Lorenzo Cioni

An algorithm for the syllabification of written Italian

("V Simposio Internacional de Comunicación Social", Santiago de Cuba, January 22-24 1997)

Abstract

The aim of the present notes is to describe an algorithm for the syllabification of written Italian that has been developed at Laboratorio di Linguistica. The present algorithm is of deterministic type and it is based upon the use of recursion and of binary tree in order to detect the boundaries of the syllables within each word.

Introduction

The aim of these notes is to describe an algorithm for the syllabification of written Italian that has been developed by the author at Laboratorio di Linguistica.

The scope of such an algorithm is that of assigning the syllable boundaries to orthographic representations of written Italian.

The outcome of the algorithm is the production of the so-called canonical syllabification, i.e. the one either defined by the rules of grammar or on which the most of the people agree, of the words it receives as an input stream (see the implementation notes).

This means that the algorithm produces the hyphenated version of each word, where each hyphen marks the presence of a syllable boundary. At the present phase of design, the algorithm produces each hyphenated word by examining a set of rules in a very particular way, that we are going to state more precisely soon, but a morphological pre-processing module is under study.

Such a module should allow the algorithm to avoid as many errors as possible such those that, at the present, occur in presence of prefixes.

Indeed, for the time being, the algorithm, by applying the rules, behaves in a way such that, for instance, a word like *biologia* (biology, bio is the prefix meaning life) is correctly hyphenated as *bio-lo-gia* whereas a word with a similar structure such as *biossido* (dioxide, bi is the prefix meaning double) is hyphenated as *bios-si-do* and not according to the canonical form *bi-os-si-do*.

Last but not least, we note that we aim at obtaining the correct hyphenation of Italian word by using a minimal set of rules. It is obvious, indeed, that by defining an ordered set of pairs *word/hyphenated-string* and by executing for a certain word a binary search we obtain easily and quickly (in log n steps for n pairs) the desired syllabification.

We can define such an approach as the n-words/n-rules approach and moreover it is easily understood that if we have enough rules arranged in some way we are able to correctly hyphenate as many words as we need.

What we are trying to do is to achieve the same result with a fixed and small set of rules.

The syllabification rules

The algorithm, by means of an abstract computational structure (a binary decision tree), implements a set of syllabification rules that are shortly outlined in what follows. We note that C denotes a generic consonant whereas V represents one of the five vowels of written Italian (a, e, i, o, u).

The main accent is explicitly marked on the right of the corresponding vowel by using the character '.

We note that the presence of the accent is necessary in some cases to get the correct syllabification and that it does not alter the binary structure of the decision tree since in any case we have a choice consonant-vowel or consonant-accented vowel and so a double choice that allows us to define more correct syllabification rules.

For instance, if we have a non accented word like *baule* (trunk) from the rules we get the wrong syllabification *bau-le* instead of the correct one *ba-u-le* that the algorithm produces if we specify the position of the accent and so if the input string is *bau'le*. The same holds in case of *paura* and *pau'ra* (fear).

We now examine the rules. The first rule states that whenever there is a combination of the form CVCV the algorithm produces the syllabification CV-CV by inserting an hyphen in the correct position. This occurs independently from the presence of accented vowels, as in the other cases that follow.

Another rule states that the group $V_1C_1C_2V_2$ is syllabified as:

- V_1C_1-C_2V_2 if $C_1 = C_2$
- V_1-$C_1C_2V_2$ if C_2 = h
- V_1-$C_1C_2V_2$ if C_2 = l or r and $C_1 \neq$ l and r
- V_1s-C_2V_2 or V_1-sC_2V_2 if $C_2 \neq$ s
- V_1C_1-C_2V_2 in any other case. The open case must be solved depending on the nature of C_2.

The group $V_1C_1C_2C_3V_2$ is syllabified as V_1C_1-$C_2C_3V_2$ but if C_1= s we can get either V_1C_1-$C_2C_3V_2$ or V_1-$C_1C_2C_3V_2$ depending on C_2 and C_3. The present rule has some exceptions represented y prefixed words such as *transcaucasico* (trans-caucasian) that must be syllabified as *trans-cau-ca-si-co* and *postdatare* (post-date) as *post-da-ta-re*.

We think, however, that such kind of exceptions should be examined from a morphological perspective and therefore that an algorithm based on the examination of a word as a sequence of consonants and vowels can make such mistakes without being considered as completely wrong. To reduce the percentage of errors a morphological module, for instance, should mark the presence of prefixes so that the algorithm would act as in presence of two words to be individually syllabified.

At last, the group $V_1C_1C_2C_3C_4V_2$ is syllabified as $V_1C_1C_2$-$C_3C_4V_2$ though there are exceptions such as the word *substrato* (substratum) that must be syllabified as *sub-stra-to*.

The presence of group of vowels makes the syllabification harder since in Italian we can have up to six consecutive vowels (as in *cuoiaio* to be syllabified as *cuo-ia-io*, a person who sells or tans leather) though no more than three vowels can be contained in a single syllable. The algorithm, in presence of three or more consecutive vowels, tries therefore to analyse such groups of vowels looking, as a first resort, for triphthongs and then for diphthongs. Anyway we present in what follows some of the rules we have used in case of two consecutive vowels.

Vowels like a, e and o when are one after the other (like ae, ao and the like) never belong to the same syllable. In a group like *quV* the vowel *u* form always a diphthong with V, where V can be a, e, i or o. We have diphthongs in cases such as
- $V_1V'_2$ where either V_1 = i and V_2 = a, e, o or u or V_1 = u and V_2 = a, e, o or i
- $V_1'V_2$ where either V_2 = i and V_1 = a, e, o or u or V_2 = u and V_1 = a, e, o or i
- V_1V_2 where either V_1 = i and V_2 = a, e, o or u or V_1 = u and V_2 = a, e, o or i
- V_1V_2 where either V_2 = i and V_1 = a, e, o or u or V_2 = u and V_1 = a, e, o or i

Such rules present many exceptions such as *coincidenza* (coincidence) to e syllabified as *co-in-ci-den-za* and *semiasse* (axle shaft) to be syllabified as *se-mi-as-se* and many more but for these exceptions we refer to our past considerations.

As to groups of three or more consecutive vowels the algorithm scans a such group from left to right and examines the first three vowels it finds so to identify the position of the syllable boundary and to skip to the next character to be examined.

At this point there can be two cases:
- the three vowels form a triphthong so that they can be considered a single character or
- they do not form a triphthong and so the group of vowels must contain the syllable boundary.

In the first case the character that follows the triphthong can be either a vowel or a consonant. If it is a vowel, the algorithm puts the syllable boundary immediately on the left of the triphthong and starts another recursive cycle, if it is the case. If it is a consonant, the algorithm pre-loads the first two vowels on the output stream and another recursive cycle starts, with a vowel (followed by a consonant) as the current position. The algorithm, however, consider the definition of a group of three vowels as a triphthong as a hard problem so it uses a pre-load technique (see below) to handle such a situation.

In the second case there are the following possibilities: $V_1V_2V_3$ is either syllabified as V_1-V_2V_3 or as V_1V_2-V_3 where only one of the V_i can be accented. If the group does not contain any accented vowel the algorithm behaves according to the following rules:
- if V_1 = a, o or e then it produces V_1-V_2V_3;
- if V_1 = i and $V_2 \neq$ u then it produces V_1V_2-V_3;
- if V_1 = u and $V_2 \neq$ i then it produces V_1V_2-V_3;
- if V_1 = i and V_2 = u or V_1 = u and V_2 = i then it pre-loads V_1 and restarts with V_2 as current position. We have defined similar rules in cases in which there is an accented vowel. The first of the aforesaid rules accounts for *soia* (soybean) as *so-ia*, the second for *ghiaia* (gravel) as *ghia-ia* and so on.

The structure of the algorithm

The algorithm is of deterministic type and it is based upon the use of recursion and of binary tree in order to detect the boundaries of the syllables within each word.

The algorithm is composed by a set of modules that handle the input of words to be syllabified either from keyboard or from a file, the output of the stream of syllables either on the screen or on a file and implement the binary decision tree.

Such a tree is obviously not complete owing to the structure of the real Italian words and is binary since each node has two children (a consonant or vowel or a consonant and an accented vowel) or is itself a leaf. The aim of the algorithm is that of defining a path from the root to a leaf to which corresponds a syllabification rule that allows the definition of a syllable boundary.

The tree is dinamically created in this way. The starting point is a non-labelled root. The first character locates one of the two possible children and so does the second, the third and so on till the algorithm does not reach a leaf whose associated rule defines which is the inner node that represents a syllable boundary. The word, not always meaningful, is then split up in two parts: a syllable and a new word on which the algorithm is applied again whereas the syllable is put on the output stream.

The process goes on till the last syllable is reached. This characteristic defines the algorithm as recursive.

We note that each of the leaves defines a syllabification rule and represents the place where possible exceptions and indefinite cases are dealt with.

The morphological module should act as either a pre-processor (so to examine the original not syllabified word and, for instance, insert a marker that points out the presence of a prefix so that the algorithm can handle the incoming exception) or a post-processor so to examine the outcome and check if the syllabification gave rise to errors (this is very important during a debugging phase).

Now we pass at the analysis of some examples of pre-processing. Let us suppose to have a word like *postromantico* (post-romantic) and a word like *postribolo* (brothel). Without pre-processing, the first word would be syllabified as *po-stro-man-ti-co* whereas post represents in this case a prefix and so the correct syllabification is *post-ro-man-ti-co*. For the other word there is no problem since there is no prefix and the correct syllabification is *po-stri-bo-lo*.

In the first case a pre-processor would insert a marker so to produce post∗romantic so that no error would occur.

Some simple examples

Let us start with a simple example, the word calamita (magnet) that is very easy to syllabify but that will help us at introducing the way the algorithm behaves.

The algorithm scans the word from left to right and when reaches the second *a* finds itself on a leaf to which corresponds the rule CVCV → CV-CV and so it splits up the word in two parts: the syllable *ca* and the meaningless word *lamita*.

The algorithm is applied again on *lamita* so to produce *la* and *mita* and again on *mita* so to produce *mi* and *ta*. At this point the processing stops and the algorithm produces a stream of syllables with hyphens in the correct positions: *ca-la-mi-ta*.

From the above example, it is easily understood that the algorithm uses some simple techniques of look-ahead so to scan the word on the right of the current position and define a syllable boundary, moreover at each node on the tree it pre-load as many characters as it can on the output stream so to speed up the process.

We note that the knowledge of a certain number of characters on the right of the current position is necessary to locate a syllable boundary whereas whenever the algorithm reaches an inner node some of the characters on the left may be unnecessary and can be stripped and loaded on the output stream.

As another example we take the word *carro* (wagon). In order to define the first (and only) syllable boundary the algorithm scans the word to the end so to find the leaf with the rule $V_1C_1C2V_2 \rightarrow V_1C_1\text{-}C_2V_2$ since the two consonants are identical but when examines C_1 it can pre-load C_0 = c and when examines C_2 it can pre-load V_1= a

Likewise the algorithm syllabifies words such as *lento* (slow) to give *len-to* or *precipitevolissimevolmente* (very very fast, the longest word of written Italian) to give *pre-ci-pi-te-vo-lis-si-me-vol-men-te*.

Let us consider some harder example such as *cuoiaio*. The algorithm scans the word from the left till it finds a group of three vowels *uoi* that cannot form a triphthong so it uses the proper rule and splits the triple as *uo-i*, then it goes on with scanning and find three more vowels *iai* that once again cannot belong to the same syllable and so, with another rule, it splits

them as *ia-i*. At this point the algorithm finds two vowels that are known to form a diphthong and so can produce the syllabified version of the original word: cuo-*ia-io*.

Other examples are *ghiaia* (gravel) as *ghia-ia*, *aiuola* (flowerbed) as *a-iuo-la* and *troiaio* (pigsty) as *tro-ia-io*.

Implementation notes

The program that implements the algorithm has been written down with ANSI C on a workstation Digital Alpha 3000/300 with Digital Unix 3.2C. The use of ANSI C should ensure its full portability.

The program has been designed so to accept input data both from keyboard and from a text file and to produce the stream of syllabified words both on the screen and on a file.

Last but not least it can be composed with other commands according to the Unix philosophy so to be used in conjunction, for instance, with a module that analyses the morphology of the words to be syllabified and that is currently under development.

Acknowledgements

The author wishes to thank Professor Pier Marco Bertinetto and Doctor Maddalena Agonigi for many of the syllabification rules on which the algorithm is based.

Bibliography

Cioni, L. (1996), "RB-tree: un algoritmo per la sillabazione dell'italiano", proceedings of XXIV Convegno Nazionale dell'Associazione Italiana di Acustica, Trento, 12-14 June 1996:145-148.

Daelemans W & A. van den Bosh, "Generalization performance of backpropagation learning on a syllabification task", other data missing.

Laks B., (1995), "A connectionist account of French syllabification", Lingua 95:51-76