93] and [Briscoe et al. 94] present various applications where the results obtained from MACO have been used.

The structure of this paper is the following: After this introduction, section 2 states some general background and motivation. Section 3 presents a general overview of MACO. On section 4 we can find a detailed description of MACO architecture: First the manager is described and then the different morphological processors. Section 5 is devoted to data acquisition for the different processors. On section 6 the performance of MACO is analysed by means of several experiments. Section 7, finally, states some conclusions and future developments.

2. Background and motivation.

2.1. Background.

Why is a morphological analyser (MA) needed? Many corpus tools, when working with English, do not include a morphological module. In these systems the identification of lexical forms is carried out by means of dictionary look-up, sometimes complemented with a set of rather simple lexical rules. In other languages with richer compositional, flexive or derivative productivity the use of a mechanism for factoring common information among a set of lexical forms seems to be necessary, at least for broad-coverage applications, like corpus analysis.
There are two reasons for such a need:

- **Storage.** It seems obvious that for languages with a high flexive and/or derivative capability, like Spanish, the distribution of information among the different fragments of the word counts for a great saving of storage (e.g. the average of verbal forms, in Spanish, corresponding to each stem is over 30, and most of them differ only on a few features, tense, person, number, etc.).

- **Acquisition.** It is difficult and labour consuming to provide manually all the allowed forms for each item. Some automatic approaches can be considered as an alternative. MRDs provide, usually, only fragmentary information and the extraction of such information is not always easy. In most dictionaries information is given only for lemmas while the other forms and their information appear distributed among flexible suffixes, examples, typification of verbs etc. Inferring morphological information for unknown words from text corpora might be a good solution but needs more experimentation, it has a rather low upper bound of success, and requires a rather broad initial set of assigned categories. Assigning word forms to previously defined morphological patterns could be, thus, a valuable solution.

There are, of course, limitations and disadvantages on using MA. The most important is that the task might be computationally expensive, depending on the kind of tool and the characteristics of involved phenomena. We need, then, to balance the trade-off between advantages and limitations arriving often to hybrid solutions: e.g. using MA during the acquisition phase but storing for run time access all the possible forms, using MA only for rare cases, and so on.

It is outside the scope of this paper to give a survey (even a short one) of current morphological analysis techniques (for extensive surveys see [Martí 88] and [Sproat 91]) but some general background knowledge is introduced below as a basis for discussing and understanding the different approaches that will be presented in section 4.2.

There are two dimensions when facing the problem of morphology: the pieces or constituents of the word and the word formation rules. The different systems and techniques differ on the importance they assign to each of these dimensions.

Words can be made up by means of concatenation or adjunction of simpler components. The former operation is rather infrequent in Spanish and can be solved straightforwardly (words like “rascacielos” -skyscraper- can be managed without taking into account its internal structure). However, the latter is the main operation of any MA.

Consider, for instance the word "adormecedoras" (sleep-inducing). This word can be decomposed in the following way:

```
"a"           prefix
"dorm"        stem
"ecedor"      derivative (adjectival) suffix
"a"           flexive (feminine) suffix
"s"           flexive (plural) suffix
```

Of course not all the combinations of stems and affixes (morphemes) are correct, i.e. form an actual word. Stems and affixes can be typified and valid combinations can be stated by means of grammar rules. Depending on the scope of such morphological rules we can speak of one-level, two-level (or recently three-level) MAs.
The most common MAs are two level (see [Koskenniemi 83] or [Ritchie et al. 90]). These MAs perform like transducers (often Finite State Transducers) between two levels: a surface level and a lexical level. The input to the transducer (the surface level) corresponds to the word to be analyzed and the output (the lexical level) to the lexeme. Lexical (formation) rules describe conditions under which fragments of the input (letters) are consumed and fragments of the output are produced.

One level MAs perform only at surface level. Rules, in this case, describe conditions under which the different morphemes can be combined. A great variety of techniques can be used for such purpose: finite state automata, CFG, transition matrices, etc.

2.2. Motivation.

When facing the problem of building MACO we stated a set of principles for guiding our design choices:

- **Flexibility**: We wanted a tool able to incorporate already existing MAs, competing or collaborating, covering different (usually overlapping) morphological phenomena, with different depth and coverage, with different granularity, with different tag sets, etc.

- **Simplicity** of use and control.

- **Robustness**: In order to deal with morphological analysis of large corpora we needed a robust MA, able to assign always the most precise information to each word (including perhaps default features) with no (or a minor) human intervention.

- **Reusability**: The system should be able to use as many of existing knowledge sources for the language being dealt with.

- **Efficiency**: Although MACO is a tool designed to run basically in batch mode some constraints on the efficiency are required due to the volume of information to be processed.

- **Incrementality**: The system should be updated in an easy way. New words, or word families, new general or specific data sources etc. should be incorporated without problems in any point without affecting the previously analyzed texts.

3. General Overview

The MACO system accepts as input a raw corpus and produce as output the morphological analyses of the words in the corpus. The architecture of the system is divided in three parts (see figure 1): preprocessor, MACO morphological process, and finally post-processor.
The preprocessor tokenizes the corpus cleaning up formatting stuff (SGML tags, ...), separating the punctuation signs from the preceding word, and finally reconstructing dashed words at the end of line. Later on, it extracts one item of each word form from the corpus in order to generate the input list to the MACO morphological processor.

The MACO morphological processor (from now on called simply MACO) accepts a list of word forms, and outputs two lists: (a) the analyzed words plus the analyses, and (b) the list of words for which MACO failed (unknown words).

Finally, the post-processor prepares the output for the application that is going to use the generated morphological analyses. Due to the fact that each application may be interested in a different information and format this module should be programmed specifically by the user. Up to now we have developed a post-processor for connecting MACO to the [Briscoe et al 94] and
4. Architecture of MACO.

One of the main aims of MACO was to reuse in an opportunistic manner existing morphological tools, covering different phenomena of the Spanish language. The idea was to use those knowledge sources without having to modify in any significant way neither their code nor their category set. Another related aim of the project was to do the integration of the different data sources in such a way that it should be easy to change the strategy to apply the analysers and combine their results.

A modular approach was chosen because of the need to incorporate rather different processors and data sources. The different analysers are considered as black-boxes, so, given a word form, they return the set of possible morphological analyses. MACO operation is controlled by a manager process that indicates when to load every analyser, when to apply it and how to translate its results to the common category set. The manager has access to a data structure, the analysers ordering list, where the available analysers and the order to apply them are stated.

Therefore, the flexibility on adding new morphological knowledge sources, as they become available, is guaranteed. The only requirement to include a new analyser is to provide two functions, one to fulfil the interface with the manager and another function to load the needed material. On the other hand, a modification on the operation of any analyser shouldn’t affect the work of others.

Another goal of MACO is to provide a simple and natural way of increasing the existing sources of Spanish lexical knowledge, allowing the user to introduce analyses for the words that are unknown by the analysers included in the system. New analyses resulting from the processing of a particular corpus would be stored to use them whenever needed in a further occasion. This functionality has been designed as a separate black-box module that is called following the manager strategy, usually when all the analysers have failed. It prompts the user to provide categories for the unsuccessful words in a menu-guided user-friendly way. Besides using the analysis provided by the user in the current corpus, it stores the results as a knowledge source to be consulted in subsequent uses of MACO.

On the other hand, MACO allows the user to define the control of the analysers performance in three ways:

- Which of the available analysers to use.
- In which order they should be applied.
- What strategy to follow on applying them (all, where all analysers are applied and their results combined, or backing-off, only words not analyzed by previous analysers are tried on subsequent ones).

The user performs the definition of these control alternatives in a declarative way.

Another important point about MACO is that it allows the user to abort the execution at any desired moment and resume the job at that point. This is implemented using entry points which are periodically saved in a status file.

Finally, there is an option for identifying the source of each analysis, labelling each result with the name of the analyser where it was from. This functionality is be useful specially on the development stage where an optimal combination of analysers is sought.
A diagram of the MACO architecture is shown in figure 2. The input to MACO is a list of word forms, and the output is composed of two lists: (a) the analyzed words with their corresponding morphological analyses, and (b) the list of words in which all the analysers failed (unknown words). The analysis process is controlled by the manager, who loads and applies the analysers in the order and strategy defined by the user in the analysers order list. Finally, the category set unifier module translates the different sets of categories used by each analyser into a common set.

Section 4.1 is devoted to the description of the manager, whilst section 4.2 describes the different analysers for Spanish used up to now.
4.1. The manager

Mainly due to space and efficiency reasons (e.g. the different analysers loaded in memory at the same time would consume too much space) analysers are applied grouped in stages. Consequently, at each stage only the analysers to be applied are loaded in memory, and once the stage is finished they are unloaded. The manager applies in an iterative manner the different stages to the input list of words, until all the stages have been tried.

A parallel strategy is only applied intra-stages (all the analysers of the same stage are tried), but not inter-stages, where a backing-off strategy is used\(^2\) (only words not analyzed by previous stages are tried by subsequent ones).

Therefore, the data structure defining the analysers ordering list consists of two levels, a sequence of stages, and for each stage a list of analysers to be applied. In figure 3 an example of such a data structure with four stages defined is shown. In this example, in case the application strategy was parallel, the analysers applied in the first stage would be: (a) Formario (lookup on a list of analyzed word forms), (b) Number (number analyser). Their results would be combined and the unknown words would be tried in the second stage. In the second and the third step only the SegWord and Amcas analysers respectively would be applied. Finally, in the fourth stage, the analysers applied would be: (a) Accumulate (lookup in the list of analyses provided by the user in previous interactions), (b) Proper-noun (lookup in a list of proper-nouns), (c) Initial (heuristic providing the proper-noun analysis if the initial letter is upper-case), and finally, (c) Default-cats (default categories for unknown words).

```
stage 1: Formario Number
stage 2: SegWord
stage 3: Amcas
stage 4: Accumulate Proper-noun Initial Default-cats
```

Figure 3. Example of analysers ordering list.

The manager of MACO has been implemented using a Production Rules approach. This approach was already used within the SEISD [Ageno et al. 92] and TGE [Ageno et al. 93,94] environments and has been mainly motivated by the need of providing a flexible and open way of defining the control. PRE (Production Rules Environment) is a rule-oriented general purpose interpreter [Ageno et al. 93]. It follows the principles of most Production Rules Systems [Brownston 86] but is deeply adapted to Natural Language applications. PRE offers a powerful (according to both expressiveness and performance) rule application mechanism and provides the possibility of defining higher level mechanisms, as rulesets (allowing inheritance capabilities), and the choice between control strategies, either user-defined or provided by the system. In PRE, there is a set of objects placed in an active data storage device (the working memory, WM). MACO operation is defined by a set of production rules that act on the WM. Rules are of type:

```
<pattern-conditions> --> <actions>
```

where <pattern> is matched against the WM, and actions both modify the WM, and call to external lisp-functions.

\(^2\) this limitation doesn’t preclude the use of all analysers on all the words, because there is the possibility of including all the analysers in a unique stage.
We hope that PRE mechanisms will provide a friendly way of enabling the user to tailor the manager (and therefore the control) of MACO to his own needs. For instance, one of the natural extensions to the current manager (not implemented yet) would be to allow for complex rules that would decide to call one or another analyser given the results of previous ones, or the position of the word being analyzed in the corpus (e.g., to consider the word a proper-noun if the initial letter is upper-case and the word is not followed by a punctuation sign, ...). Given the declarativity and expressiveness of PRE it wouldn’t be difficult to incorporate such extensions. Of course, using this meta-level for controlling the MACO performance could be redundant with some of the processors and can introduce, so, an undesirable overhead. Using complex control rules requires, so, a careful analysis of the morphological phenomena to be treated as well as the way the different processors perform this treatment.

We introduce next PRE syntax through a running example. Interested readers can refer to [Ageno et al. 93] for a complete introduction on PRE capabilities. Consider the following example extracted from the manager code:

```
(rule rule-assign-categories
  ruleset process-word
  control one
  priority 2
  (analysers ^all ?all ^pending-set ?pending-set
   ^current-set ?current-set
   ^pending-current (?current *new-pending-current))
  (word ^form ?word ^cats ?previous-cats)
  ->
  (modify 1 ^all ?all ^pending-set ?pending-set
   ^current-set ?current-set
   ^pending-current ?new_pending current)
  (?categories := (assign-categories ?word ?current))
  (?final-cats :=
   (concatenate 'list ?previous-cats ?categories))
  (modify 2 ^form ?word ^cats ?final-cats)
)
```

This rule will be executed only in case the pattern-condition is fulfilled by the objects present in the WM. In this case the condition requires the occurrence of two objects: (1) one named analysers (corresponding to the execution copy of the analysers order list), which should have the ^all, ^pending-set, ... attributes, and whose values will be matched against the patterns ^all, ^pending-set, ..., respectively. And (2), another named word (corresponding to the current word to be analyzed), which should have the ^form, ^cats attributes, and whose values will be matched against the patterns ^word, and ^previous-cats respectively. If the pattern-condition succeeds then the analysers object will be modified in order to take into account that another analyser is being tried on the current word, the current analizer is called on the current word by means of a lisp call (assign-categories ), and the resulting categories are added to the ones obtained before and are finally updated in the word object.

The MACO manager contains up to now 5 rulesets, and 24 rules. In the appendix 2 the code corresponding to the complete set of rules is included. In the following paragraphs we explain briefly the operation of the different rulesets for carrying out the control task.

Apply-stages, the main ruleset, is executed iteratively until all the stages are consumed. At each stage it performs the following algorithm:

(a) It applies the initialise ruleset, which creates some WM objects and, depending on
the type of process, initialises the entry points or resumes the status to the last saved entry point.

(b) It opens the input and output files.

(c) It retrieves the following stage to be applied.

(d) It applies the ruleset *process-words*, which performs an interaction through the input executing the following steps for each word.

   (1) It applies the *process-word* ruleset, which tries the different analysers on the word as stated in the current stage in either a parallel or backing-off strategy depending on the current mode of operation.

   (2) It writes the corresponding results on the output files. Two files are used, one for analyzed words and another for unknown words.

(e) Finally, it closes the input and output files, leaving them ready to be processed through another stage.

4.2. Morphological Processors and Data Sources.

4.2.1. SegWord.

SegWord morphological analyser has been successfully used in SEISD system [Ageno et al. 92] for analysing dictionary definitions (quite restricted as regards derivations). Its main advantage lies on being the only analyser which uses a dictionary as a source of stems, the Vox LDB in our case, as a source for stems, representing therefore a potential of 90,000 entries. Unfortunately, SegWord has proven to be inadequate when morphologically analysing unrestricted corpus in Spanish, because more complex derivations (verb tenses, etc.) occur, and the number of irregular forms is much higher. The main reason for this failure has been the extremely simple architecture of SegWord together with the low expressive capability of the rules syntax. We will firstly describe this system briefly (for a deeper view on SegWord architecture and operation see respectively [Sanfilippo 90a] and [Sanfilippo 90b]) and then we will discuss the problems found in its application.

SegWord’s strategy consists of segmenting words step by step, extracting potential suffixes, prefixes or entire words and trying to apply some rule(s) from the set defined by the user. Thus, there are three types of rules: suffix-rules, prefix-rules and compound-rules. Some examples of each rule type follow (only the first one is explained in detail):

```plaintext
(SUFFIX-RULE RESTRICTIONS final-only CANONICAL-FORM as SURFACE-FORM as BASIC (er ir ar o a e) CATEGORY (V > VPI2) (V > VPS2) (ADJ > ADJFP) (N > NFP))
```

This rule, given a stem categorised in the dictionary as a verb (suffix *ar*, *er* or *ir*), produces two output categories of verb, the first in Indicative-Present, 2nd Singular Person and the second in Subjunctive-Present, 2nd Singular Person; given a stem categorised as an adjective (ending *o*, *a* or *e*) gives an output category of feminine plural adjective, and for a noun stem
(ending o, a or e) gives as a result a category of feminine plural noun.

(SUFFIX-RULE
Restrictions final-only : plurals -Ón
Canonical-form ones
Surface-form ones
Basic Ón
Category (N > NP) (Adj > ADJP))

(SUFFIX-RULE
Restrictions final-only : adverbs
Canonical-form mente
Surface-form mente
Category (Adj > Adv))

(COMPound-RULE V N > N)

(PREFIX-RULE
Canonical-form re
Surface-form re
Category (N > N) (Adj > Adj) (V > V))

Up to now we have developed 257 SegWord rules for Spanish. As stated above, although the interaction with the Vox LDB allows access to a big volume of information, the system has proved too simple to tackle the analysis of unrestricted corpus in Spanish. Some problems cannot be solved, and in order to solve other ones an increase in the number of rules would be necessary which would drop the performance of the system significantly. Various problems found when using SegWord are listed below. We first refer to general problems and then to the more specific due to the characteristics of Spanish morphology.

a.- At a most general level, SegWord's main disadvantage is the impossibility of declaring models of morphological behaviour. This makes this parser specially unsuitable for parsing verbs as long as Spanish shows a large number of paradigms of verbal irregularity.

To give an example, a big group of verbs bearing the vocalic sound -o- on its stem change it to -ue- when inflecting; e.g. mover -> nuevo, mueves... The effect is that when parsing a word as nuevo SegWord will look for an entry as *muever on the lexicon and of course it will not find it. So the parsing fails. Phenomena like these are not isolated exceptions on Spanish verbal morphology, there is a wide range of models of vocalic and consonantal alternations of this kind.

Moreover, the simple segmentation of Spanish verbs on two parts: stem and desinence, falls short for accounting on their complex morphology. It is widely assumed that one can mark out on Spanish verbs four distinct morphemes which combine to give raise to surface forms. These four morphemes are (left to right): (i) the strict stem of the word, (ii) the thematic vocal which marks conjugation paradigm, (iii) an auxiliary morpheme which marks mood and tense, and (iv) a concordance morpheme which marks number and person. (E.g. cant-a-ba-s, cant-a-ba-mos, cant-a-re-mos, sub-i-re-mos...)

Data like these show that one cannot expect recognising verbal forms neither by supposing standard behaviours on stems obtained via segmentation of the lexical entry, nor expecting for a simple regular one-form-agglutinated inflection. Thus, one can not rely on SegWord for parsing many Spanish verbs: a model-based parser is needed to do that.
b.- More specific limitations of SegWord are the following:

b.1.- One can declare only one rule for each surface-form, what unavoidably leads to over-parsing, as long as all phenomena (no matter how distinct they are) which that specific surface-form involves, have to be agglutinated inside that rule. For instance, let us look at the rule dealing with the surface-form -a:

(SUFFIX-RULE
RESTRICTIONS final-only
CANONICAL-FORM a
SURFACE-FORM a
BASIC (er ir ar o e)
CATEGORY (V > VP13) (V > VPS13) (V > VM2) (V > VM3) (N > NF)
(ADJ > ADJF))

It can be observed that this suffix produces feminine forms of nouns and adjectives and also different verbal inflections. Leaving apart the doubtful consistency of having to manage nominal morphology inside the same rule where one deals with verbs, the user is forced to agglutinate in the same rule two quite different verbal derivations: adding -a to 1st conjugation verb (those whose Infinitive ends in -ar) an Indicative-Present Singular 3rd Person (VP13) and an Imperative-Singular 2nd Person (VM2) are obtained; but adding -a to a 2nd or 3rd conjugation verb (-er, -ir) Subjunctive-Present Singular 1st and 3rd (VPS13) and Imperative 3rd (VM3) are formed. The effect here is that every verb of every conjugation is compulsorily going to get both kinds of results -that is, four verbal parsing outputs from which two of them are incorrect. This is just an example, but other unexpected wrong parses often result from that limitation of SegWord.

b.2.- In Spanish many words change their graphical accentuation when inflecting. SegWord can not deal with this, since its only operation consists on extracting an affix form from a string to subsequently add there another affix. Therefore, if the change in the accentuation falls on the invariable part (the stem) the new form is not recognized. E.g. the plural form of volumen is volúmenes. So when SegWord tries to parse volúmenes applying the rule which states that *n noun-stem + e s* gets plural, it will look for *volúmen (not for volume) in the lexicon. It will not find it so the parse will fail.

b.3.- Dealing with some simple lexical phenomena usually leads to an increasing number of rules. For instance, Spanish pronominal verbs (those showing a reflexive meaning) are encoded in the dictionary adding the reflexive pronoun -se to the infinitive (e.g. arrepentirse: ‘to repent (one’s sins)’). If one wants any form derived from this kind of verbs to be parsed, for every usual rule belonging to verbs ending in -ar, -er, -ir, another rule for basic forms -arse, -erse, -irse must be stated.

b.4.- Other problems come from by certain consonantal sounds which lead in Spanish to graphic changes depending on the context, e.g. g(a,o,u) -> gu(e,i) (jugar -> jugue); c(a,o,u) -> qu(e,i); g(e,i) -> j(a,o,u).

b.5.- SegWord can not apply consecutively two or more rules. This is one of its most important limitations. To overcome this problem a big number of rules can be needed even to deal with simple lexical phenomena. For instance one can not easily treat suffix of non-personal verbal forms (e.g. saltando -> saltándome, saltándote; that is gerund + pronoun) as long as the gerund is formed from the infinitive form (saltar -> saltando) and it is not possible to apply a second rule which add the pronoun to the
gerund already obtained. The solution would be again an inefficient display of rules, i.e. positing for every basic form -ar rules parsing surface-form suffixes -ando, -ándome, -ándote, -ándose, etc. This is going to grow worse considering that Spanish may concatenate more than one pronoun to a basic form (-ándo-me-lo, ándo-me-la, -ándo-me-los, ándo-me-las, etc.)

b.6.- Finally, besides the intrinsic disadvantages explained above, SegWord has shown some unexpected drawbacks, as the drop in performance when using prefix rules.

4.2.2 AMCAS.

AMCAS (Analizador Morfológico del Castellano) is a powerful morphological analyzer based on a finite state machine enriched with conditions attached to the arcs. Each state recognizes some part of the Spanish morphology, and each arc between states restricts future steps. The aim on designing AMCAS was to focus more on the complexity of Spanish morphology rather than a broad coverage. In this sense AMCAS can be seen as a complementary tool of SegWord. A detailed description of AMCAS can be found in [Marti 88] (in fact this reference describes a Catalan version of the analyzer but the tool is the same and the linguistic content quite close).

4.2.2.1 Functional description.

The general structure of a word to be analyzed is:

\[
\text{[prefix]} \text{stem} \text{[derivative suffixes]* [flexive suffix]}
\]

So far AMCAS can only analyze words composed by a stem and any number of suffixes (prefixed stems are considered new stems). The space is not considered as a separator and thus lexemes represented as multiwords can be recognized. AMCAS provides the morphological category as well as other morphological information in an incremental way. The different pieces that constitute the whole word contribute to the output information.

As stated previously the internal structure of the words can be expressed in terms of a finite state automaton. AMCAS performs like a non-deterministic recognizer of the language generated by the automaton. During the recognition process the analyzer will associate properties (in form of feature-value pairs) to the fragment of word it recognizes. Once a word has been properly analyzed, the list of properties corresponding to each correct analysis is returned.

Figure 4 shows a simplified version of the current automaton. The initial state is \textit{START} and the final state \textit{F}. The rough meaning of the different states is the following: \textit{RNA} is the state recognizing nouns and adjectives. \textit{RNA1} is an equivalent state coming from the verbal derivation. \textit{RVI} is the state recognizing verbs. In \textit{RV} the verbal flexion has been incorporated. \textit{INFIJ} is a state for recognizing infixes. \textit{RGN} is the state for recognizing the gender and number flexion.

The first step is to recognize the stem. From then on, the task is to find the successive suffixes until the word is completely consumed. Then if the state is a final one the analysis is successful. AMCAS returns all the successful analyses found.
4.2.2.1.1 Data structures involved.

All the examples in this section are taken from an implementation of AMCAS used in the GUAI System [Rodríguez, 89]. As pointed out in fig 5, linguistic knowledge is organised in six data structures in AMCAS:

- **stems dictionary**: Contains the different stems that correspond to the initial fragments of valid words. Each stem is typified as belonging to a specific model and owns its specific properties. A default inheritance mechanism is used allowing the distribution of information between the stem and the model. Of course several different interpretations can be attached to each stem.

- **suffixes dictionary**: Contains in a similar way the possible suffixes. Each suffix belongs to a model and owns its specific properties.

- **stem models**: Allows the grouping of different stems in classes tied to similar morphological patterns.

- **suffix models**: Allows the grouping of different suffixes in classes tied to similar morphological patterns.

- **states**: Describes the nodes of the automaton. Different nodes can be tagged as initial or final ones. Currently, only one state (START) is tagged as initial and only one state (F) is tagged as final but other organisations can be stated without problems.

- **rules**: Describes the valid transitions between states and the conditions attached to each transition.
A central issue in the organisation of AMCAS is the use of models for grouping both stems and suffixes. The fact that each stem and suffix must be attached to one or more models allows a straightforward way of expressing conditions attached to the transitions between states. Other advantages of such approach come from the fact that common information can be attached to models and not to individual stems with the obvious saving of storage and acquisition effort.

We will give next a short description of the contents of these structures:

- **Stem dictionary**: Each entry consists of a pair formed by the headword and the attached information. The former is a string corresponding to the stem and the latter a list with the following information:
  
  - **model**
  - **divisibility**: Is a flag that indicates whether during the analysis the fragment can be split in simpler components.
  - **property-list**: Contains a possibly empty list of properties (sometimes including the category). Each property is described in terms of a feature and a value or list of values.

Figure 6 depicts a fragment of the stem dictionary.

- **Suffixes dictionary**: Each entry consists of a pair composed by the headword and the attached information. The former is a string corresponding to the suffix and the latter is a list with the following information:
  
  - **model**
  - **divisibility**: Is a flag that indicates if during the analysis the fragment can be
split in simpler components.

- **property-list**: Contains a possibly empty list of properties (sometimes including the category)

### STEM DICTIONARY (FRAGMENT)

<table>
<thead>
<tr>
<th>STEM</th>
<th>MODEL</th>
<th>PROPERTIES</th>
<th>DIVISIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;d&quot;</td>
<td>D</td>
<td>(&quot;B1&quot; &quot;DORW&quot;&quot;TVM&quot; &quot;VI&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;de&quot;</td>
<td>PREP</td>
<td>0</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;del&quot;</td>
<td>PREP</td>
<td>0</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;deposit&quot;</td>
<td>AM</td>
<td>(&quot;TGN&quot; &quot;OM&quot;) (&quot;BL&quot; &quot;&amp;3&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;dese&quot;</td>
<td>AM</td>
<td>(&quot;TGN&quot; &quot;OM&quot;) (&quot;B1&quot; &quot;OSO&quot;) (&quot;TVM&quot; &quot;VI&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;dich&quot;</td>
<td>HECI</td>
<td>(&quot;TGN&quot; &quot;OM&quot;) (&quot;CONJ&quot; &quot;3&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;dich&quot;</td>
<td>DETN</td>
<td>(&quot;DET&quot; &quot;DEM&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;dich&quot;</td>
<td>PRON</td>
<td>(&quot;PRN&quot; &quot;DEM&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;diner&quot;</td>
<td>NOM</td>
<td>(&quot;B1&quot; &quot;DAF&quot;) (&quot;B2&quot; &quot;ALM&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;director&quot;</td>
<td>NOM</td>
<td>()</td>
<td>nil</td>
</tr>
</tbody>
</table>

Figure 6

### SUFFIXES DICTIONARY (FRAGMENT)

<table>
<thead>
<tr>
<th>STEM</th>
<th>MODEL</th>
<th>PROPERTIES</th>
<th>DIVISIBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;a&quot;</td>
<td>AASAM</td>
<td>(&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>AASFEM</td>
<td>(&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>GAF</td>
<td>(&quot;GEN&quot; &quot;FEM&quot;) (&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>GAM</td>
<td>(&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>GBAJ</td>
<td>(&quot;GEN&quot; &quot;FEM&quot;) (&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>GBAW</td>
<td>(&quot;GEN&quot; &quot;FEM&quot;) (&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>GN1</td>
<td>(&quot;GEN&quot; &quot;FEM&quot;) (&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>GOAJ</td>
<td>(&quot;GEN&quot; &quot;FEM&quot;) (&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>IMP</td>
<td>(&quot;NUM&quot; &quot;SG&quot;) (&quot;PERS&quot; &quot;2&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>IPO</td>
<td>(&quot;NUM&quot; &quot;SG&quot;) (&quot;PERS&quot; &quot;3&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>SP2</td>
<td>(&quot;NUM&quot; &quot;SG&quot;) (&quot;PERS&quot; &quot;1/3&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;aba&quot;</td>
<td>IMA</td>
<td>(&quot;PERS&quot; &quot;1&quot;) (&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>...</td>
<td>PROE</td>
<td>(&quot;ENCL&quot; &quot;LO&quot;) (&quot;BL&quot; &quot;&amp;3&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>...</td>
<td>PROE</td>
<td>(&quot;BL&quot; &quot;&amp;3&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;&amp;3&quot;</td>
<td>GBF&amp;3</td>
<td>(&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
<tr>
<td>&quot;&amp;3&quot;</td>
<td>GBM&amp;13</td>
<td>(&quot;NUM&quot; &quot;SG&quot;)</td>
<td>nil</td>
</tr>
</tbody>
</table>

Figure 7

16
Figure 7 depicts a fragment of the dictionary.

• **Stem and suffix models:** These structures group respectively stems and suffixes that exhibit the same behavior for the derivational and flexive phenomena. Each entry consists of a pair formed by the headword and the attached information. The former is a symbol identifying the model and the latter a list of properties following the same format than in the case of dictionaries. Figures 8 and 9 show a fragment of such structures.

### STEM MODELS (FRAGMENT)

<table>
<thead>
<tr>
<th>MODEL</th>
<th>PROPERTIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSS</td>
<td>((&quot;CAT&quot; &quot;CONJ&quot;) (&quot;TCON&quot; &quot;CSS&quot;) (&quot;BL&quot; &quot;SI&quot;))</td>
</tr>
<tr>
<td>D</td>
<td>((&quot;CAT&quot; &quot;VERB&quot;) (&quot;TV&quot; &quot;D&quot;))</td>
</tr>
<tr>
<td>DETDI</td>
<td>((&quot;CAT&quot; &quot;DET&quot;) (&quot;PERS&quot; &quot;1&quot;) (&quot;TGN&quot; &quot;EAO&quot;) (&quot;BL&quot; &quot;SI&quot;) (&quot;DET&quot; &quot;DEM&quot;))</td>
</tr>
</tbody>
</table>

Figure 8

### SUFFIX MODELS (FRAGMENT)

<table>
<thead>
<tr>
<th>MODEL</th>
<th>PROPERTIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>AASAM</td>
<td>((&quot;GEN&quot; &quot;AMBI&quot;))</td>
</tr>
<tr>
<td>AASFEM</td>
<td>((&quot;GEN&quot; &quot;FEM&quot;) (&quot;CAT&quot; &quot;ADJ&quot;))</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>GAF</td>
<td>((&quot;CAT&quot; &quot;NOM&quot;) (&quot;GEN&quot; &quot;FEM&quot;))</td>
</tr>
<tr>
<td>GAM</td>
<td>((&quot;CAT&quot; &quot;NOM&quot;) (&quot;GEN&quot; &quot;MASC&quot;))</td>
</tr>
<tr>
<td>GBAJ</td>
<td>((&quot;CAT&quot; &quot;ADJ&quot;))</td>
</tr>
<tr>
<td>GBAW</td>
<td>((&quot;CAT&quot; &quot;ADJ&quot;))</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>GN1</td>
<td>0</td>
</tr>
<tr>
<td>GOAJ</td>
<td>((&quot;CAT&quot; &quot;ADJ&quot;))</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>IMP</td>
<td>((&quot;CAT&quot; &quot;VERB&quot;) (&quot;TEMP&quot; &quot;PRES&quot;) (&quot;PROE&quot; &quot;SI&quot;) (&quot;MODE&quot; &quot;IMP&quot;))</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>IPO</td>
<td>((&quot;CAT&quot; &quot;VERB&quot;) (&quot;TEMP&quot; &quot;PRES&quot;) (&quot;BL&quot; &quot;SI&quot;) (&quot;MODE&quot; &quot;IND&quot;))</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>PROE</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>SP2</td>
<td>((&quot;CAT&quot; &quot;VERB&quot;) (&quot;TEMP&quot; &quot;PRES&quot;) (&quot;MODE&quot; &quot;SUBJ&quot;))</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>&amp;3</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 9
- **Rules**: This structure permits to express the allowed transitions between states and the conditions attached to each transition. Each rule contains the following information:

  - **Initial state**: Indicates the origin node in the transition
  - **Final state**: Indicates the destination node in the transition
  - **Model**: Indicates the model to which the stem or suffix to be consumed must belong in order to allow the transition.
  - **Conditions**: Indicates the set of conditions that must be satisfied to allow the transition. The set of conditions is implicitly considered as a disjunction of conditions. That means that at least one of the conditions must be satisfied in order to allow the transition. The satisfaction of a condition means that the properties of the current entry will contain a property (feature, value) unifiable with such condition.

Figure 10 depicts a fragment of the set of rules.

<table>
<thead>
<tr>
<th>INITIAL STATE</th>
<th>FINAL STATE</th>
<th>MODEL</th>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR</td>
<td>RGN</td>
<td>AS</td>
<td>(&quot;TGN&quot; &quot;OAJ&quot;)</td>
</tr>
<tr>
<td>RV</td>
<td>F</td>
<td>EL</td>
<td>(&quot;BL&quot; &quot;SI&quot;)</td>
</tr>
<tr>
<td>RV</td>
<td>F</td>
<td>&amp;3</td>
<td>(&quot;BL&quot; &quot;&amp;3&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>RV</td>
<td>PROE</td>
<td>(&quot;PROE&quot; &quot;SI&quot;)</td>
</tr>
<tr>
<td>RV1</td>
<td>INFIJ</td>
<td>CC</td>
<td>(&quot;U&quot; &quot;CCVD&quot;)</td>
</tr>
<tr>
<td>RV1</td>
<td>RGN</td>
<td>AASFEM</td>
<td>(&quot;TI&quot; &quot;OAJ&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>RV1</td>
<td>RGN</td>
<td>(&quot;TI&quot; &quot;AF&quot;) (&quot;TGN&quot; &quot;AF&quot;)</td>
</tr>
<tr>
<td>RV1</td>
<td>RGN</td>
<td>GAF</td>
<td>(&quot;TGN&quot; &quot;I&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>RV1</td>
<td>GAF</td>
<td>(&quot;TGN&quot; &quot;AF&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>RV1</td>
<td>GOAJ</td>
<td>(&quot;B1&quot; &quot;TOJ&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>RV1</td>
<td>IMP</td>
<td>(&quot;TV&quot; &quot;R&quot;) (&quot;TV&quot; &quot;ACUE&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>RV1</td>
<td>IPO</td>
<td>(&quot;TV&quot; &quot;R&quot;) (&quot;TV&quot; &quot;ACUE&quot;)</td>
</tr>
<tr>
<td>...</td>
<td>RV1</td>
<td>SPB</td>
<td>(&quot;TV&quot; &quot;PONG&quot;)</td>
</tr>
<tr>
<td>START</td>
<td>RV1</td>
<td>D</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 10
4.2.2.1.2 Performance of the analyser.

At a high abstract level the analysis algorithm is the following:

- find all matching stems
- for every valid stem do
  - inherit properties from model
  - while a final state is not reached and a valid continuation is possible do
    - advance one state if possible
    - find all matching suffixes and inherit attributes from models

A transition can be performed (consuming a fragment of the word being analysed) if the current set of properties match any of the conditions of some rule going from the former state with the current word model. The idea is that suffixes change the expected morphology of the word and give more clues about its future morphology. The properties from suffixes have precedence over previous properties of the word allowing, thus, a kind of overwriting; thus when a suffix is found, its properties are added, the properties of its model are inherited by default, and the resulting set is appended to the current set. If the pending fragment is the empty word when getting to the final state, the analysis is successful.

It could be helpful to introduce an example in detail. Consider the word "dámeolo" (give it to me). The word is at a preprocessing step converted into "dameolo&3". We have adopted in AMCAS the decision of shifting the accents to the end of words indicating also the position (syllable) where the accent was placed. The aim is to ease the expression of the morphological behaviour of the words (in Spanish many verbal forms, for instance, present accent alterations as has been described previously).

The process starts by looking up in the stem dictionary. In this case only the entry "d" is selected. The rest of the fragment "ameolo&3" remains pending. In Figure 6 we can see that the model associated to "d" is D, while the list of properties include (("BL" "DORW") ("TVM" "VI")). From Figure 8 we can obtain the list of properties of model D. We incorporate, then, such properties to the properties currently attached to the fragment obtaining finally the following property list:

\[
(("BL""DORW")("TVM""VI")("CAT" "VERB")("TV" "D")).
\]

We can see by now that the category of the stem is VERB and that the type of verb is D.

From Figure 10 we can see that the only valid transition from the initial state START through the model D reaches the state RV1. No conditions are attached to this transition and thus the transition can be carried out.

In this stage, the current state is RV1, the pending string is "ameolo&3" and the current list of properties is depicted above.

Now we look for a new fragment, in this case in the suffixes dictionary. Only the suffix "a" is found. The rest of the fragment "meolo&3" remains pending. We can see in Figure 7 that there exist a lot of different interpretations corresponding to "a". We must deal with all these interpretations in turn. We must incorporate, then, all the properties owned by each suffix (and its corresponding model) to the current list of properties and look for further transitions, starting with state RV1. We can see in Figure 10 that AASFEM, GAF, GAM and G0A1 allow transitions to RGN though none of them succeed in applying the conditions. IMP, IPO and SP2 allow transitions to RV and further on, the former and latter of the models succeed in
applying the conditions.

At this point the current state is RV, the pending fragment is "me\&3" and we have two possible set of properties:

\[
\begin{align*}
&\{("B1" "DORW")("TVM" "VI")("CAT" "VERB") \\
&("TV" "D")("TEMP" "PRES") ("PROE" "SI")("MODO" "IMP") \\
&("NUM" "SG")("PERS" "2")\}
\end{align*}
\]

corresponding to the model IMP and

\[
\begin{align*}
&\{("B1" "DORW")("TVM" "VI")("CAT" "VERB") \\
&("TV" "D")("NUM" "SG")("PERS" "1/3")("TEMP" "PRES") \\
&("MODO" "SUBJ")\}
\end{align*}
\]

corresponding to the model SP2.

The following fragment, "me", appears once in the suffixes dictionary (Figure 7). The pending fragment is by now reduced to "lo\&3". The model corresponding to "me" is PROE. The model does not incorporate any property while the suffix incorporates ("BL" "&3"). From RV only one transition is allowed for the model PROE. The transition leads to the same state (RV) but a condition ("PROE" "SI") must be satisfied. From the two active possibilities only the first one remains.

The situation is now the following: The current state is RV, the pending fragment is "lo\&3" and the current properties are:

\[
\begin{align*}
&\{("B1" "DORW")("TVM" "VI")("CAT" "VERB") \\
&("TV" "D")("TEMP" "PRES") ("PROE" "SI")("MODO" "IMP") \\
&("NUM" "SG")("PERS" "2")("BL" "&1")\}
\end{align*}
\]

The next fragment ("b") is also unique. The remaining string is now "&3". The model to which "lo" belongs is PROE too. The suffix will incorporate the properties ("ENCL" "LO")("BL" "&3"). The allowed transition is the same that in the prior case. The current property list is now:

\[
\begin{align*}
&\{("B1" "DORW")("TVM" "VI")("SEM" "DECIR-1")("CAT" "VERB") \\
&("TV" "D")("TEMP" "PRES") ("PROE" "SI")("MODO" "IMP") \\
&("NUM" "SG")("PERS" "2")("BL" "&3")("ENCL" "LO")\}
\end{align*}
\]

The last fragment "&3", belonging to the model \&3, permits the final transition from state RV to the final state F. No new properties are incorporated. As the current state is a final one and no pending fragment longer exists the word is considered to be correct and the corresponding property list will be returned as the final result (of course not all the returned properties are of interest, some of them are for internal use of the analyser, the MACO unifier (see 4.3), will be in charge of filtering this set of properties).
4.2.2.2 Implementation.

The current release of AMCAS, included in MACO, is written in Common Lisp and uses some LDB facilities.

The sizes of the different data structures involved in AMCAS are currently the following:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stem dictionary</td>
<td>6598 entries</td>
</tr>
<tr>
<td>Suffixes dictionary</td>
<td>329 entries</td>
</tr>
<tr>
<td>Stem models</td>
<td>166</td>
</tr>
<tr>
<td>Suffix models</td>
<td>329</td>
</tr>
<tr>
<td>Rules</td>
<td>544</td>
</tr>
</tbody>
</table>

All these structures can be updated by means of acquisition programs as described in section 5.3. The last four data structures are stored in Ascii Files and loaded into hash tables at running time to speed up search time. The sizes are moderate and the contents are relatively stable. As the size of the stem dictionary is too huge, stem entries have been indexed via LDB, which makes problematic the updating. To overcome this disadvantage, a hash table of stem overflow has been built, and whenever the size requires it, stems are reindexed.

4.2.3 Other Knowledge Sources.

Apart from the described analysers, MACO uses other knowledge sources to analyse words.

- **Formario**: A list of analysed word forms, integrated and indexed into a Lexical Database (LDB). MACO can consult it and retrieve an analysis for a given word.

- **Proper Nouns List**: A list of 180,000 proper nouns where a word can be looked up (using binary search) in order to determine whether it is a proper noun. Besides, a word starting with a capital letter and not recognised by any other analyser, is supposed to be a proper noun, although it is not added to the Proper Nouns List.

- **Accumulate File**: The file that stores the results of prompting the user for the analysis of an unrecognised word (see 5.4, Unrecognised words). The file can be consulted to avoid missing the analysis of a word twice. Periodically, the contents of this file are integrated into the Formario.

- **Number recognizer**: A very simple number recognizer, which given a string is able to determine whether it is a number. It consists of a swallow analysis (not even a regular expression of a number) which determines whether all the characters in the string are digits, a sign, a decimal point, or an exponential symbol.

---

3This list has been obtained merging different publicly available proper names lists, obtained mainly from three sources: (1) the Consortium for Lexical Research, New Mexico State University (anonymous ftp to clrl.nmsu.edu) (2) the Cobuild “Bank of English” developed by Collins and the University of Birmingham (anonymous ftp to titania.cobuild.collins.co.uk/pub/proper_names), and finally (3) the WordNet 1.3 lexicon developed at the Cognitive Science Laboratory in Princeton University (anonymous ftp to clarity.Princeton.edu/pub).
4.3. Categorisation.

Each analyser employed in MACO system uses a particular format for categories, so it is necessary to perform a unification step once each entry has been assigned one or more categories by any of these analysers.

Our SgWord categories format has already been seen (although not fully described) in section 4.2.1. This format is very simple: each syntactic category is described by means of the initial single letter (or set of letters), and there may be some additional ones, depending on the main category, indicating other properties, such as gender, number or verb tenses. There are 15 main categories.

AMCAS categories format is quite more complex and quite more complete, as this analyser is directly oriented to Spanish. Each individual category is defined by means of a list of pairs attribute-value, so that depending on the category some attributes may not have any value. The attribute discriminating the main category is the cat attribute, for which there are 13 possible values. A total of 35 attributes are defined, where the most numerous attribute may take one out of 162 possible values.

The output set of categories is the one used by Formario (see 4.2.3), which has been considered the one able to keep the highest quantity of information collected by any of the other different possible formats of categories. Thus, the strategy when translating from a particular category set to the unified output set will be trying to keep the maximum deal of information, and only when certain information cannot be filled in, using the symbol "*" to substitute it. Obviously, the code for the category must always be present. The set of output categories is defined next (a rough equivalence between our codes and Penn Treebank ones is included in appendix 1).

The codes for each lexical entry are formed by a main syntactic category code followed by a variable sequence of attribute codes depending on the particular category. Each category is defined following the format: Category_Code, Category_Identifier, Attribute1, Attribute2, Attribute3... The current list of categories is the following:

```
S,SUSTANTIVO,A,B,P    (noun)
A,ADJETIVO,A,B,C,D,P   (adjective)
P,PRONOMBRE,A,B,E,F,I,P (pronoun)
V,VERBO,H,G,I,J,B,P,A  (verb)
Y,HAY                  (existential there)
-E,SER,H,G,I,J,B,P     (to be)
D,ADVERBIO,K           (adverb)
C,CONJUNCION,N         (conjunction)
W,NOMBRE_PRO           (proper noun)
J,ARTICULO,A,B,O,P     (determiner)
N,NUMERO               (number)
Z,PUNTUACION,Q         (punctuation sign)
H,HABER,H,G,I,J,B,P    (auxiliary have)
R,PREPOSICION,M,L,A,B   (preposition)
```

Each attribute is defined following the format: Attribute_Code, Subcategory, Agreement, Attribute_Name, Value1_Code, Value1_Name...(where Agreement equal to 1 means that this attribute keeps agreement).

The definition of the codes for each attribute is described next. Any attribute may in turn be a
subcategory. This can be specified by means of the second element of the attribute definition. In case this integer takes value 1, the attribute is a subcategory (0 in the other case). Whenever a category encloses a subcategory attribute, this means that a more refined categorisation will be performed. Two words with the same category and different values for that attribute are considered as different categories. Taking into account subcategorisation, the total number of categories is 40. These attributes should not get the value ““ when translation from another format is performed, so this fact must be considered by the unification program.

The current list of attributes follows:

A, 0, 1, GENERO, 1, MASCULINO, 2, FEMENINO, 3, COMUN, 0,
SIN_GENERO
(gender: masculine, feminine, common, no_gender)

B, 0, 1, NUMERO, -, SINGULAR, +, PLURAL, 3, COMUN, 0, SIN_NUMERO
(number: singular, plural, common, no_number)

C, 0, 0, TIP_ADJ_PI, 2, POSESIVO, 3, DEMOSTRATI, 4, RELATIVO, 7,
INDEFINIDO, 8, INTERROGAT, 9, NUMERAL, 0, ALTERADO
(adjective_class: possessive, demonstrative, relative, indefinite, interrogative, numeral, modified)

D, 1, 0, TIP_ADJ_MA, 0, ATRIBUTIVO, 1, DETERMINAN
(adjective_class: attributive, determinative)

E, 0, 0, TIP_PRO_PI, 2, POSESIVO, 3, DEMOSTRATI, 4, RELATIVO, 7,
INDEFINIDO, 8, INTERROGAC, 9, PERSONAL
(pronoun_class: possessive, demonstrative, relative, indefinite, interrogative, personal)

F, 1, 0, TIP_PRO_MA, 1, NOMINATIVO, 4, RELATIVO, 5, OTROS
(pronoun_class: nominative, relative, others)

G, 0, 0, MODO, 1, INDICATIVO, 2, IMPERATIVO, 3, SUBJUNTIVO, 4,
CONDICIONA, 0, SIN_MODO
(mode: indicative, imperative, subjunctive, conditional, no_mode)

H, 1, 0, FORMA_VERB, 0, PERSONAL, 1, PARTICIPIO, 2, INFINITIVO, 3,
GERUNDIO
(verbal_form: tensed, participle, infinitive, gerund)

I, 0, 0, TIEMPO, 1, PRESENTE, 2, IMPERF_RA, 3, IMPERF_SE, 4,
INDEFINIDO, 5, FUTURO, 0, SIN_TIEMPO
(tense: present, past_1, past_2, past_3, future, no_tense)

J, 0, 1, PERSONA, 1, PRIMERA, 2, SEGUNDA, 3, TERCERA, 0,
SIN_PERSON
(person: first, second, third, no_person)

K, 0, 0, TIP ADV_PI, 0, MODO, 1, LUGAR, 2, DUDA, 3, CANTIDAD, 4, AFIRM
ACION, 5, NEGACION, 6, TIEMPO, 7, CUALIDAD, 8, INTERROGAC
(adverb_class: mode, place, doubt, quantity, affirmation, negation, time, quality, interrogative)

L, 0, 0, TIP_PRE_PI, 0, FINAL, 1, NEGATIVA, 2, CAUSAL, 3, LUGAR, 4,
5. Information acquisition.

5.1. Control Information

MACO may be considered as a toolbox. Therefore, it allows the user to select many operation parameters. In order to define this working options the user can modify the following data, many of which are provided with a default value:

File names: name and path of the input file. The output file names are obtained by appending the strings ".results" (for the successfully analyzed words) and ".aux" (for the unknown words) to the input file name.

Analysers ordering list: it is a list of stages, each of which consists of a list of analysers.

Operation mode: It defines the strategy to follow on applying the analysers at each stage and may be all (parallel), or one-by-one (backing-off). By default one-by-one.

Default categories: the categories to be assigned by the Default-cats analyser. By default it corresponds to some of the open-class categories (i.e., verb, noun and adjective).

Proper noun categories: the categories to be assigned by the Proper-noun analyser.

Process type: it defines whether MACO should start over on processing the input file (start), or should resume the status to the last saved entry point, and continue from there (continue). By default start.

Buffer size: it defines the interval (in number of processed words) between updations of the entry points (by default 20).
Once the user has defined the default working options, the rules that implement the MACO manager can be loaded using the preload-rulesets menu-option. This process checks the syntax of the rules (giving messages when an error occurs) and keeps them in a precompiled form for the PRE interpreter.

5.2. SegWord.

Unfortunately, no facilities at all are provided for knowledge acquisition in this analyser. Rules edition can be performed by means of any editor (no special means for entering rules are included), and the rules file must be initially loaded. Moreover, once a certain set of rules is loaded, illegal formats are ignored, so that although only legal formats are used by the analyser in order to perform recognition of derived words, the user is not warned about illegal ones, making more complex the task of rule debugging.

![Figure 11](image)

5.3. AMCAS information.

The tool for AMCAS maintenance has been developed to keep coherence. As previously said, entries structure must be followed, and some references need integrity checking.

Tools for maintenance of different knowledgetypes are provided: suffixes, stems, suffix models, stem models, and rules. They may be easily modified, created, deleted or consulted
through a friendly interface. In what follows we present the suffix maintenance tool. The other ones are alike.

When executing the tool, a main window (see Figure 11) showing all previously defined suffixes appears. There are five buttons to allow the user to choose among different tasks: *alta* (add), *baixa* (delete), *consulta* (consult), *modificacio* (modify) and *acabar* (exit).

When creating a new suffix (*alta*), the user is prompted to introduce the name. Then multiple model entries for this suffix can be chosen from a list (Figure 12). The suffix is displayed on the upper right hand side of the dialog. Under it there is the list of default properties for the chosen model, and specific properties may be written as a LISP list into the box named *propietats*. The first item of the list is a flag that indicates if this suffix can be split into smaller suffixes. The rest is a list of (property, value) pairs. Each entry is introduced by pressing *afegir* (add).

![Figure 12](image)

When the user pushes *consultar* (consult), a screen shows every entry defined for the selected suffix. (Figure 13).
When the user pushes *modificar* (modify), a screen allows the user to modify previously defined entries, delete wrong entries or add new entries for this suffix string (Figure 14).

### 5.4 Unrecognised words

When MACO is not able to find an interpretation for a word in a text, it stores it as “unknown”. On user’s convenience, he can ask the system to manually analyse the “unknown” words.

Upon this, the system presents a list of “pending” words, and the user chooses one of them. For the chosen word, a dialog (see 5.4.1) is opened, enabling the user to select the category or categories for that word.

Once the user has entered the analyses for a word, the word is stored in the Accumulate file, to enable the system to analyse its further occurrences.

A management dialog is provided for this file, allowing to add, delete, read or modify an entry, consisting of a word and its analysis list. (see 5.4.2)

The categorisations and their attributes are completely configurable. This is done through two configuration files: the CATEGORIES file and the ATTRIBUTES file, that enable the user to define which categories will be used, which attributes will have each of them, and which are the possible values for each attribute. (see 5.4.3)
5.4.1 The Categorisation Dialog

A new dialog (Figure 15) is presented in which the user can select the categorisation of the word, as well as the subcategorisation or attributes (if any are possible) required for the selected category. (WARNING: If the user doesn’t supply them, default values, which may not be appropriate, will be taken for the subcategories and attributes).

When the user is satisfied with his choices, he must press the “Add” button to add the analysis to the analysis list for the word. If the word accepts several analyses, this process may be repeated as many times as necessary.

When all the analyses have been added to the list, pressing the “Ok” button will close the dialog.

Pressing the “Cancel” button will have the effect of closing the dialog and ignoring any additions to the analysis list that might have been made.
5.4.2 The Accumulate File Management Dialog

A list containing the words in the Accumulate File is presented to the user, who may select one, and press the button corresponding to the action he wants to do: Read, Delete, Modify, Add (although for the later no word selection is necessary). Figure 16 shows an execution of the dialog.

An analysis list consists of a LISP-like list in which the first element is a category and the others element are the values of its attributes, in the same order that those attributes are defined in the CATEGORIES file.

**Read:** Opens a dialog in which the user can see the chosen word and its analysis list.

**Delete:** Displays the word and its analysis list, and asks the user to confirm the deletion.

**Add:** Gets a new word from the user, and opens the Categorisation Dialog (5.4.1) to obtain the analysis list.

**Modify:** Opens a dialog with the analysis list of the selected word. The user can choose any of the analyses in the list, and press one of the buttons: Delete, Modify, or Add (although no selection is necessary for the later). Modifications and additions are done editing the analysis list. Figure 17 displays an execution of the modify dialog.
5.4.3 The Configuration Files

There are two main configuration files: On the one hand, the CATEGORIES file, which describes what are the possible categories for a word, and what attributes has each one of them. On the other hand, the ATTRIBUTES file, which describes what are the possible values for each attribute.

**CATEGORIES File**: A LISP-like list in which each element is a list describing a category and its attributes. For each category there is a pair (category code, category name) followed by the codes of all the attributes for the category. Figure 18 shows an example.

**ATTRIBUTES File**: A LISP-like list in which each element is a list describing an attribute and its possible values. For each attribute there is a list of pairs. The first pair is (attribute code, attribute name) and a the rest of the list are pairs (attribute value code, attribute value name). Figure 19 shows an example of the format and content.

6. Performance of MACO.

Several experiments have been carried out in order to test MACO's performance, using different strategies and combinations of knowledge sources.
Results are shown in Figure 20. Performance has been evaluated in several cases: (A) using only one of the available analysers. (B) using all analysers in backing-off, in different orders. (C) using all analysers in parallel for each word.

(((S SUSTANTIVO) A B)
((A ADJETIVO) A B C D)
((P PRONOMBRE) A B E F J)
((V VERBO) H G I J E A)
((Y HAY) )
((E SER) H G I J B)
((D ADVERBIO) K)
((C CONJUNCION) N)
((W NOMBRE_PRO) )
((O ARTICULO) A B O)
((N NUMERO) )
((Z PUNTUACION) Q)
((H HABER) H G I J B)
((P PREPOSICION) M L A B)
((L CONSTRUCCI))

Figure 18. Example of a CATEGORIES File
((A GENERO) (1 MASCULINO) (2 FEMENINO) (3 COMUN) (0 SIN_GENERO))
((B NUMERO) (- SINGULAR) (+ PLURAL) (3 COMUN) (0 SIN_NUMERO))
((C TIP_ADJ_PI) (2 POSESSIVO) (3 DEMOSTRATI) (4 RELATIVO) (7 INDEFINIDO)
(8 INTERROGAT) (9 NUMERAL) (0 ALTERADO))
((D TIP_ADJ_MA) (0 ATRIBUTIVO) (1 DETERMINAN))
((E TIP_PRO_PI) (2 POSESSIVO) (3 DEMOSTRATI) (4 RELATIVO) (7 INDEFINIDO)
(8 INTERROGAC) (9 PERSONAL))
((F TIP_PRO_MA) (1 NOMINATIVO) (2 CLITICO) (3 POSESSIVO) (4 RELATIVO)
(5 OTROS))
((G MODO) (1 INDICATIVO) (2 IMPERATIVO) (3 SUBJUNTIVO) (4 CONDICIONA)
(0 SIN_MODO))
((H FORMA_VERB) (0 PERSONAL) (1 PARTICIPIO) (2 INFINITIVO) (3 GERUNDIO))
((I TIEMPO) (1 PRESENTE) (2 IMPERF_RA) (3 IMPERF_SE) (4 INDEFINIDO)
(5 FUTURO) (0 SIN_TIEMPO))
((J PERSONA) (1 PRIMERA) (2 SEGUNDA) (3 TERCERA) (0 SIN_PERSONA))
((K TIP_ADV_PI) (0 MODO) (1 LUGAR) (2 DUDA) (3 CANTIDAD) (4 AFIRMACION)
(5 NEGACION) (6 TIEMPO) (7 CUALIDAD) (8 INTERROGAC))
((L TIP_PRE_PI) (0 FINAL) (1 NEGATIVA) (2 CAUSAL) (3 LUGAR) (4 DIRECCION)
(5 APROXIMACI) (6 RECIPROCID) (7 TIEMPO) (8 ARTICULADA))
((M TIP_PRE_MA) (0 PREPOSICIO) (1 ARTICULO))
((N TIP_CON_PI) (0 FINAL) (1 CONDICIONA) (2 MODAL) (3 CAUSAL)
(4 CONCESIVA) (5 TEMPORAL) (6 CONSECUTIV) (7 EXPLICATIV)
(8 ADVERSATIV))
((O TIP_ART_PI) (0 DETERMINAD) (1 INDETERMIN))
((Q SIGNOS_PUN) (1 TODOS_SIG) (2 COMA)))

Figure 19. Example of an ATTRIBUTES File

We used two different ratios to measure performance, namely recall and precision. Recall measures the analysis generation capacity of the analyser, that is, how many of the possible correct analysis are generated by the analyser. Precision measures the overgeneration (more accurately, the non-overgeneration) of the analyser, that is, how many of the generated analysis were correct.

We used two kinds of computation for precision and recall. The precision and recall labelled (1) are computed weighting the influence of each word with its frequency. The ones labelled (2) do not take word frequencies into account.

Recall and precision were computed in the following way:

- A sample of 300 words from a corpus was taken, and repetitions filtered, keeping the number of occurrences for each word. This yield 169 different words to be analysed.

- All correct analyses were manually produced for each of the 169 words.

- For each word, recall is computed as the number of generated correct analyses over the total number of correct analyses, and precision is computed as the number of generated correct analyses over the total number of generated analyses. Total precision and recall are the division of the sums of each term over all words (weighted with word

---

4 This corpus was composed of the whole novel “El Sur” by Adelaida García Morales, 17,044 word occurrences, 3,347 different word forms.
frequency in table (1))

- Only category and subcategory are considered to decide whether an analysis is correct.

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>FORMARIO (FO)</td>
<td>96.21%</td>
<td>77.21%</td>
</tr>
<tr>
<td>AMCAS (AM)</td>
<td>94.81%</td>
<td>66.83%</td>
</tr>
<tr>
<td>SEGWORD (SE)</td>
<td>83.59%</td>
<td>82.12%</td>
</tr>
<tr>
<td>Prop. Noun (PN)</td>
<td>26.67%</td>
<td>13.33%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AM-SE-FO-PN</td>
<td>94.55%</td>
<td>71.69%</td>
</tr>
<tr>
<td>SE-AM-FO-PN</td>
<td>84.21%</td>
<td>82.64%</td>
</tr>
<tr>
<td>FO-SE-AM-PN</td>
<td>95.57%</td>
<td>80.37%</td>
</tr>
<tr>
<td>FO-AM-SE-PN</td>
<td>96.69%</td>
<td>78.51%</td>
</tr>
<tr>
<td>PN-SE-AM-FO</td>
<td>82.80%</td>
<td>79.55%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PARALLEL</td>
<td>74.92%</td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Figure 20. Performance of MACO

As we introduced above, we can analyze the performance of the individual analysers (Figure 20, A) on two dimensions: precision and recall (from now on we consider only the weighted measures (1)). About the first ratio, it seems that FORMARIO is the best one (96.2%), followed closely by AMCAS (94.8%) and finally SEGWORD at a considerable lower ratio (83.6%). On the other hand, considering the second ratio, recall, it seems that SEGWORD is the best one (82.1%), followed by FORMARIO at a certain distance (77.2%) and finally, at a considerable lower recall AMCAS (66.8%). The proper-nouns list seems to be very distant of the other analysers with precision and recall, 26.7% and 13.3% respectively.

Concluding, it seems that FORMARIO shows a good performance, both in precision and recall. On the other hand, SEGWORD, although it seems pretty robust (it performs the highest recall), has a rather low precision, while AMCAS shows a contrary behaviour with a pretty good
precision and a poor recall.

About the combining analysers we tried two modes of execution: backing-off, and parallel. In the second case the results obtained were excellent regarding recall (100%, i.e. all the possible analyses were generated). On the other hand, the precision was quite low, being worse than all the other configurations (except the proper-nouns analyser).

Nevertheless on executing the analysers on backing-off we can combine the good performance of the first analysers with the robustness of subsequent analysers. So, almost all of the combined analysers outperformed the individual analyser being the first applied on the sequence. Specifically:

The FO-AM-SE-PN and FO-SE-AM-PN outperformed FORMARIO. The former improves slightly the precision (+0.5% of difference) and recall (+1.3%). The later makes a little worse the precision (-0.6%) but improves significantly the recall (+3.1%).

SE-AM-FO-PN behaves in a way very similar to SEGWORD, although it improves slightly both precision (+0.6%) and recall (+0.4%).

AM-SE-FOR-PN compared to AMCAS, although it makes a little worse the precision (-0.3%), it improves the recall outstandingly (+4.9%).

Finally, PN-SE-AM-FO outstands the corresponding PN analyser: +56.13% in precision and 66.22% in recall.

As regards the comparison in performance of the different orderings of combining the individual analysers we also considered the weighted precision and recall. About the first ratio, we see that the best ones are FO-AM-SE-PN (96.7%, the highest of all), FO-SE-AM-PN (95.6%) and AM-SE-FO-PN (94.6%). At a certain distance (less than 85%) there appear the other two combinations. About the recall, the ordering is: SE-AM-FO-PN (82.6%, the highest one of all except the parallel combination), FO-SE-AM-PN (80.4%), PN-SE-AM-FO (79.6%), FO-AM-SE-PN (78.5%), and finally, at a certain distance AM-SE-FO-PN (71.7%).

Summarising, it seems that both, FO-AM-SE-PN and FO-SE-AM-PN, show a very good performance, with an excellent precision and a quite good recall. While, SE-AM-FO-PN shows a poor precision although it has the best recall, and AM-SE-FO-PN has quite a good precision but the worse recall.

The individual analysers and their combination were also tested on the whole corpus, although no human supervision was performed on these results. Consequently, we couldn’t measure ratios were the number of correct analyses was taken into account (as precision and recall). Nevertheless, we calculated the number of word forms recognised by each analyser without weighting each form by its frequency. The results are shown below with the corresponding percentage calculated respect to the 3,347 word forms of the corpus.

<table>
<thead>
<tr>
<th>Analyser</th>
<th>Word Forms</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formario</td>
<td>2,217</td>
<td>66.24</td>
</tr>
<tr>
<td>SegWord</td>
<td>3,134</td>
<td>93.64</td>
</tr>
<tr>
<td>AMCAS</td>
<td>2,517</td>
<td>75.20</td>
</tr>
<tr>
<td>Proper-Noun</td>
<td>92</td>
<td>2.75</td>
</tr>
<tr>
<td>Combination†</td>
<td>3,332</td>
<td>99.56</td>
</tr>
</tbody>
</table>

†The combination was made-up by the following analysers: SegWord, AMCAS, Formario, Number and Proper-Noun. The others were excluded either because they were tailored to the test-bed corpus (Accumulate), or because they were open-ended (Default-Cats and Initial).
From the 3,347 word forms, only 15 words remained unanalysed by all the analysers (except accumulate) (0.4%). This ratio (99.6%) cannot be considered as a recall ratio (there was no human validation) but nevertheless would confirm the figures discussed so far.

7. Conclusions.

Regarding the conclusions to be drawn from the development, use and evaluation of MACO we can reach two kinds of judgements. First, those concerning the functionality of the tool, both from the point of view of the developer and from that of the user. Second, the evaluation of the MACO analysers and their combination, in a corpus of real unconstrained text.

From our experience on using MACO as a tool to integrate different knowledge sources, the system has proven to accomplish the goals stated in section 2, i.e., flexibility, simplicity, robustness, reusability, efficiency and incrementality. In the future we want to develop a menu-guided interface to help the user in the definition of control parameters.

The performance of MACO analysers and their combination on real text seems to be particularly positive. Analysing the recall and precision of the different configurations we can see that in general all of them have excellent results on the Spanish morphology, really unbelievable when we started working on MACO. On comparing the real scope of MACO utility as a previous step to other NLP tasks such as POS tagging or parsing of unconstrained text, we have to consider as more important recall than precision, inasmuch as recall measures the ability to generate all the correct analyses while precision measures the ability for non-overgenerating.

In this regard, on the one hand it seems pretty straightforward that the recall of any backing-off strategy overpasses the individual analyser being the first one in the sequence. On the other hand the parallel combination gets a 100% recall, what it's by its own an excellent result on the integration of different (often complementary) analysers.

However the precision reached by the morphological analyser could be important in such tasks as parsing disambiguation, where too much overgeneration could make the task worse. In this regard, it also seems that the backing-off strategies get better precisions than the one obtained by the first analyser in a sequence.

About the recall and precision results of each analyser we can see that Formario shows an excellent performance although we have to take into account that the majority of the lexicon was built for the particular corpus we tested on. In addition, SegWord shows an excellent recall, while AMCAS has a very good precision.

Nevertheless, the results about precision and recall are hardly extrapolable to some NLP tasks where the required level of detail of the morphological analyses is more than the category and subcategory (that was the information taken into account at the correctness evaluation). If we take other features into account, it seems that AMCAS would have a better performance evaluation while SegWord results would drop considerably.

Finally the robustness of MACO is proven by the results provided in terms of percentage of word forms analyzed. So, all the individual analysers (of course, except Proper-Noun) surpass a 65% of word forms analyzed and SegWord reaches a 93.6%. In addition, the combination of them gets a 99.6%.

Summarising, the results seem to support the motivation of MACO as a tool for integrating different knowledge sources, as far as the results of some experiments performed on real text show that the sum of different analysers outperform the results of each one alone.
In the future we would like to extend MACO in two ways: (1) extending the linguistic coverage of the analysers, and (2) improving their combination.

The extension of the different analysers would be done through their specialisation. Thus, it seems that SegWord is doing pretty well on nouns analyses, while AMCAS has excellent results (both of precision and recall) on the verbal system. AMCAS extension would mainly consist on feeding the stems file of the AMCAS analyser with the lemmas lexicon used for SegWord in a semi-automatic way. Therefore, AMCAS recall would augment significantly while its precision would be at the same level due to the tight constraints its formalism imposes.

The modifications on the combination of the various analysers would explore ideas such as: introducing complex rules in the MACO manager to call one or other analyser depending on the results provided by previous ones, to take into account the word’s context on assigning its morphological analysis, etc.

Another limitation is that due to the facts that the preprocessor tokenizes only individual words and it is totally separated from the operation of the MACO manager, it seems difficult to integrate important information on analysing a corpus, such as word context in the text. The context can help to determine key factors on the morphological analysis of a real text such as collocations, compound words, and idiomatic expressions (a pervasive phenomena in Spanish), etc.

Finally, it might be interesting to go further on the evaluation of the different MACO configurations in two ways. First, introducing more elaborated distinctions on the evaluation test in order to take into account differences on the detail of the information obtained that could be important for some NLP tasks. This evaluation could be done directly on some of these NLP tasks, e.g. replicating the experiments made in [Briscoe et al. 94] using a richer tag set, and measuring the impact of the morphological analyser in the tagger performance. Second, it also would be interesting to calculate the recall and precision figures on the assignment of the morphological analysis of what as it appears in the corpus, and not using all the possible correct analysis as it has been done in this work.

8. References.


36


[Sanfilippo 90a] Sanfilippo A. "A morphological Analyser for English & Italian". Computer Laboratory, University of Cambridge. ESPRIT BRA-3030 ACQUILEX WP NO. 004

[Sanfilippo 90b] Sanfilippo A. "Notes on SegWord". Computer Laboratory, University of Cambridge.

## Appendix 1. MACO and Penn Treebank Tag Sets.

The following is a very rough equivalence between our codes and Penn Treebank codes. * stands for whatever value and [n] stands for any number but n.

<table>
<thead>
<tr>
<th>Code in PT</th>
<th>Our Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. CC</td>
<td>C</td>
</tr>
<tr>
<td>2. CD</td>
<td>N</td>
</tr>
<tr>
<td>3. DT</td>
<td>A**[#8]1* or J****</td>
</tr>
<tr>
<td>4. EX</td>
<td>Y</td>
</tr>
<tr>
<td>5. FW</td>
<td></td>
</tr>
<tr>
<td>6. IN</td>
<td>R or C</td>
</tr>
<tr>
<td>7. JJ</td>
<td>A**<em>0</em></td>
</tr>
<tr>
<td>8. JJR</td>
<td></td>
</tr>
<tr>
<td>9. JJS</td>
<td></td>
</tr>
<tr>
<td>10. LS</td>
<td>Z2</td>
</tr>
<tr>
<td>11. MD</td>
<td>S*-</td>
</tr>
<tr>
<td>12. NN</td>
<td>S*+</td>
</tr>
<tr>
<td>13. NNS</td>
<td>W</td>
</tr>
<tr>
<td>14. NP</td>
<td>W</td>
</tr>
<tr>
<td>15. NPS</td>
<td></td>
</tr>
<tr>
<td>16. PDT</td>
<td></td>
</tr>
<tr>
<td>17. POS</td>
<td></td>
</tr>
<tr>
<td>18. PP</td>
<td>p<em><strong>r</strong></em></td>
</tr>
<tr>
<td>19. PP$</td>
<td>p<strong>23</strong></td>
</tr>
<tr>
<td>20. RB</td>
<td>D[#8]</td>
</tr>
<tr>
<td>21. RBR</td>
<td></td>
</tr>
<tr>
<td>22. RBS</td>
<td></td>
</tr>
<tr>
<td>23. RP</td>
<td></td>
</tr>
<tr>
<td>24. TO</td>
<td></td>
</tr>
<tr>
<td>25. UH</td>
<td></td>
</tr>
<tr>
<td>26. VB</td>
<td>V20000*0</td>
</tr>
<tr>
<td>27. VBD</td>
<td>V0<em>4</em>**0</td>
</tr>
<tr>
<td>28. VBG</td>
<td>V30000*0</td>
</tr>
<tr>
<td>29. VBN</td>
<td>V10000*0</td>
</tr>
<tr>
<td>30. VBP</td>
<td>V0<em>1</em>[#3]**0</td>
</tr>
<tr>
<td>31. VBZ</td>
<td>V0*13-*0</td>
</tr>
<tr>
<td>32. WDT</td>
<td>A**81*</td>
</tr>
<tr>
<td>33. WP</td>
<td>P**[#2]4** or P<strong>8</strong>* or P<strong>4[#3]</strong></td>
</tr>
<tr>
<td>34. WP$</td>
<td>P<strong>43</strong> or P<strong>24</strong></td>
</tr>
<tr>
<td>35. WRB</td>
<td>D8</td>
</tr>
</tbody>
</table>
Appendix 2. Manager PRE Rule sets.

;;; Ruleset top

(ruleset top
  control one-cycle
  sort-proc standard-sort-proc
  sort-type static
  final-cond nil)

;;; Ruleset "apply-stages" rules

(ruleset apply-stages
  control forever
  isa top)

(rule rule-initialise
  ruleset apply-stages
  priority 1
  control one
  (name-files ^input ?)
  (process ^type ?? ^analysers ??)
  (buffer ^size ??)
  (analysers ^all ??)
  (not stream-files)
  ->
  (apply-ruleset initialise))

(rule rule-stop
  ruleset apply-stages
  priority 2
  control stop
  (analysers ^pending-set nil)
  (stream-files ^log ?log)
  ->
  (?? := (close ?log))
  (?? := (empty-wm))
)

(rule rule-open-files-start
  ruleset apply-stages
  priority 3
  control one
  (name-files ^forms ?forms ^results ?results ^pending ^pending)
  (stream-files ^log ?log)
  (process ^type start)
  ->
  (?stream-forms := (open ?forms :direction :input))
  (?stream-results :=
    (open ?results :direction :output :if-exists :new-version :if-does-
     not-exist :create))
  (?stream-pending := (open ?pending :direction :output :if-exists
     :new-version :if-does-not-exist :create))
  (modify 2 ^log ?log ^forms ?stream-forms ^results ?stream-results

39
(rule rule-open-files-continue
  ruleset apply-stages
  priority 3
  control one
  (name-files ^forms ?forms ^results ?results ^pending ?pending)
  (stream-files ^log ?log)
  (process ^type continue)
  ->
  (?stream-forms := (open ?forms :direction :input))
  (?stream-results :=
  (?stream-pending := (open ?pending :direction :output :if-exists
    :append :if-does-not-exist :create))
  (modify 2 ^log ?log ^forms ?stream-forms ^results ?stream-results
  ^pending ?stream-pending)
)

(rule rule-apply-process-words
  ruleset apply-stages
  priority 4
  control one
  (not stream-files ^stream-forms nil)
  (not stream-files ^stream-results nil)
  (not stream-files ^stream-aux nil)
  ->
  (??: := (print ?current-set))
  (??: := (print ?pending-set))
  (??: := (load-analysers ?current-set))
  (apply-ruleset process-words)
)

(rule rule-actualize-stage
  ruleset apply-stages
  priority 5
  control one
  (buffer ^size ?size ^pending ?pending ^results ?results ^dump ?dump
    ^current-register ?number)
  ->
  (?new-pending-set := (cdr ?pending-set))
  (?new-current-set := (car ?new-pending-set))
  (modify 1 ^all ?all ^pending-set ?new-pending-set ^current-set ?new-
    current-set)
  (modify 2 ^size ?size ^pending ?pending ^results ?results ^dump ?dump
    ^current-register 0)
  (apply-ruleset update-and-close-files)
)

;;; Ruleset "process-words"
(ruleset process-words
  control forever
  isa top)

(rule rule-read-first-word
  ruleset process-words
  control one
  priority 1
  (not word)
  (stream-files ^forms ?stream-forms)
  ->
  (?word := (next-word ?stream-forms))
  (create word ^form ?word ^cats nil))

(rule rule-read-next-word
  ruleset process-words
  control one
  priority 1
  (word)
  (stream-files ^forms ?stream-forms)
  ->
  (?word := (next-word ?stream-forms))
  (modify 1 ^form ?word ^cats nil))

(rule rule-eof
  ruleset process-words
  control stop
  priority 2
  (word ^form eof;)
  ->
  (delete 1))

(rule rule-not-eof
  ruleset process-words
  control one
  priority 3
  (not word ^form eof)
  ->
    ^pending-current ?current-set)
  (apply-ruleset process-word))
)

(rule rule-write-result
  ruleset process-words
  control one
  priority 4
  (word ^form ?word ^cats (?? *))
  (word ^form ?word ^cats (*categories))
  (buffer ^size ^size ^pending ^pending ^results ^results ^dump ^dump
    ^current-register ^number)
  ->
  (?new-results := (concatenate 'list ^results (list (list ^word
    ^categories)))))
)

(rule rule-write-pending
  ruleset process-words
  control one
  priority 4
  (word ^form ?word ^cats nil)
  ->
  (?new-pending := (concatenate 'list ?pending (list (list ?word)))))
  (?new-number := (+ ?number 1))
)

(rule rule-buffer-full
  ruleset process-words
  control one
  priority 5
  (buffer ^size ?size ^pending ?pending ^results ?results ^dump nil ^current-register ?number)
  ->
  (?dump := (and (eq (mod ?number size) 0)
  (not (eq ?number 0))))
  (modify 1 ^size ?size ^pending ?pending ^results ?results ^dump ?dump ^current-register ?number)
)

(rule rule-dump-buffer
  ruleset process-words
  control one
  priority 6
  (buffer ^size ?size ^pending ?pending ^results ?results ^dump t ^current-register ?number)
  ->
  (??: := (write-results ?stream-results ?results))
  (??: := (write-results ?stream-pending ?pending))
  (??: := (write-log ?stream-log 'current-register ?number))
  (modify 1 ^size ?size ^pending nil ^results nil ^dump nil ^current-register ?number)
)

;;; Ruleset "process-word"

(ruleset process-word
  control forever
  isa top)

(rule rule-stop-1

42
ruleset process-word
control stop
priority 1
(analysers ^pending-current nil)

(rule rule-stop-2
  ruleset process-word
  control stop
  priority 1
  (process ^analysers one-by-one)
  (word ^cats (?? *??))

(rule rule-assign-categories
  ruleset process-word
  control one
  priority 2
                   ^pending-current (?current *new-pending-current))
  (word ^form ?word ^cats ?previous-cats)

           ^pending-current ?new-pending-current)
  (?categories := (assign-categories ?word ?current))
  (?final-cats := (concatenate 'list ?previous-cats ?categories))
  (modify 2 ^form ?word ^cats ?final-cats)
)

;; Ruleset "initialise"

(ruleset initialise
  control one-cycle
  isa top)

(rule rule-create-names
  ruleset initialise
  control one
  priority 1
  (name-files ^input ?input)

  (?results := (concatenate 'string ?input ".results"))
  (?pending := (concatenate 'string ?input ".pending"))
  (?log := (concatenate 'string ?input ".log"))

  (modify 1 ^log ?log ^input ?input ^forms nil ^results ?results ^pending

(rule rule-continue
  ruleset initialise
  control one
  priority 2
  (process ^type continue)
(name-files ^log ?log ^input ?input ^forms nil ^results ?results ^pending ?pending)

(analysers ^all ??)

->

(?stream-log :=

(create stream-files ^log ?stream-log)

(?name-forms := (read-log ?stream-log 'name-forms))

(?current-register := (read-log ?stream-log 'current-register))

(?process-analysers := (read-log ?stream-log 'process-analysers))

(?analysers := (read-log ?stream-log 'analysers))

(?pending-analysers := (read-log ?stream-log 'pending-analysers))

(?current-set := (car ?pending-analysers))

(?new-name-forms := (concatenate 'string ?input ".aux"))


(??: (write-log ?stream-log 'name-forms ?new-name-forms))

(??: (write-log ?stream-log 'current-register 0))

(modify 1 ^type continue ^analysers ?process-analysers)

(modify 2 ^log ?log ^input ?input ^forms ?new-name-forms ^results
^pending ?pending)

(modify 3 ^all ^analysers ^pending-set ?pending-analysers ^current-set
^current-set)

)

(rule rule-start
  ruleset initialise
  control one
  priority 2
  (process ^type start ^analysers ?process-analysers)
  (name-files ^log ?log ^input ?input ^forms nil ^results ?results
^pending ?pending)

(analysers ^all ?all)

->

(modify 2 ^log ?log ^input ?input ^forms ?input ^results ?results
^pending ?pending)

(?current-set := (car ?all))

(modify 3 ^all ?all ^pending-set ?all ^current-set ?current-set)

:if-does-not-exist :create))

(create stream-files ^log ?stream-log)

(??: (write-log ?stream-log 'name-forms ?input))

(??: (write-log ?stream-log 'current-register 0))


(??: (write-log ?stream-log 'analysers ?all))

(??: (write-log ?stream-log 'pending-analysers ?all))

)

(rule rule-create-buffer
  ruleset initialise
  control one
  priority 3
  (buffer ^size ?size)

->

(modify 1 ^size ?size ^pending nil ^results nil ^dump nil
;; Ruleset "update-and-close-files"

(ruleset update-and-close-files
  control one-cycle
  isa top)

(rule rule-dump-last-buffer
  ruleset update-and-close-files
  control one
  priority 1
  (buffer "size ?size "pending ?pending "results ?results "dump ?
  "current-register ?number)
  (stream-files "forms ?stream-forms "results ?stream-results "pending
   ?stream-pending)
  ->
  (?? := (write-results ?stream-results ?results))
  (?? := (write-results ?stream-pending ?pending))
  (modify 1 "size ?size "pending nil ?results nil "dump nil "current-
  register ?number)
)

(rule rule-close-files
  ruleset update-and-close-files
  control one
  priority 2
  (stream-files "forms ?stream-forms "results ?stream-results "pending
   ?stream-pending)
  (name-files "log ?log "input ?input "forms ?forms "results ?results
   "pending ?pending)
  ->
  (?? := (close ?stream-forms))
  (?? := (close ?stream-results))
  (?? := (close ?stream-pending))
)

(rule actualise-forms-file
  ruleset update-and-close-files
  control one
  priority 3
  (name-files "log ?log "input ?input "forms ?forms "results ?results
   "pending ?pending)
  (analyzers "all ?all "pending-set ?pending-set)
  (stream-files "log ?stream-log)
  (process "type ?type "analyzers ?process-analyzers)
  ->
  (?new-forms := (concatenate 'string ?input ".aux"))
  (?? := (rename-always ?pending ?new-forms))
  (modify 1 "log ?log "input ?input "forms ?new-forms "results ?results
   "pending ?pending)
  (?? := (write-log ?stream-log 'name-forms ?new-forms))
  (?? := (write-log ?stream-log 'current-register 0))
  (?? := (write-log ?stream-log 'process-analyzers ?process-analyzers))
  (?? := (write-log ?stream-log 'analyzers ?all))
  (?? := (write-log ?stream-log 'pending-analyzers ?pending-set))
  45
(modify 5 ^type continue ^analyzers ?process-analyzers)
)

LSI-94-2-R. "Degrees and reducibilities of easy tally sets", Montserrat Hermo.

LSI-94-3-R. "Isothetic polyhedra and monotone boolean formulae", Robert Juan-Arinyo.

LSI-94-4-R. "Una modelización de la incompletitud en los programas" (written in Spanish), Javier Pérez Campo.


LSI-94-7-R. "On the expected depth of boolean circuits", Josep Díaz, María J. Serna, Paul Spirakis, and Jacobo Torán.

LSI-94-8-R. "A transformation scheme for double recursion", José L. Balcázar.

LSI-94-9-R. "On architectures for federated DB systems", Félix Saltor, Benet Campderrich, and Manuel García-Solaco.

LSI-94-10-R. "Relative knowledge and belief: SKL preferred model frames", Matías Alvarado.


LSI-94-12-R. "Analysis of an optimized search algorithm for skip lists", Peter Kirschhofer, Conrado Martínez, and Helmut Prodinger.

LSI-94-13-R. "Bases de dades bitemporals" (written in Catalan), Carme Martín and Jaume Sistac.


LSI-94-18-R “Medición del factor modificabilidad en el proyecto LESD” (written in Spanish), Núria Castell and Olga Slávkova

LSI-94-19-R “Algorismes paral·lels SIMD d'extracció de models de fronteres a partir d'arbres octals no exactes” (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-20-R “Una paral·leltzació SIMD de la conversió d'objectes codificats segons el model de fronteres al mdel d'octrees clàssics” (written in Catalan), Jaume Solé and Robert Juan-Arinyo.

LSI-94-21-R “Clausal proof nets and discontinuity”, Glyn Morrill.


LSI-94-24-R “On infinite sequences (almost) as easy as π”, José L. Balcázar, Ricard Gavaldà, and Montserrat Hermo.

LSI-94-25-R “Updating knowledge bases while maintaining their consistency”, Ernest Teniente and Antoni Olivé.

LSI-94-26-R “Structural facilitation and structural inhibition”, Glyn Morrill.


LSI-94-29-R “Towards a VRQS representation”, Mariona Taulé Delor.


LSI-94-32-R “LDB/LKB integration”, German Rigau, Horacio Rodríguez, and Jordi Turmo.

LSI-94-33-R “Learning more appropriate selectional restrictions”, Francesc Ribas.

LSI-94-34-R “Oracles and queries that are sufficient for exact learning”, Nader H. Bshouty, Richard Cleve, Ricard Gavaldà, Sampath Kannan, and Christino Tamon.
Copies of reports can be ordered from:

Nuria Sánchez  
Departament de Llenguatges i Sistemes Informàtics  
Universitat Politècnica de Catalunya  
Pau Gargallo, 5  
08028 Barcelona, Spain  
secrelsi@lsi.upc.es