Finite-state technology in a verse-making tool

Manex Agirrezabal, Iñaki Alegria, Bertol Arrieta
University of the Basque Country (UPV/EHU)
maguirrezaba008@ikasle.ehu.es, i.alegria@ehu.es, bertol@ehu.es
Mans Hulden
Ikerbasque (Basque Science Foundation)
mhulden@email.arizona.edu

Abstract

This paper presents a set of tools designed to assist traditional Basque verse writers during the composition process. In this article we are going to focus on the parts that have been created using finite-state technology: this includes tools such as syllable counters, rhyme checkers and a rhyme search utility.

1 The BAD tool and the Basque singing tradition

The BAD tool is an assistant tool for verse-makers in the Basque bertso-lari tradition. This is a form of improvised verse composition and singing where participants are asked to produce impromptu compositions around themes which are given to them following one of many alternative verse formats. The variety of verse schemata that exist all impose fairly strict structural requirements on the composer. Verses in the bertso-lari tradition must consist of a specified number of lines, each with a fixed number of syllables. Also, strict rhyme patterns must be followed. The structural requirements are considered the most difficult element in the bertso-lari—however, well-trained bertsolaris can usually produce verses that fulfill the structural prerequisites in a very limited time.

The BAD tool presented here is mainly directed at those with less experience in the tradition such as students. One particular target group are the bertso-eskola-s (verse-making schools) that have been growing in popularity—these are schools found throughout the Basque Country that train young people in the art of bertso-lirtza.

The primary functionality of the tool is illustrated in figure 1 which shows the main view of the utility. The user is offered a form in which a verse can be written, after which the system checks the technical correctness of the poem. To perform this task, several finite state transducer-based modules, are used, some of them involving the metrics (syllable counter) of the verse, and others the rhyme (rhyme searcher and checker). The tool has support for 150 well known verse meters.

In the following sections, we will outline the technology used in each of the parts in the system.

2 Related work

Much of the existing technology for Basque morphology and phonology uses finite-state technology, including earlier work on rhyme patterns (Arrieta et al., 2001). In our work, we have used the Basque morphological description (Alegria et al., 1996) in the rhyme search module. Arrieta et al. (2001) develop a system where, among other things, users can search for words that rhyme with an introduced pattern. It is implemented in the formalism of two-level morphology (Koskenniemi, 1983) and compiled into finite-state transdukers.

We have used the open-source foma finite-state compiler to develop all the finite-state based parts of our tool. After compiling the transducers, we use them in our own application through the C/C++ API provided with foma.

3 Syllable counter

As mentioned, each line in a verse must contain a specified number of syllables. The syllable counter module that checks whether this is the case consists of a submodule that performs the syllabification itself as well as a module that yields variants produced by optional apocope and syncope effects. For the syllabification itself, we use the approach described in Hulden (2006), with some modifications to capture Basque phonology.

In our examples, FST expressions are written using foma syntax. For details, visit http://foma.googlecode.com.
3.1 Syllabification

Basque syllables can be modeled by assuming a maximum onset principle together with a sonority hierarchy where obstruents are the least sonorous element, followed in sonority by the liquids, the nasals and the glides. The syllable nuclei are always a single vowel (a,e,i,o,u) or a combination of a low vowel (a,e) and a high vowel (i,o,u) or a high vowel and another high vowel.

The syllabifier relies on a chain of composed replacement rules (Beesley and Karttunen, 2003) compiled into finite-state transducers. These definitions are shown in figure 2. The overall strategy is to first mark off the nuclei in a word by the rule MarkNuclei which takes advantage of a left-to-right longest replacement rule. This is to ensure that diphthongs do not get split into separate syllables by the subsequent syllabification process. Following this, syllables are marked off by the markSyll rule, which inserts periods after legitimate syllables. This rule takes advantage of the shortest-leftmost replacement strategy—in effect minimizing the coda and maximizing the size of the onset of a syllable to the extent permitted by the allowed onsets and codas, defined in Onset and Coda, respectively.

To illustrate this process, supposing that we are syllabifying the Basque word *intransitiboa*. The first step in the syllabification process is to mark the nuclei in the word, resulting in \{i\}ntr\{a\}nis\{i\}t\{i\}b\{o\}\{a\}. In the more complex syllabification step, the markSyll rule assures that the juncture ntr gets divided as n.tr because n.tr would produce a non-maximal onset, and i.ntr would in turn produce an illegal onset in the second syllable. The final syllabification, after markup removal by the Cleanup rule, is then *in.trans.i.ti.bo.a*. This process is illustrated in figure 3.

In *bertsolaritza*, Basque verse-makers follow this type of syllable counting in the majority if cases; however, there is some flexibility as regards the syllabification process. For example, suppose that the phrase *ta lehenengo urtian* needs to fit a line which must contain six syllables. If we count the syllables using the algorithm shown above, we receive a count of eight (*ta le.hen.en.go ur.ti.an*). However, in the word lehenengo we can identify the syncope pattern vowel-h-vowel, with the two vowels being identical. In such cases, we may simply replace the entire sequence by a single vowel (ehe → e). This is phonetically equivalent to shortening the ehe-sequence (for those dialects where the orthographical h is silent). With this modification, we can fit
the line in a 7 syllable structure. We can, however, further reduce the line to 6 syllables by a second type of process that merges the last syllable of one word with the first of the next one and then resyllabifying. Hence, ta lehenengo urtian, using the modifications explained above, could be reduced to ta.le.nen.gour.ti.an, which would fit the 6 syllable structure. This production of syllabification variants is shown in figure 4.

4 Finite-state technology for rhymes

4.1 Basque rhyme patterns and rules

Similar to the flexibility in syllabification, Basque rhyme schemes also allow for a certain amount of leeway that bertsolaris can take advantage of. The widely consulted rhyming dictionary Hiztegi Errimatua (Amuriza, 1981) contains documented a number of phonological alternations that are acceptable as off-rhymes: for example the stops p, t, and k are often interchangeable, as are some other phonological groups. Figure 5 illustrates the definitions for interchangeable phonemes when rhyming. The interchangeability is done as a prelude to rhyme checking, whereby phonemes in certain groups, such as p, are replaced by an abstract symbol denoting the group (e.g. PTK).

4.2 Rhyme checker

The rhyme checker itself in BAD was originally developed as a php-script, and then reimplemented as a purely finite-state system. In this section we will focus on the finite-state based one.

As the php version takes advantage of syllabification, the one developed with transducers does not. Instead, it relies on a series of replacement rules and the special _eq()_ operator available in foma. An implementation of this is given in figure 6. As input to the system, the two words to be checked are assumed to be provided one after the other, joined by a hyphen. Then, the system (by rule rhympat1) identifies the segments that do not participate in the rhyme and marks them off with “{“ and “}” symbols (e.g. landa-ganga → <{l}anda>-<{g}anga>).

The third rule (rhympat3) removes everything that is between “{“ and “}”, leaving us only with the segments relevant for the rhyming pattern (e.g. <{NMBDGR}a>-<{NMBDGR}a>). Subsequent to this rule, we apply the phonological grouping reductions mentioned above in section 4.1, producing, for example (<aNMBDGRa>-<aNMBDGRa>).

After this reduction, we use the _eq(X, L, R)_ operator in foma, which from a transducer X, filters out those words in the output where material between the specified delimiter symbols L and R are unequal. In our case, we use the < and > symbols as delimiters, yielding a final transducer that does not accept non-rhyming words.

4.3 Rhyme search

The BAD tool also includes a component for searching words that rhyme with a given word. It is developed in php and uses a finite-state component likewise developed with foma.

Similarly to the techniques previously described, it relies on extracting the segments relevant to the
define rhympat1 \[0: (* \ ? \ 0: *)\]
   \[
     \[
       ([V+ C+] \ (V) V \ | \ [(C) V V]) \ C+ \]
     \]
   \];
   # constraining V V C pattern
define rhympat2 \[\(? \ + V \ * \) V C\];
   # cleaning non-rhyme part
define rhympat3 \[\(? \ * \) \] -> 0;
define rhympat ryhmpat1 .o. rhympat2 .o.
   rhympat3;
   # rhyming pattern on each word
   # and phonological changes
define MarkPattern rhympat .o.
   phoRules .o. phoRules.i .o.
   dictionary ;

define MarkTwoPatterns
0:%< MarkPattern 0:%> %-
0:%< MarkPattern 0:%> ;
define Verify _eq(MarkTwoPatterns, %<, %>)
regex Verify .o. Clean;

Figure 6: Rhyme checking using foma.

rhyme, after which phonological rules are applied
(as in 4.1) to yield phonetically related forms. For
example, introducing the pattern era, the system
returns four phonetically similar forms era, eda, ega,
and eba. Then, these responses are fed to a trans-
ducer that returns a list of words with the same end-
ings. To this end, we take advantage of a finite-state
morphological description of Basque (Alegria et al.,
1996).

As this transducer returns a set of words which
may be very comprehensive—including words not
commonly used, or very long compounds—we then
apply a frequency-based filter to reduce the set of
possible rhymes. To construct the filter, we used
a newspaper corpus, (Egunkaria²) and extracted the
frequencies of each word form. Using the frequency
counts, we defined a transducer that returns a word’s
frequency, using which we can extract only the n-
most frequent candidates for rhymes. The system
also offers the possibility to limit the number of syl-
lables that desired rhyming words may contain. The
syllable filtering system and the frequency limiting
parts have been developed in php. Figure 7 shows
the principle of the rhyme search’s finite-state com-
ponent.

5 Evaluation
As we had available to us a rhyme checker written
in php before implementing the finite-state version,
²http://berria.info

it allowed for a comparison of the application speed
of each. We ran an experiment introducing 250,000
pairs of words to the two rhyme checkers and mea-
sured the time each system needed to reply. The
FST-based checker was roughly 25 times faster than
the one developed in php.

It is also important to mention that these tools
are going to be evaluated in an academic environ-
ment. As that evaluation has not been done yet, we
made another evaluation in our NLP group in or-
der to detect errors in terms of syllabification and
rhyme quality. The general feeling of the experiment
was that the BAD tool works well, but we had some
efficiency problems when many people worked to-
gether. To face this problem some tools are being
implemented as a server.

6 Discussion & Future work

Once the main tools of the BAD have been de-
veloped, we intend to focus on two different lines
of development. The first one is to extend to flexibil-
ity of rhyme checking. There are as of yet patterns
which are acceptable as rhymes to bertsolaris that
the system does not yet recognize. For example,
the words filma and errima will not be accepted by
the current system, as the two rhymes ilma and ima
are deemed to be incompatible. In reality, these two
words are acceptable as rhymes by bertsolaris, as
the l is not very phonetically prominent. However,
adding flexibility also involves controlling for over-
generation in rhymes. Other reduction patterns not
currently covered by the system include phenomena
such as synaloepha—omission of vowels at word
boundaries when one word ends and the next one
begins with a vowel.

Also, we intend to include a catalogue of melodies
in the system. These are traditional melodies that
usually go along with a specific meter. Some 3,000
melodies are catalogued (Dorronsoro, 1995). We are
also using the components described in this article in
another project whose aim is to construct a robot ca-
pable to find, generate and sing verses automatically.
Acknowledgments

This research has been partially funded by the Spanish Ministry of Education and Science (OpenMT-2, TIN2009-14675-C03) and partially funded by the Basque Government (Research Groups, IT344-10).

We would like to acknowledge Aitzol Astigarraga for his help in the development of this project. He has been instrumental in our work, and we intend to continue working with him. Also we must mention the Association of Friends of Bertsolaritzo, whose verse corpora has been used to test and develop these tools and to develop new ones.

References

Alegria, I., Artola, X., Sarasola, K., and Urkia, M. (1996). Automatic morphological analysis of Basque. *Literary and Linguistic Computing*, 11(4):193–203.

Amuriza, X. (1981). *Hiztegi errimatua [Rhyme Dictionary]*. Alfabetatze Euskalduntze Koordinakundeak.

Arrieta, B., Alegria, I., and Arregi, X. (2001). An assistant tool for verse-making in Basque based on two-level morphology. *Literary and linguistic computing*, 16(1):29–43.

Beesley, K. R. and Karttunen, L. (2003). *Finite state morphology*. CSLI.

Dorronsoro, J. (1995). *Bertso doinutegia [Verse melodies repository]*. Euskal Herriko Bertsolari Elkartea.

Hulden, M. (2006). Finite-state syllabification. *Finite-State Methods and Natural Language Processing*, pages 86–96.

Koskenniemi, K. (1983). Two-level morphology: A general computational model for word-form production and generation. *Publications of the Department of General Linguistics, University of Helsinki. Helsinki: University of Helsinki.*