# A Logic Programming Application for the Analysis of Spanish Verse

Pablo Gervás

Departmento de Inteligencia Artificial,
Universidad Europea de Madrid,
Villaviciosa de Odón, Madrid 28670,
pg2@dinar.esi.uem.es

**Abstract.** Logic programming rules are provided to capture the rules governing formal poetry in Spanish. The resulting logic program scans verses in Spanish to provide their metric analysis. The program uses DCG grammars to model the division of each word into syllables, and additional predicates are employed to define metric phenomena such as synaloepha, syllable count of a verse, rhyme of a word... The system is tested over a set of Spanish Golden Age sonnets and shown to give reasonable results, providing a very useful pegdagogical application for teaching Spanish poetry.

## 1 Introduction.

Logic programming has provided many tools for natural language processing (see [1, 11]). It has been applied with success in various fields like generation, understanding, parsing, and semantic representation. The present paper extends the application of logic programming to natural language processing beyond everyday language. Logic programming rules are provided to capture the rules governing formal poetry in Spanish. The resulting logic program scans verses in Spanish to provide their metric analysis. As such it constitutes a useful tool for literary study of Spanish poetry, or as a pedagogical tool for teaching students how Spanish poetry is analysed. The system is tested over a set of Spanish Golden Age sonnets and shown to produce good results.

The system has immediate aplication as a pedagogical tool to help in the process of teaching Spanish poetry and metric at school and university level. As well as scanning metrically a set of verses, the system allows easy location of places where the poet has used poetic licenses such as hiatus, dieresis and syneresis, thereby providing a good tool for stylistic analysis of texts.

## 2 Description of the problem.

Formal poetry in Spanish is governed by a set of rules that determine a valid verse form and a valid strophic form. A given poem can be analysed by means of these rules in order to establish what strophic form is being used. Another

set of rules is applied to analyse (or *scan*) a given verse to count its metrical syllables. This paper presents a logic programming application that carries out this analysis. The program uses DCG grammars to model the division of each word into syllables, and additional predicates are employed to define metric phenomena such as synaloepha, syllable count of a verse, rhyme of a word...

### 2.1 Parsing the Syllables in a Word.

As a starting point the process of scanning a verse requires a preliminary decomposition of the verse into syllables. These syllables, as understood by the layman with no notions of formal analysis of poetry, do not match metric syllables, but they constitute the starting point of the analysis. The Spanish language presents advantages over other languages in this respect in the sense that one can work out the division of a word into syllables algorithmically from the way it is written. The process of parsing the list of letters corresponding to a word in order to generate a list of syllables applies a set of ortographical rules governing the way letters are grouped together into syllables (see [8, 7]). These rules have to be taken into account when parsing automatically. The next step in the process of scanning a verse is to take into account metric considerations that transform this initial list of syllables into metric syllables, and to generate a count of metric syllables in the verse. These processes depend on the position of the stressed syllables of each word.

### 2.2 Locating Stressed Syllables.

The accents that need to be counted are not those that are usually represented by a slanted dash over a word. Every word has its own prosodic accent, and this is placed over the *sílaba tónica*, the one that carries the stress when it is pronounced. Spanish does have a set of rules that allow automatic location of the stressed syllable of a word from its written form (see [8]). This is important because it allows analysis with no need for a lexical entry for each word in the verse.

According to the distance from the stressed syllable to the end of the word, words are classified into three different types:

- *palabras agudas*, or oxytone words, those in which the stressed syllable is the last syllable of the word
- *palabras llanas* , or paroxytone words, those in which the stressed syllable is the one before the last syllable of the word
- *palabras esdrújulas* , or proparoxytone words, those in which the stressed syllable lies two syllables from the end of the word

### 2.3 Metric Syllable Count.

In working out the count of metric syllables of a verse the concept of syllable involved differs slightly from its everyday equivalents. A metric syllable does not

always match the corresponding morphological syllable. When a word ends in a vowel and the following word starts with a vowel, the last syllable of the first word and the first syllable of the following word constitute a single syllable. This is known as *synaloepha* (see [7], or [13] for an overview in English), and it is one of the problems that we are facing. For instance, the following list of syllables for a verse

bás - te - te a - mor lo que ha por mi pa - sa - do
13 syllables

turns into the list of metric syllables:

bás - te - **te_ a** - mor lo **que _ ha** por mi pa - sa - do
11 syllables

because it shows two instances of synaloepha (marked in bold).

Another phenomenon that affects the syllable count is given by the position of the accent of the last word a verse (see [7], or [13] for an overview in English). If the last word is a *palabra llana* (stress one syllable from the end) the verse is considered to have as many metric syllables as it has morphological syllables.

to - dos sen - ti - dos hu - ma - nos
8 metric syllables

If the last word is a *palabra aguda* (stress right at the end) the verse is considered to have one more metric syllable than it has morphological syllables.

A - sí, con tal en - ten - der
7 + 1 = 8 metric syllables

If the last word is a *palabra esdrújula* (stress two syllables from the end) the verse is considered to have one metric syllable less than it has morphological syllables.

A - mor, tus fuer - zas rí - gi - das
8 - 1 = 7 metric syllables

### 2.4 Distribution of Stressed Syllables.

The considerations presented so far describe how a division of a verse into metric syllables and the corresponding metric syllable count is obtained. Formal analysis of poetry also studies the position of stressed syllables over a verse. For the verse to sound pleasing, the prosodic accents must be distributed according to precise patterns. This distribution of prosodic patterns provides the quality of being pleasant to the ear.

For instance, for an eleven syllable long verse to sound pleasing, it needs some of the stressed syllables of its words to fall on certain specific positions. It is not

necessary for the stressed syllables of every word in the verse to be in specific positions. It is enough for certain strategic syllabic positions within the verse to have a stressed syllable. There are four accepted combinations that produce the required prosodic effect (see [7, 13]).
1) Stressed syllables fall on positions 1, 6 and 10. The verse is then referred as an *endecasílabo enfático*. In the following examples, one such verse is first given in its original form, followed by a divided version in which words have been split up into syllables and syllables from adjoining words are linked together whenever *synaloepha* has occurred. Syllabic positions are numbered in the line below, and stressed syllables falling in key positions are marked out in bold

**bás-** te- **te_ a- mor** lo **que _ ha** por mi pa- **sa-** do
1 2 3 4 5 6 7 8 9 10 11

2) Stressed syllables fall on positions 2, 6 and 10. The verse is then referred as an *endecasílabo heroico*. For instance "en verdes hojas vi que se tornaban":

en- **ver-** des ho- jas **vi** que se tor- **na-** ban
1 2 3 4 5 6 7 8 9 10 11

3) Stressed syllables fall on positions 3, 6 and 10. The verse is then referred as an *endecasílabo melódico*. For instance"a la entrada de un valle, en un desierto":

a la_ en- **tra-** da de_ un **va-** lle_ en un de- **sier-** to
1 2 3 4 5 6 7 8 9 10 11

4) Stressed syllables fall on positions 4, 6 or 8, and 10. The verse is then referred as an *endecasílabo sáfico*. For instance, the verse "que con lloralla cresca cada día":

que con llo- **ra-** lla **cres-** ca ca- da **dí-** a
1 2 3 4 5 6 7 8 9 10 11

Once a verse has been scanned, and the stressed syllables have been located for every word in the verse, analysis of the positions of stressed syllables is a simple matter.

### 2.5 Extracting Word Rhyme.

In order to identify the strophic form of a given poem it is important to identify the rhyme of each verse. In Spanish (see [7, 13]) the last vowel of the verse and all the following letters (both vowels and consonants) are the same in verses that rhyme. The rules described in the previous sections also allow determination of the set of letters at the end of the word that constitute its rhyme.

### 2.6 Related Work.

Work along similar lines has been carried out for Italian Renaissance poetry [9], French [3], Middle Indo-Aryan [4], and Old English [2]. In particular, [9] discusses the need for metrically scanned electronic versions of classical texts, and discusses the severe limitations of automatic procedures for marking accents and counting syllables in the case of Italian. As a conclusion of this discussion, an interactive program is suggested as the best means of scanning the text metrically. Similar difficulties are met in [3] while dealing with French text. French requires prior specific transcription of the written text into phonemes before the metrical analysis can be carried out. The study of Middle Indo-Aryan Prakrit in [4] presents an additional problem due to the fact that a special character set is required for the language, so the text must be first be transcribed onto Roman script. The work carried out on Old English in [2] concentrates on analysis of phonological patterning, with particular emphasis on alliteration.

Spanish does not present as many difficulties as Italian or French, because the phonetics of a word can be unambiguously obtained from its written form. This includes word accents, since the language provides special means of representing accents graphically. However, Spanish presents the additional difficulty of allowing a number of poetic licenses concerning metrical scansion. This implies that scansion of Spanish verse cannot be decided unambiguously without resorting to the context. A verse scanned as 12 syllables may be re-interpreted as 11 syllables long if it appears in the context of a poem built solely of 11 syllable long verses, provided that the right conditions for poetic licence to occur are met.

Existing work on analysis of Spanish poetical texts, such as [10], applies computational techniques to six main streams of research: sentential analysis, word length analysis, running words analysis, word frequency analysis, use of words analysis, and cluster analysis. At present we are not aware of any work that concentrates specifically on the metrical analysis of Spanish verse.

## 3 The Logic Programming Implementation.

The analyser presented in this paper is implemented in SWI Prolog (see [12]). In order to carry out the whole set of analyses required to decide whether a given text is a correct poem, verses must be treated as lists of words, words must be treated as lists of characters (later as lists of syllables) and syllables as lists of characters. Prolog provides very useful mechanisms to process such a representation. In this program, Definite Clause Grammars (DCG) (see [5]) have been used to encode the different stages of analysis. A detailed description of the implementation of each step is given below.

The predicate that carries out the analysis of a single verse is implemented as a predicate `analysis`. The predicate `analysis` obtains the following information from a verse represented as a list of words: number of syllables after applying all the rules, list of syllables in the verse, list of positions of stressed syllables, and rhyme of the verse.

For instance, for the following call:

```
?- analysis([Cerrar,podra_,mis,ojos,la,postrera],N,Ls,La,Ri).
```

the system would return the following results:

```
N = 11
Ls = [Ce,rrar,po,dra_,mis,o,jos,la,pos,tre,ra]
La = [2,4,5,6,8,10]
Ri = era
```

These items are obtained in two stages. To carry out the task in hand, the system needs to know the following information about each word in the verse: number of syllables, list of syllables in the word, position of stressed syllable of the word, and rhyme of the word.

The first step during analysis is to check whether such information is available in the system vocabulary (a database of facts) for all the words in the verse. If data are missing for a given word of the verse, they are worked out from the word using the rules as described below. The system adds this information to the database of vocabulary facts available for metric analysis.

The second step starts when all data are available. The verse is analysed recursively, using the data worked out for each word, and applying the metric rules to scan the verse properly. Two types of analysis are possible: finding the list of syllables of a word (and therefore the metric syllable count for the verse), or finding the list of positions of stressed syllables (which determines the correctness of the verse).

The division of the general process into two different stages gives the system exceptional flexibility when confronted with new texts. On the one hand, any unfamiliar word appearing in the text can be analysed to obtain the required data. On the other hand, the system may resort to previously stored analysis to avoid recomputation of the data for a specific word. This may have a dual effect on the efficiency of the system, transferring the load from computation time to memory requirements. And additional advantage lies in the fact that there are many exceptions to the general rules implemented in the system. The rules capture the general pattern of word formation of the Spanish language, but many words borrowed from other languages but now fully accepted do not conform to these rules. The particular structure proposed allows the relevant data for problematic to be loaded into the system prior to its use, therefore avoiding errors during the analysis due to exceptions to the rules appearing in a text.

The rest of this section provides brief descriptions of the different processes involved in the analysis.

### 3.1 Syllabic Analysis of a Word: Word Syllable Parser.

The Word Syllable Parser module takes as an input the list of characters of the word to be analysed. The complete operation is carried out in three stages of parsing.

**Parsing Character Type.** The first parse classifies each character (or group of two characters) into one of 14 different categories according to its ability to combine with other characters to form basic sound groups (double consonants, consonants that can group with other consonants, high, low or middle vowels). Information about vowel classification plays an important role in sorting out syllable boundaries in diphthongs (see [7, 13] for details). For instance, an *r* following an *r* is grouped into a single *rr* 'double consonant' group. This grouping is carried out by a DCG of 4 basic rules and 63 rules describing terminals. This proliferation of rules for terminals is due to the fact that both upper and lower case characters must be taken into account, and combinations of these in cases of double consonants. Only the simplest examples of the grammar are presented here. There are many variations and many exceptions to each rule (*u* after *q* acts as a single consonant, *ns* at the end of a syllable cannot be separated, double consonants *ch*, *rr*, *ll* and *pr* need separate rules...).

Terminal elements of the grammar are defined as facts of the form:

```
vowel(va([a]))-->[a].                    /* a */
vowel(vm([e]))-->[e].                    /* e */
vowel(vc([i]))-->[i].                    /* i */
vowel(vu([u]))-->[u].                    /* u */

consonant(n([n]))-->[n].
consonant(cp([p]))-->[p].                 /* p */
consonant(cl([l]))-->[l].                 /* l */
consonant(c([v]))-->[v].

double_consonant(cd([c,h]))-->[c,h].      /* ch */
double_consonant(cd([r,r]))-->[r,r].      /* rr */
```

Vowels are classified into high (*i,u*), middle (*e,o*)or low (*a*). This information is required to work out the behaviour of diphthongs (see [8]). Groups of different vowels are grouped together into a single syllable (a dyphthong) when certain circumstances are met. These circumstances are specified in terms of whether the group of vowels is made up of different combinations of vowels from one group or another, and whether the corresponding vowels are stressed or not. The rules for the grouping of vowels into dyphthongs requrie this information at a later stage, so the distinction is noted during this stage of the analysis.

The vowel *u* requires special treatment because it also acts as auxiliary to consonants *g* and *q*.

Consonants are classified according to their ability to form different letter groups either with other consonants (groups `ns`, `*r`, `*l`...) or with specific vowels (groups `qu`, `gu`...). Letters with different combination properties are identified as members of the specific group and marked as such at this stage. In each case, a functor is generated that identifies the type of letter considered, and contains the letter itself (in list format if it is a complex letter such as a double consonant) as an argument.

The rules of the grammar simply parse the input assigning categories to each group of letters:

```
parse_letters([])-->[].
parse_letters([X|Rest])-->double_consonant(X),
                          parse_letters(Rest).
parse_letters([X|Rest])-->consonant(X),
                          parse_letters(Rest).
parse_letters([X|Rest])-->vowel(X),
                          parse_letters(Rest).
```

**Parsing Character Groups.** The second parse operates over the output of the first and joins together basic sound groups that make up either a consonantal group or a vowel group. For instance, *pr*, *gl*, *ns*, group together as consonants; and vowels are grouped into diphthongs or triphthongs according to the rules. This DCG has 60 rules. The basic rule for this level of analysis (identifying groups of consonants and groups of vowels) is defined as:

```
group_letters([])-->[].
group_letters([X|Rest])-->group(X),group_letters(Rest).
```

The rest of the rules take the form:

```
group(cons(X))-->[cd(X)].
group(cons(X))-->[c(X)].

group(gv(X))-->[va(X)].
group(gv([A,B]))-->[vc([A]),vm([B])].
```

Certain cases require a step of look-ahead in order to make the right decision. For instance, the rules for *g*, *u* combinations before *i* and *e* (in which cases the *u* is silent); or cases where a certain combination of vowels forming a diphthong is broken by the graphic accent of *tilde*[1]. For instance, the combination of a low vowel and a stressed high vowel, such as *...aí...* does not form a dyphthong and should be parsed into two syllables, whereas the combination of a stressed low vowel and a high vowel, such as *...ái...* does, and should be parsed into the same syllable.

```
group(cons([G,U]), [gr([G]),vu([U]),vm([e])|X],[vm([e])|X]).
group(cons([G,U]), [gr([G]),vu([U]),vc([i])|X],[vc([i])|X]).

% dyphthong
group(gv([A,T,B]))-->[va([A]),ac([T]),vu([B])].

% no dyphthong (requires lookahead)
group(gv(A),[va(A),vc(B),ac(T)|X],[vc(B),ac(T)|X]).
```

[1] The *tilde* is represented in the system as an `ac(T)` functor.

The examples presented here are only a selection of particular instances of the type of rule discussed in each case.

**Parsing Syllables.** The third parse takes as input the list of consonant and vowel groups and tacks together those that form valid syllables. This DCG has 14 rules. The basic structure of this grammar is:

```
find_syllables --> syllable,find_syllables.
find_syllables --> [],!.

syllable -->initial_consonant,vowel_group,final_consonant.
syllable -->initial_consonant,vowel_group.
syllable -->vowel_group,final_consonant.
syllable -->vowel_group.
```

These rules include an additional argument which returns the list of the syllables found for the word, each one converted into a Prolog atom.

**Locating the Stressed Syllable of a Word.** Simple Prolog predicates carry out the task of locating the stressed syllable of a given word, taking as input the list of syllables. The representation in use includes a notation symbol for the tilde, used in Spanish orthography to help locate stressed syllables using a few simple rules. This rules are taught to every primary school student (and forgotten regrettably fast!). For a detailed reference, see [8].

**Extracting the Rhyme of a Word.** The rhyme of a word is given by the fragment of the word that lies beyond the last stressed vowel of the word. The rhyme of each word is obtained by a predicate `obtain_rhyme` that operates over the list of syllables for that word. Taking into account the location of the stress in the word, it truncates the stressed syllable to obtain its tail (from the stressed vowel to the end) and it appends to it all the remaining syllables to the end of the word. The procedure is made up of three rules. Each rule deals with one possible stress pattern for the word. Every rule works by locating the last stressed syllable, applying the predicate `end_of_syllable` (which returns the portion of the syllable which lies beyond the last stressed vowel), and joins it together with any remaining syllables between the stressed syllable and the end of the word.

The information about the rhyme of a word is also included in the fact stored for it in memory. Therefore, during the metric analysis of a verse, identifying the rhyme of a word involves just querying the vocabulary database.

**Managing the Vocabulary Database.** The system operates with a vocabulary database that is declared in the form of Prolog facts. When the system reads a word, whether during selection of rhymes or during the writing of a draft, it consults the vocabulary database. If the word is not found, the required procedures are invoked to obtain the necessary information.

This information is stored as facts of the form:

```
known_word(1,ar,[ce,rrar],no,no,cerrar).
```

where: the first argument `1` shows the position of the stressed syllable from the end of the word, `ar` is the rhyme of the word, `[ce,rrar]` is the list of syllables of the word, the first `no` indicates whether the word starts with a vowel, the second `no` indicates whether the word ends with a vowel, and the final argument `cerrar` is the word itself, to be used as key when retrieving the rest of the information.

If a given word in the text was not present could not be found in the database, the results of the analysis are declared in memory in the same format as existing items in the database. At the end of a session, the system allows the updated database to be saved. This makes the result of all new words that have been already parsed available in database form for later analyses.

This method allows the option of starting the system with an empty vocabulary database each time (thereby optimising memory use), or to build the vocabulary database incrementally by loading a previous version before the analysis of each new text. This second option implies growing memory requirements for subsequent executions, but improves the computational efficiency of the analysis. It has the additional advantage of progressively building a vocabulary database of metrically analysed Spanish words.

### 3.2 Metric Analysis of a Verse: The Metric Syllable Counter.

The metric syllable counter is defined as a set of Prolog predicates that operate recursively over the list of words in a verse. All the information about the division into syllables and the location of the stressed syllable obtained for a given word during previous two stages is declared in memory as Prolog facts. In this way, while processing each word in the verse the syllable counter has access to the list of syllables of the word and the position of the stressed syllable. From this information it can also determine easily whether a word starts or ends with a vowel.

**Counting the Metric Syllables in a Verse.** The predicate `word_analysis` works out the scansion of the verse and it returns a list of metric syllables corresponding to that verse, as well as the number of syllables found in the verse.

A simple predicate `synaloepha` implements the decision rule for synaloepha, and appends two contiguous syllables whenever necessary. This is done by means of two auxiliary predicates (`start_vowel` y `end_vowel` ) that identify whether a word starts or ends with a vowel. When processing a word, the final syllable of the previous word is carried over, together with a variable indicating whether it ended in a vowel or not. The last syllable of the previous word is treated as if it were part of the word. According to the variable one or the other rule of the procedure is used. If synaloepha takes place, the rule that requires the previous word to end in a vowel simply acts as an interface, calling the other version with the corresponding alterations of the list of parameters: the number of syllables found so far is reduced by one, and the last syllable of the previous word and

the first syllable of the present one are fused into a single syllable. Since this operation is carried out recursively, two syllables appended in this way may yet be appended to a third one if the conditions are right. This is quite common when one vowel articles such as *a* occur between words that finish and start with vowels. This method requires special attention to be paid to border situations. For the first call (at the beginning of a verse) an additional ghost syllable must be added. This ghost syllable must be eliminated from the final list of syllables at the end. In the same way, for the last call (which would given the empty list as a result) the last syllable of the previous word must be returned as sole member of the list of syllables found, to ensure that it is not lost.

The predicate `reevaluate` takes into account the effect on the count of metric syllables of the verse (as opposed to prosodic ones) of the position of the stressed syllable of the last word. It has three different versions, one for each possible pattern of stress placement. They all carry out basic arithmetic operations over the resulting number of syllables: subtract one if the last word is proparoxytone, leave as it is for a paroxytone word, add one for an oxytone word.

A similar predicate, `accent_analysis`, operating over the same list of words for the verse and using the information facts in memory for each word, translates the list of words into a list of stressed syllable positions. This process is parallel to the one carried out when creating the list of syllables, with a numerical counter taking the part of the syllables.

**Using Results to Validate a Verse.** Over the results of the analysis, a diagnostic of the correctness of the verse can be obtained by pattern matching with a set of valid patterns declared in memory. The predicate `conclusions` takes the obtained results and works out whether the verse under analysis is valid as *endecasílabo*. If it is, the type of *endecasílabo* is identified by checking the positions of the stressed syllables.

In the example above, the verse:

Cerrar podrá mis ojos la postrera

is an *endecasílabo heróico*, because it is eleven syllables long (`N = 11`)and its list of positions of stressed syllables (`La = [2,4,5,6,8,10]`) contains the key positions `2, 6, 10`.

## 4 Evaluation of the system.

The system has been evaluated over a set of classical poems. The number of incorrect analyses is worked out as a percentage. Specific sources of error are identified and their contribution to the general error is discussed. In each case, possible solutions are discussed.

### 4.1 The choice of test data

The system has been tested over a set of 64 classic Spanish Golden Age sonnets (taken from [6]). These sonnets were fed to the system in separate files containing the sonnets by different authors. The system output a file with the syllable count, list of syllables, list of stressed positions, and rhyme for each of the given verses. Sonnets were chosen as benchmark poems because they have a very rigid formal structure. Every verse in a Spanish sonnet must be 11 syllables long according to the rules. Over the resulting files, verses that had been assigned by the system a syllable length other than 11 were hand checked for errors.

### 4.2 The results

The results obtained are presented in table 1.

**Table 1.** Raw Results

| Author | Sonnets | Verses | Errors | % Error |
|---|---|---|---|---|
| Quevedo | 21 | 294 | 31 | 89.5 |
| Lope | 16 | 224 | 17 | 92.4 |
| Góngora | 11 | 154 | 19 | 87.7 |
| Garcilaso | 8 | 112 | 21 | 81.3 |
| Boscán | 3 | 42 | 2 | 95.2 |
| Aldana | 5 | 70 | 11 | 84.3 |
| Totals | 64 | 896 | 101 | 88.7 |

### 4.3 The problem of poetic license

A poet is allowed a number of poetic licences to make the verses fit into the metric structure. These poetic licences are known as *syneresis*, *dieresis*, and *hiatus* (see [13] for details). Of these, *dieresis* allows a diphthong to be broken (adding one syllable to a given word), *syneresis* allows an illegal diphthong to be created (subtracting a syllable to a word), and *hiatus* allows synaloepha to be broken (adding a syllable to a given verse). Resolving issues of poetic license presents special problems because the main idea behind the concept is too allow the same verse to be parsed as having a certain length in one specific context and as having a different one in another. This is achieved by overlooking or enforcing two different metric rules: the rules for synaloepha and the rules for dyphthong formation. Two syllables that might constitute synaloepha (or two vowels making a dyphthong) and be parsed as one under normal conditions can exceptionally be broken up if the poet needs an extra syllable to his verse at that

stage, resulting in hiatus (or dieresis). Alternatively, syllables (or vowels) that would be parsed as separate may be parsed as a unit if the poet wants one syllable less, resulting in hiatus (or syneresis). The fact that correct solutions are context dependent means that they cannot be implemented for isolated verses, but only when verses are scanned as part of a whole poem, in which case, the metric rules for the poem impose specific lengths on verse. In a wider setting, these poetic licenses can easily be solved by rephrasing synalopeha and dypthhong rules as defeasible inferences (or simply allowing the implicit backtracking in Prolog to find the alternative solution).

The system as it stands allows easy location of poetic licenses, which gives added-value when analysing poetry. The results for poetic licenses obtained for the test data are presented in table 2. The number of poetic licenses taken in each set of poems is listed for each kind. The percentage of error when problems of poetic license are discounted is also shown. A great improvement in system efficiency is appreciated.

**Table 2.** Poetic Licenses

| Author | Verses | Hiatus | Dieresis | Syneresis | Total | Real % error |
|---|---|---|---|---|---|---|
| Quevedo | 294 | 5 | 1 | 1 | 7 | 97.6 |
| Lope | 224 | 6 | 2 | 1 | 9 | 96.0 |
| Góngora | 154 | 3 | 4 | 4 | 11 | 92.9 |
| Garcilaso | 112 | 10 | 2 | 2 | 14 | 87.5 |
| Boscán | 42 | 0 | 0 | 0 | 0 | 100 |
| Aldana | 70 | 2 | 3 | 1 | 6 | 91.4 |
| Totals | 896 | 26 | 12 | 9 | 47 | 94.8 |

### 4.4 Detected errors and planned improvements

The most important source of real errors is the conjunction *y*, which can act both as a vowel and as a consonant. This creates problems when determining whether synaloepha is possible or not between words. The rules for synaloepha must be redesigned to acoutn for these variations.

The remaining errors can be attributed to exceptions in diphthong formation rules. Although there are generally accepted rules governing diphthong formation, certain cases constitutes exceptions (for historical or etymological reasons). Foreign words that have been accepted into Spanish also constitute exceptions to the syllabic rules. The fact that the system keeps a database of facts for the words in its vocabulary allows exceptions to the rules to be declared directly into the database. In this way, known exceptions can be included from the start, thereby improving the accuracy of the system.

Table 3 shows final results for the system once problems of poetic licence and probles related with conjunction are excluded.

**Table 3.** Final Results

| Author | Sonnets | Verses | Y Errors | % Error |
|---|---|---|---|---|
| Quevedo | 21 | 294 | 20 | 98.6 |
| Lope | 16 | 224 | 7 | 99.6 |
| Góngora | 11 | 154 | 8 | 100 |
| Garcilaso | 8 | 112 | 6 | 99.1 |
| Boscán | 3 | 42 | 2 | 100 |
| Aldana | 5 | 70 | 5 | 100 |
| Totals | 64 | 896 | 48 | 99.3 |

## 5 Conclusions and Further Work.

Several conclusions can be drawn from the above observations. On one hand, the logic programming tools used to implement the system have demonstrated their power and flexibility in coping with a complex problem of symbolic processing. On the other hand, the system shows potential both in the fields of literary analysis of large texts, and in the field of teaching.

### 5.1 Advantages of Logic Programming for the Task.

The problem tackled in this paper presented a complex structure of several layers of analysis (at character level, at character group level, at syllable level, at the level of words in a verse), each implemetned as a DCG. Solutions with fewer layers are possible, and some have been tested during the development of the system, but were rejected because of the excessive amount of backtracking required whenever information from bottom layers affects the decision process in top layers.

The declarative nature of logic programming, and the modular design of the program (with general rules set distinctly apart from specific data for a given word in the vocabulary database) allows very easy encoding of exceptions. It also allows the system to act both as a parsing tool and as a vocabulary database generator. None of these advantages would have been available in an imperative implementation of the system, even though such an implementation would possibly carry out the parsing problem more efficiently if all the different cases were coded. The modularity of the program allows easy modification and might allow reuse of some parts of the code for languages with similar phonetic structure.

### 5.2 The System as a Practical Tool.

The system is shown to give very good results over a basic sample. Although a considerably bigger sample would be required for a proper validation of the system, the fact that different authors (and therefore different vocabulary and different use of poetic licence) are involved improves the significance of the result.

It is also important to note that although the system is based on linguistic rules for contemporary Spanish, the results are good even when applied to the work of Sixteenth Century poets – which speakers of modern Spanish sometimes find obscure. This shows that the system is not dependent on the specific database of vocabulary facts that it starts with, and it can reasonably be expected to cope with new words however alien to contemporary speakers (as long as they conform to the rules).

The system may be put to practical use as an autonomous analysis tool (as a first approximation to block analysis of texts), or as a pedagogical tool in teaching environments.

## References


1. Allen, J. : Natural Language Understanding, London, Benjamin/Cummings (1995).
2. Barquist, C.R., Shie, D.L.: Computer Analysis of Alliteration in Beowulf Using distinctive Feature Theory. Oxford Journal of Literary and Linguistic Computing, vol. 6, no. 4 (1991).
3. Beaudoin, V., Yvon,F.: The Metrometer: a Tool for Analysing French Verse. Oxford Journal of Literary and Linguistic Computing, vol. 11 no. 1 (1996).
4. Ousaka, Y., Yamazaki, M.: Automatic Analsyis of the canon of Middle Indo-Aryan by personal computer. Oxford Journal of Literary and Linguistic Computing, vol. 9, no. 2 (1994).
5. Pereira, F.C.N. and Warren, D.H.D.: Definite Clause Grammars for Language Analysis: a Survey of the Formalism and Comparison with Augmented Transition Networks. Artificial Intelligence, vol. 13, no. 3 (1980).
6. Pérez de la Cruz Molina, J.L., personal web page, http://www.lcc.uma.es/personal/cruz/sonetos/pral.html
7. Quilis, A.: Métrica española, Ariel, Barcelona (1985).
8. Real Academia Española (Comisión de Gramática): Esbozo de una nueva gramática de la lengua española, Espasa-Calpe, Madrid (1986).
9. Robey, D.: Scanning Dante's the Divine comedy: a computer Based approach. Oxford Journal of Literary and Linguistic Computing, vol. 8, no. 2 (1993).
10. Stratil, M., Oakley, R.J.: A Disputed Authorship Study of Two Plays Attributed to Tirso de Molina. Oxford Journal of Literary and Linguistic Computing, vol. 2, no. 3 (1987).
11. Saint-Dizier, P. Advanced Logic Programming for Language Processing, London, Academic Press (1994).
12. Wielemaker, Jan, http://swi.psy.uva.nl/usr/jan/SWI-Prolog.html
13. Wiliamsen, V.G. and Abraham, J.T., Association for Hispanic Classical Theater web page, ftp://listserv.ccit.arizona.edu/pub/listserv/comedia/poetic1.html