Porting Basque Morphological Grammars to *foma*, an Open-Source Tool

Iñaki Alegria[1], Izaskun Etxeberria[1], Mans Hulden[2], and Montserrat Maritxalar[1]

[1] IXA group
University of the Basque Country
i.alegria@ehu.es
[2] University of Arizona
Department of Linguistics
mhulden@email.arizona.edu

Abstract. Basque is a morphologically rich language, of which several finite-state morphological descriptions have been constructed, primarily using the Xerox/PARC finite-state tools. In this paper we describe the process of porting a previous description of Basque morphology to *foma*, an open-source finite-state toolkit compatible with Xerox tools, provide a comparison of the two tools, and contrast the development of a two-level grammar with parallel alternation rules and a sequential grammar developed by composing individual replacement rules.

1 Introduction

In this paper we describe some aspects of the design of a Basque morphological processing system built with finite-state techniques. Since we have recently ported a Basque morphological grammar from the Xerox formalism to the open-source *foma* toolkit (Hulden, 2009), we shall focus on aspects and experiences highlighted by this process. In light of this migration to an open source toolkit, which forced us to convert an older two-level description of the morphology to an equivalent sequential rule-based one, we shall also contrast some practical design aspects of two-level grammars (Koskenniemi, 1983; Karttunen et al., 1987) with so-called sequential replacement rule grammars (Kaplan and Kay, 1994; Beesley and Karttunen, 2003).

2 Basque morphology

Basque is an agglutinative language with a rich morphology. Earlier descriptions of Basque morphological analysis with finite-state techniques include Aldezabal et al. (1994); Alegria et al. (1996). A later implementation using the Xerox/PARC compilers is described in Alegria et al. (2002).

From the point of view of designing a complete description of Basque morphology, the most prominent and features of the language include:

– Basque morphology is very rich. The determiner, the number and the declension case morphemes are appended to the last element of the noun phrase and always occur in this order.
– Basque nouns belong to a single declension; the 15 case markers are invariant.
– Functions that prepositions normally fulfill are realized by case suffixes inside word-forms. Basque offers the possibility of generating a large number of inflected word-forms. From a single noun entry, a minimum of 135 inflected forms can be generated. While 77 of these are simple combinations of number, determiners, and case markings (and not capable of further inflection), the rest (58) include one of the two possible genitive markers (possessive and locative) and they can create new declension cases. With this in mind, Basque can be called an agglutinative language.
– Basque has ergative case, which marks the subjects of transitive verbs. Linguistic theories and nomenclature about this phenomenon are varying: some use the terminology 'ergative language,' and others 'non-accusative.'
– The verb provides all the grammatical and agreement information about the subject, the two possible objects, as well as tense and aspect-related information, etc.

3 Earlier work and current migration to open source

The earlier two-level descriptions referred to above have been used in different applications, for example in a spell checker and corrector named *Xuxen*,[3] which is widely used among Basque speakers. In addition to the standard morphological description, additional finite-state transducers that extend the basic grammar and is useable for other tasks have been built (Alegria et al., 2002). These are:

– A standard analyzer that includes normalized morphemes and standard phonological rules.
– An enhanced analyzer for the analysis and normalization of linguistic variants (modeling dialectal usage and competence errors). This is a critical tool for languages like Basque in which standardization is recent and dialectal lexical and phonological variants are common and widespread.
– A 'guesser,' or analyzer for words which have no lemma listing in the lexicon. For this, the standard transducer is simplified and the lexical entries in the open categories (nouns, proper names, adjectives, verbs, etc.) are removed. These entries, which constitute the vast majority of the lexicon, are substituted by a general automaton that accepts any combination of characters in the orthography, subject to some phonological constraints.

Migration to open source technology of the various finite-state-based grammars for Basque that have been developed is still an ongoing process. Much of what has been accomplished so far has been done in a semi-automatic way; some of this work is described in e.g. Alegria et al. (2008).

[3] `http://www.xuxen.com`

For open-source spell checking and correction applications, we have used *hunspell* (Nemeth et al., 2004), since *hunspell* is directly supported in the later versions of OpenOffice and Mozilla/Firefox. However, *hunspell* is limited in its descriptive power. It is, for instance, not possible to express phonological alternations independently of the lexicon, which results in that the conversion from the original transducer-based descriptions is not at all straightforward.

For this reason, we have also experimented with the internal support *foma* provides for spell checking and spelling correction applications based on finite automata, and plan to incorporate this in the array of finite-state based applications for Basque already available.

4 Porting Basque grammars to *foma*

The original source description of the Basque (Alegria et al., 1996) was compiled with the the Xerox toolkit (Beesley and Karttunen, 2003), using a number of the formalisms that it supports. The lexicon specification language, *lexc*, was used for modeling the lexicon and constraining the morphotactics, and the two-level grammar language, *twolc*, was used for constructing a transducer that models the phonological and orthographical alternations in Basque.

As *foma* provides no method for compiling two-level rules, the first step in the migration consisted of translating the rules formerly compiled with *twolc* to the form of replacement rules supported by foma—largely identical to the rules supported by *xfst*. The original description of the Basque morphology was built prior to the introduction of publicly available tools to manipulate and compile sequentially composed replacement rules, and thus followed the two-level formalism.

We were also aware of the fact that there has been a recent shift in preference toward favoring sequential replacement rules rather than two-level rules in the design of morphological parsers (Beesley and Karttunen, 2003). This was part of the motivation for our decision to explore the replacement rule paradigm when reimplementing our grammars with open-source tools.

Porting grammars from one formalism to another one is an interesting problem for which there are few resources to be found in the literature. This is especially true in the case of sequentially composed replacement rules and the two-level formalism. Grammars written in either of the two are of course compilable into finite-state transducers and are therefore equivalent in a sense, which in turn should motivate a comparison between the two from the point of view of the grammar developer. The most prominent aspects of such a comparison should include differences in grammar size and grammar complexity, ease and clarity of rules and rule interactions as well as ease of debugging rules.

In table 1 we illustrate a simple case of debugging two-level grammars as opposed to debugging sequential rule grammars. A large part of the work involved in developing a two-level grammar consists of avoiding rule conflicts. Here, we have two rules in (a) that constrain the realization of **k**: the first rule dictates that **k** should elide before consonants while the second rule states that it shall become

voiced following a consonant. The first rule is exemplified by underlying-surface pair forms such as **horiektan**:**horietan**, while the second handles alternations such as **eginko**:**egingo**. Unfortunately, as stated, the two rules conflict as there is no unequivocal statement about what a **k** should correspond to in case it occurs with a consonant on both sides. Additional information needs to be provided for the proper compilation of the grammar, often in the form restating the two rules with more complex and restricted contextual parts. By contrast, the ordered rules in (b), where the elision rule is assumed to apply before the voicing rule, give the correct forms without a need for additional specification about what occurs in cases where the contexts overlap. In many cases this kind of an approach allows one to design an ordered rewrite rule grammar in a more isolated way such that each rule targets one phonological generalization only—such as elision or devoicing—without mixing information from two arguably separate phenomena in both rules.

```
(a)
k:0 <=> _ Cons
k:g <=> Cons _

(b)
k -> 0 || _ Cons
k -> g || Cons _
```

Table 1. *Simple rule conflict cases which often resolve automatically when implemented as ordered rules.*

In table 2 we provide an example of the differences between both descriptions. The example rules illustrate a number of phenomena. In Basque, the prefix **ber** can precede verbal roots. Before a vowel, the final character (**r**) of this prefix is doubled, but before a consonant the **ber** prefix changes into **bir**. In addition to this, if the first character of the root is **h**, this **h** disappears. If the prefix is expressed as **beR** (indicating a hard **r**) the chain **beR+egin** (a lexical expression) generates **berregin**, while **beR+gai+tu** generates **birgaitu**, and **beR+has+i** changes into **berrasi**.

In our experience, the sequential rules provided conceptual simplicity and were easier to debug than two-level rules. This was largely because one can avoid complex interactions in rules and their contexts of application, which in general one cannot when designing a two-level grammar. However, the order of the rules needs to be designed carefully, which is also a nontrivial problem.

4.1 Conversion procedure

In converting the rules, we decided to follow a simple strategy. First, rules with limited contextual requirements (single-symbol simple contexts) were trans-

```
# parallel rules
(a)
R:r <=> _
+:0 <=> _
h:0 <=>  # b e R:r 0:r +:0 _ Vowel
0:r <=>  R:r _ +:0 (h:0) Vowel
e:i <=>  # b _ R +:0 Cons

# sequential rules
(b)
h -> 0  || # b e R + _ Vowel
0 -> r  || R _ + Vowel
e -> i  || # b _ R + Cons
R -> r
+ -> 0
```

Table 2. *Simplified comparison between parallel and sequential rules in Basque morphology.*

formed into sequential rules. Also, rules that were very specific in their conditioning environments (such as those interacting with only one particular prefix) were converted.

Following this step, rules making use of information not available at the surface level—abstract morphophonemes, diacritic symbols, or complex morphological information—were resolved. These rules fell naturally into the early parts of the chain of compositions since diacritics and abstract symbols could be deleted and would no longer be available for subsequent rules. In a way, the strategy was to move step by step from the lexical level of abstract diacritic symbols and morphophonemes toward the actual surface words.

The most complex rules—those with multiple contexts and interactions with other rules—were placed at the end. These include rules such as *e*-epenthesis, *r*-epenthesis, and enforcing a final *a* in the lemmas. Within this last block of rules, the internal ordering of the rules was also much more important and required more care in the design process.

During each of the above conversion steps, examples from the previously used two-level system were used for testing.

After converting the two-level description to context-dependent replacement rules, porting the description to *foma* was straightforward because the lexc description is fully compatible. Two experiments were carried out: the first morphology-oriented with the whole lexicon containing the full morphological description of all the morphemes (including category, case, tense, person, etc.) and the second spelling-oriented with only the lexical description of the morphemes (surface level and indispensable marks for the rules). In both descriptions the same phonological rules were used.

After this, more thorough testing and debugging was performed. We applied the two morphological analyzers (the original two-level one and the new one) to

a corpus of 80,000 different word forms to find discrepancies by running the unix tool *diff* on the results.

In the end, we managed to reduce the discrepancies to only one analysis of one word, which was then found to be a mistake in the lexicon grammar. The lexicon had omitted a hard *R* for place names (all final R's in place names are hard in Basque).

Some auxiliary applications, such as a spelling corrector module, had already been developed in the *xfst* replacement rule paradigm after the original two-level morphological grammar, and thus no conversion was required for constructing these transducers in *foma* as it compiled the original *xfst* rules to identical transducers as the Xerox tools.

4.2 Compatibility and efficiency

In porting the Xerox-based grammatical descriptions to *foma*, we noted very few discrepancies in compatibility. *Foma* has no separate program for extensive debugging of lexicon specifications in the *lexc* format, but is able to import *lexc* descriptions through the main interface. Our earlier lexicons were thus imported without changes, and compiled to identical transducers as with Xerox's *lexc*: 2,321,693 states and 2,466,071 arcs for our more detailed lexicon, and 63,998 states and 136,857 arcs for the less detailed one. Compiling the complete system which requires compiling the lexicon (89,663 entries), compiling 77 replacement rules, and composing all of these separate transducers under *lexc* and *xfst* took 28.96s,[4] and 15.39s with *foma*,[5] on a 2.93GHz Pentium 4 computer with 2Gb memory running Linux. *Foma*, however required far more temporary memory for compilation—a peak usage of 788.2 Mb which occurred while compiling the larger lexicon—while *lexc* used a maximum 161.2 Mb for the same task.

5 Auxiliary applications: spelling correction

One of the approaches to capture certain kinds of spelling errors for Basque has been to identify predictable and often occurring types of misspellings and suggest corrections for these. Because of the fairly recent standardization of the orthography, and because of the amount of dialectal variation in Basque, cognitive errors of this type are not uncommon, and it is important for a spelling corrector to identify these errors accurately.

One of the components of the Basque morphological system is a transducer that encodes a set of 24 hand-coded string perturbation rules reflecting errors commonly found in Basque writing, such as confusing a **z** and and **c**, an **x** and a **z**, and so on. This transducer is composed with an automaton encoding the possible surface forms of the morphology, providing a markup of possible misspellings, which is then composed with a filter that accepts only those misspellings that

[4] xfst-2.10.29 (cfsm 2.12.29), and lexc-3.7.9 (2.12.29)

[5] version 0.9.6alpha.

match a word in the lexicon (see Fig. 1). This error correction mechanism was originally modeled with *xfst* replacement rules, and so was directly portable to *foma*, and usable as such.

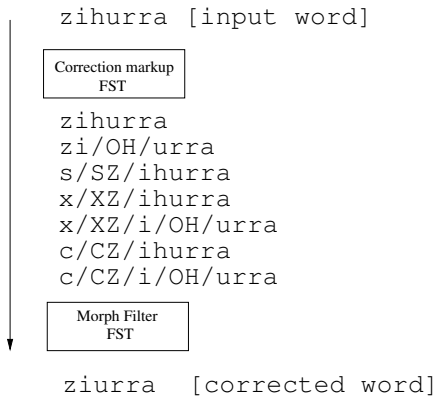

Fig. 1. *An illustration of the functioning for the part of the spelling corrector that recognizes typical orthographical errors. A correction markup filter nondeterministically changes the input to a number of possible candidate errors, after which the morphological filter retains only those that are actual words.*

5.1 Internal support for spelling correction in foma

A recent addition to the *foma* toolkit and API is an algorithm for quickly finding *approximate* matches to an input word and an automaton. The default metric of distance is the Levenshtein distance, i.e. minimum edit distance where character substitutions, deletions, and insertion all have a cost of 1 unit. However, *foma* also provides the possibility of defining separate costs for different types of string perturbation operations.

This feature is convenient in that one automatically has a spelling corrector available, given a morphological analyzer. In our experiments with foma, we have simply extracted the range (lower side in the terminology of Beesley and Karttunen (2003)) of our morphological analyzer transducer, producing a cyclic finite-state automaton which encodes all the words that have a legitimate morphological analysis in the original system, directly producing a spelling corrector application.

We have also done some preliminary experiments in modeling the cognitive errors described above, by specifying an additional weight confusion matrix to *foma's* minimum edit distance algorithm, giving each of the 24 string perturbation operations in our earlier separate finite-state correction grammar a low cost (1 unit), and other operations the cost (2 units), yielding an spelling corrector

application very similar to the earlier hand-built one, although much easier to construct.

An example of the interactive spelling correction is given in Fig. 2. In the future we hope to automatically derive the weights for different edit operations for increased accuracy, and integrate the spell checker and spelling corrector using *foma's* C language API to other applications, such as OpenOffice and Mozilla/Firefox.

```
foma[0]: regex MORPHO.l;
80512 states, 346119 arcs, Cyclic.
foma[1]: apply med
Using confusion matrix [Euskara]
apply med> zihurra

ziurra     Cost[f]: 1
zimurra    Cost[f]: 2
zigurra    Cost[f]: 2
zuhurra    Cost[f]: 2
bihurra    Cost[f]: 2
```

Fig. 2. *Applying the minimum edit distance finder of* foma *produces spelling correction suggestions like those of our hand coded rules seen in Fig. 1, given a similar specification in the form of a confusion matrix.*

6 Conclusion

We have described a segment of an ongoing process to migrate Basque natural language processing tools to open-source technology—that of porting a wide-coverage morphological description of the language to compile with the *foma* toolkit. This entailed rewriting a formerly two-level grammar into sequential replacement rules. We also hope to address the porting of other applications—such as a spelling corrector—with *foma*.

7 Acknowledgements

The first author has been partially funded by the Spanish Ministry of Education and Science (OpenMT: Open Source Machine Translation using hybrid methods, TIN2006-15307-C0301). We wish to thank the anonymous reviewers for helpful comments and suggestions.

Bibliography

Aldezabal, I., Alegria, I., Artola, X., Díaz de Ilarraza, A., Ezeiza, N., Gojenola, K., and Urkia, M. (1994). EUSLEM: Un lematizador/etiquetador de textos en Euskara. In *Actas del X. Congreso de la SEPLN Córdoba*.

Alegria, I., Aranzabe, M., Ezeiza, A., Ezeiza, N., and Urizar, R. (2002). Using finite state technology in natural language processing of Basque. *LNCS: Implementation and Application of Automata*, 2494.

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

Alegria, I., Ceberio, K., Ezeiza, N., Soroa, A., and Hernandez, G. (2008). Spelling correction: from two-level morphology to open source. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odjik, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Beesley, K. and Karttunen, L. (2003). *Finite-State Morphology*. CSLI, Stanford.

Hulden, M. (2009). Foma: a finite-state compiler and library. In *EACL 2009 Proceedings*, pages 29–32.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Karttunen, L., Koskenniemi, K., and Kaplan, R. M. (1987). A compiler for two-level phonological rules. In Dalrymple, M., Kaplan, R., Karttunen, L., Koskenniemi, K., Shaio, S., and Wescoat, M., editors, *Tools for Morphological Analysis*. CSLI, Palo Alto, CA.

Koskenniemi, K. (1983). *Two-level morphology: A general computational model for word-form recognition and production*. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki.

Nemeth, L., Tron, V., Halacsy, P., Kornai, A., Rung, A., and Szakadat, I. (2004). Leveraging the open source ispell codebase for minority language analysis. *Proceedings of SALTMIL*.