Computational and methodological aspects of the Corpus Taurinense disambiguation process

Marco Tomatis

This article deals with the development of a morphosyntactic disambiguation system for the Corpus Taurinense, a corpus of old Italian Florentine texts. The aim of this project is building a reference corpus suitable to be used for training any stochastic tagging system. After presenting its general working principles, the internal structure and the error control methodology of the disambiguation program will be explained.

Introduction

The Part of Speech (POS) disambiguation of the Corpus Taurinense has represented the final phase of a broad project which was born with the aim of providing philologists and language historians a new, innovative tool for carrying on linguistic research about the origins of the Italian language. The usage of a representative corpus of the Florentine variety of Italian texts of the XIIIth Century published online in a markupped and Part-Of-Speech (POS) tagged electronic format and freely accessible to any scholar as a reference source, was the main element of innovation this project brought forth. However, notwithstanding the positive aspects, such a project required a big effort in terms of time and organization needed to prepare the tagset and all the related linguistic tools. For such reason, during its development, the important role that an accurately disambiguated version of the Corpus Taurinense could play as *training corpus* for a HMM (Hidden Markov Models) stochastic tagging systems emerged vigorously. Yet, the total lack of any previous work developed for managing old Italian texts in electronic format, forced us to treat the language contained in the Corpus Taurinense as a totally unknown entity from a computational point of view. Therefore, after evaluating all the possible ways to design and manage the disambiguation process, we realized the only feasible solution was to build a rule-based disambiguation engine. In order to avoid the need to develop a specific formalism for properly managing all our linguistic rules, we decided to adopt "GAWK", a scripting language whose syntax is very similar to the language "C", to design the whole disambiguation procedure.

The disambiguation system

As stated before, since the disambiguation program was developed using a scripting language, it works in a procedural way. Unfortunately, due to the very idiosynchratic nature of old Italian, the

disambiguation rules have to manage not only general, context-free linguistic patterns, but also quite a large amount of specific, context-bound patterns. For such reason, the whole system of rules was divided into six different independent modules acting in accordance to a specific hierarchical sequence. The first of the disambiguation modules operates on an input text which must be fully tokenized, POS-tagged and markupped. All the remaining modules run sequentially in cascade; they get as input text the result of the previous block of rules. In general terms, the first three modules are devoted to manage all the different exceptions, while the last ones contain the most general rules. In this way, the filtering action the disambiguator is able to carry on is optimized and its flexibility is improved.

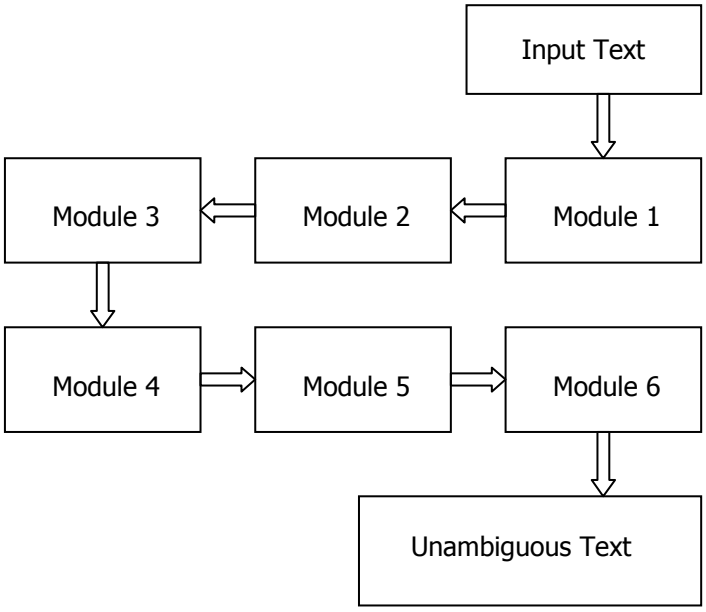

Fig. 1 - Program data flow

Within the whole corpus the disambiguation rules may face three different kinds of general ambiguous patterns, which are defined as *external*, *internal* and *inter-POS*. The first one of these patterns is related to the different POS the token may assume (e.g. *la_(lem=la,60,0,5,6,0,0);(lem=la,39,3,5,6,0,0);(lem=là,45,0,0,0,8,0)*); the second one refers to POS specific morphologic data such as gender, number, person, etc. (e.g. *che_(lem=che,36,0,4;5,6;7,0,0)*), while the third one is related to different morphologic information tied to the same POS type (i.e. verb mode, tense, etc.). Despite the hierarchical organization of the different modules, the rules they contain may operate, even simultaneously if needed, on any of the three patterns described above. As regards the internal representation of the part of speech and morphosyntactic data associated to a particular token, a numeric codification has been chosen, as shown by the examples above. Though at first glance such a solution may be considered in some

ways awkward because of the lack of immediate intelligibility of the linguistic information the corpus contains, in facts it revealed to be the most functional in terms of computational efficiency and textual compactness.

The following example is a little fragment of ambiguous input text. The first three lines represent markup and metadata, which are never analyzed by the disambiguation rules.

```
@Rinuccino@@Sonetti@@@Lir
%001
&V_lem=versesection,71,0,0,0,0,0 $0035$ #001@
D'_(lem=da,56,0,0,0,0,0);(lem=di,56,0,0,0,0,0);(lem=di,51,0,0,0,0,0);(lem=di;da,56,0,0,0,0,0)
amore_lem=amore,20,0,4,6,0,0 abiendo_lem=avere,224,0,0,0,0,0 gioia_lem=gioia,20,0,5,6,0,0
interamente_lem=interamente,45,0,0,0,8,0 ,_lem=comma,71,0,0,0,0,0
lasso_lem=lasso,26,0,4,6,8,0 ,_lem=comma,71,0,0,0,0,0 nonn¬_lem=non,45,0,0,0,8,0
aio_lem=avere,211,1,0,6,0,0 in_(lem=in,56,0,0,0,0,0);(lem=in,51,0,0,0,0,0);(lem=in,75,0,0,0,0,0)
altro_lem=altro,32,0,4,6,0,0 intendimento_lem=intendimento,20,0,4,6,0,0
né_lem=né,50,0,0,0,0,0
che_(lem=che,36,0,4;5,6;7,0,0);(lem=che,51,0,0,0,0,0);(lem=ché,51,0,0,0,0,0);(lem=che,35,0,4;5,6,0,0)
;(lem=che,40,0,4;5,6,0,0);(lem=che,32,0,4,6,0,0);(lem=che,45,0,0,0,8,0) partisse_lem=partire/-
si/,116,3,0,6,0,0 lo_(lem=lo,60,0,4,6,0,0);(lem=lo,39,3,4,6,0,0)
cor_(lem=cuore,20,0,4,6,0,0);(lem=cor,75,0,0,6,0,0) né_lem=né,50,0,0,0,0,0
la_(lem=la,60,0,5,6,0,0);(lem=la,39,3,5,6,0,0);(lem=là,45,0,0,0,8,0)
mente_(lem=mente,20,0,5,6,0,0);(lem=mentire,115,2,0,6,0,0)
```

Fig. 2 - Example of non-disambiguated input

As regards in particular the internal organization of the whole disambiguation engine, each module is made of a number of mutually excluded blocks of rules. This means that inside any module a specific text pattern can be managed by a single set of rules only. The whole text is scanned by a number of pointers that match the tokens they are pointing at with those defined inside each disambiguation rule. Only one of the said pointers is responsible for selecting the current token to disambiguate; all the other ones have auxiliary functions because they are just used to define the exact pattern the rule should match for entering into action. As the textual pattern matches, the rule is supposed to take the control and select the correct part of speech. Then, like a radio tuner, all the pointers will move on synchronously to the next token until a new correspondence between the token pattern and the one defined by the rules is found. If no matching is available, the system will automatically move the pointers forward, to the next token on their right. Since the text contains both lexical tokens and markup code, the pointer control system is designed to automatically recognize and avoid all the non-lexical elements in the text. In such a way it is possible for the rules to take into account the lexical structure only, without being influenced by any external code.

The picture below (fig. 3) is a graphical representation of the just described working flow.

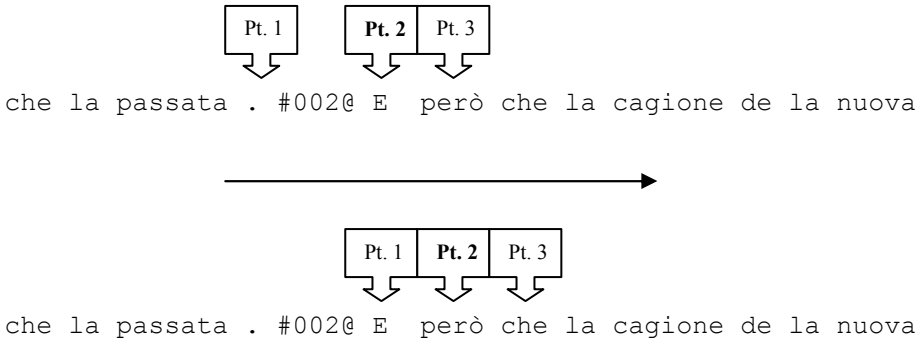

Fig. 3 - The pointers working principle

Organization of the rules

As already mentioned before, within the six modules the set of linguistic rules are strictly organized in a hierarchical structure. More specifically, the first three modules have been designed for taking into account all the particular, context sensitive word patterns, the fourth module contains a mixture of ad-hoc and general rules, while the last two modules host context-free, general rules only. In particular, the whole set of rules is grouped into 1059 different blocks, each of which is designed to manage one specific ambiguity only. Yet, since different tokens may show a variable number of ambiguous POS values they are tied to, any disambiguation block may contain up to seven different POS selecting rules. The following is an example of a block of rules written in GAWK, taken from the second module:

```
# (2) Regola vera e propria
#
# Regola per la disambiguazione esterna di 'piano'
#
      else
      if (campo ~ /^piano_/ && campo ~ /\);\(/)
            {
            if (campo ~ /¥$/)
                  end = "¥"
            else
                  end = ""
            nf++
            if ($bw ~ /^troppo_/ || $bw ~ /^e_/ || $fw ~ /^che_/)
                  {
                  assegna(campo, "45", end)
                  }
            else
            if ($bw ~ /^÷l_/ || $bw ~ /^a_/ || $bw ~ /^di_/)
                  {
                  assegna(campo, "20", end)
                  }
            else
                  {
                  assegna(campo, "26", end)
                  }
            }
```

Fig. 4 - The disambiguation rules for *piano*

Translated into current English, the three rules above mean:

1. The word “piano” is an adverb (POS “45”) if the token before it is “troppo” or “e” or if the following token is “che”.
2. The same word is a noun (POS “20”) if the token before it is “÷l” or “a” or “di”
3. “piano” is an adjective (POS “26”) in all the other cases.

Within the six modules, the block of rules are distributed in the following way:

Module typology	Ambiguous POS managed	Block of rules
1 – ad-hoc rules	verbs, nouns, adjectives	547
2 – ad-hoc rules	nouns, adjectives, adverbs	265
3 – ad-hoc rules	prepositions, conjunctions, adverbs	150
4 – ad-hoc and general rules	verbs, prepositions, conjunctions	57
5 – general rules	determiners	19
6 – general rules	nouns, adjectives	21

Fig. 5 - The program modules organization scheme

The table above shows a far lower presence of rules in the last three modules if compared with the first three ones. This can be easily explained by examining the scope of the general rules, which are able to take into account a broader variety of possible ambiguous cases if compared with the ones designed for managing specific ambiguities only. Related to this aspect, it is also worth highlighting that in comparison with nouns or adjectives, the verbs disambiguation process resulted to be the most expensive in terms of overall quantity of rules involved.

The error tracking system

Altough the technical framework we adopted provided us a convenient way to design the different linguistic rules, during the development of the disambiguation system a number of organizational problem raised. In particular, we had to face various issues related to the development of both a hierarchical control flow and an error control method the different rules had to be subject to. Whereas the first aspect was quite easily solved by grouping together within the same module all the rules dealing with a particular POS, the development of an error control method required a more considerable effort. Indeed, because of the large amount of rules the disambiguation system is made of, a single solution revealed not to be sufficient enough for managing all the possible errors the developer involuntarily could make during his work. As a matter of fact, any mistake inside the disambiguation rules may produce two different kind of errors. The first one, the most nasty and hard to pinpoint inside the output text, is produced by the system any time a discrepancy between

the POS selected by a rule and the POS grabbed by the main pointer occurs. In such a case, due to the lack of matching between the said values, the program would be forced to select and print a non-existent entity. Consequently, the resulting text would be affected by the absence of specific tokens. Since such a problem could involve the output of any module, we found the most feasible solution was to compare the output of two consecutive modules and check that the number of tokens present in each line of both files was exactly the same. Doing so, the control program would be able to find one or more missing tokens in one or more different lines, providing the developer all the information needed to track and solve the program errors.

The second kind of problem affecting the disambiguation rules is related to the definition of their very scope. Usually, before realizing an effective linguistic rule, it is necessary to spend lots of time in a trial-and-error testing process involving the whole set of modules. We aimed at avoiding that time losing activity, therefore we had to completely change our methodological approach. For testing the scope effectiveness of a specific rule, a testing module called "PEX" was prepared. Differently from the standard module structure, the PEX allowed the developer to run only one rule per time. In this way, it was easier for the user to understand when a new rule was not working properly or when its scope had to be adjusted or modified in some ways. The great advantage of adopting such a different methodology was the huge amount of time earned by the developer, who was not forced to wait for the completion of the whole disambiguation process to analyse the results of his work.

Future developments

Altought the system described in this paper would be ready to prepare a training corpus able to feed a stochastic POS tagging procedure, in fact the representativeness of the current version of the Corpus Taurinense is still too low. For overcoming such limitation, we are planning to increase the amount of linguistic data inside the corpus by enhancing it with other Florentine XIIIth Century selected texts. Consequently, for taking into account all the grammatical changes the new tokens will produce inside the whole text structure, specific new disambiguation rules will be created, while the old ones will be modified consequently, in accordance to any specific need that from time to time may rise.

Bibliography

BARBERA, Manuel / CORINO, Elisa / ONESTI, Cristina, 2007, "Cosa è un corpus? Per una definizione più rigorosa di corpus, token, markup". In: BARBERA, Manuel / CORINO, Elisa / ONESTI, Cristina (a c. di), *Corpora e linguistica in rete*, Perugia, Guerra Edizioni. ('L'officina della lingua. Strumenti'): 25-88.

BARBERA, Manuel, 2007, "Un tagset per il Corpus Taurinense". In: BARBERA, Manuel / CORINO, Elisa / ONESTI, Cristina (a c. di), *Corpora e linguistica in rete*, Perugia, Guerra Edizioni. ('L'officina della lingua. Strumenti'): 135-168.

BARBERA, Manuel / MARELLO, Carla (a c. di), 2001, *Linguistica dei corpora per l'italiano antico. Annotazione morfosintattica di testi fiorentini del Duecento*, Alessandria, Edizioni dell'Orso ('Gli argomenti umani' 6).

BRENNAN, Michael, 2000, *GAWK: Effective AWK Programming: A User's Guide for GNU AWK*, 2nd edition, Free Software Foundation Inc.

TOMATIS, Marco, 2007, "La disambiguazione del Corpus Taurinense. Problemi teorici e pratici". In: BARBERA, Manuel / CORINO, Elisa / ONESTI, Cristina (a c. di), *Corpora e linguistica in rete*, Perugia, Guerra Edizioni. ('L'officina della lingua. Strumenti'): pp. 169-181.

TOMATIS, Marco, 2009, "La disambiguazione. Trattamento finale". In: BARBERA, Manuel, *Schema e storia del "Corpus Taurinense". Linguistica dei corpora dell'italiano antico*, Alessandria, Edizioni dell'Orso: 171-191.