BUILDING FRIENDLY ARCHITECTURES FOR TAGGING

M. Vilares Ferro
J. Graña Gil
A. Pan Bermúdez

Computer Science Department
University of Corunna
Campus de Elviña s/n
15071 A Coruña
Spain
E-mail: {vilares, grana, pan}@dc.fi.udc.es.

Abstract

We introduce a development environment for the generation of taggers. Our proposal is based on the notion of finite automaton. In relation to previous approaches, our system separates the execution strategy from the implementation of the tagging interpreter, which is guided by the system itself. That facilitates the maintenance at the time that assures the robustness of the taggers so generated. Empirical tests prove the adequation of our approach to deal with languages whose morphology is non-trivial, in particular in relation with the sharing of structures and computations during tagging.

Key Words: Tagging, User Interface, Maintenance.

1 Introduction

The last years, has seen a renewal of interest in the consideration of the finite automaton (FA) model to the design of taggers in natural language processing (NLP), even in the case of part-of-speech tagging [4]. This is due to the speed and compactness of the representations. In effect, the growing complexity of current tagging systems make that the space required for implementations is an important issue in commercial applications, together with computational efficiency. This is specially the case for inflectional languages with a great variety of morphological processes, such as Spanish.

However, the FA model has not the flexibility of grammar-oriented approaches [2]. In particular, the maintenance of FA-based systems is not always trivial. So, most of authors propose updating protocols based on the simple re-compilation from the set of grammatical rules constituting the descriptive formalism for tagging [1, 3]. This technique is more friendly that the direct modification of the FA serving as kernel for the system, but the process should also assure the sharing of linguistically related paths in the automaton, in order to permit both the implementation of efficient error recovery and debugger tools. In relation with this, classic determinization and minimization techniques for FAs do not guarantee sharing in basis to this requirement. This implies a lost of declarative power and make the study of segmentation phenomena difficult, which is often of interest for language specialist. At this point, our goal is to reconcile declarative power, and computational efficiency and safety.

Section 2 of this work introduces Spanish as running example. Section 3 describes the system at work, introducing most relevant available functionalities. In section 4 we show some interesting practical tests. Finally, section 5 is a conclusion about the work presented.

2 Spanish as running example

To illustrate our work, we consider the case of Spanish, an inflectional language, as a running example throughout this paper. Spanish shows a great variety of morphological processes,

This work was partially supported by the Autonomous Government of Galicia under project XUGA20403B95.

particularly non-concatenative ones, which make it adequate for our purposes. We summarize some of the outstanding problems we have to deal with:

1. A highly complex conjugation paradigm, with nine simple tenses and nine compound tenses, both on the six different persons. If we add the Present Imperative with two forms, Infinitive, Compound Infinitive, Gerund, Compound Gerund, and Participle with four forms, then 118 inflected forms are possible for each verb.
2. Irregularities in both verb stems and endings. Very common verbs, such as `hacer` (*to do*), have up to seven different stems: `hac-er`, `hag-o`, `hic-e`, `haré`, `hiz-o`, `haz`, `hech-o`. Approximately 30% of Spanish verbs are irregular. We have implemented 39 groups of irregular verbs.
3. Verbal forms with enclitic pronouns at the end. This can produce changes in the stem due to the presence of accents: `da` (*give*), `dame` (*give me*), `dámelo` (*give it to me*). We have implemented forms even with three enclitic pronouns, like `tráetemelo` (*bring it for you and me*). Here, the analysis has to segment the word and return four tokens.
4. There exist some highly irregular verbs that can be handled only by including many of their forms directly in the lexicon. This is, for example, the case of `ir` (*to go*) and `ser` (*to be*).
5. Gaps in some verbs paradigms, in which some forms are missing or simply not used. For instance, meteorological verbs such as `nevar` (*to snow*) are conjugated only in third singular person.
6. Duplicate past participles, like `impreso` and `imprimido` (*printed*). In such cases, the tagger has to treat both.
7. A highly complex gender inflection, with words with only one gender as `hombre` (*man*) and `mujer` (*woman*), and words with the same form for both genders as `azul` (*blue*). In relation to words with separate forms for masculine and feminine, we have a lot of models: `autor`, `autora` (*author*); `jefe`, `jefa` (*boss*); `poeta`, `poetisa` (*poet*); `rey`, `reina` (*king*) or `actor`, `actriz` (*actor*). We have implemented 20 variation groups for gender.
8. An also highly complex number inflection, with words presenting only the singular form, as `estrés` (*stress*), and others where only the plural form is correct, as `matemáticas` (*mathematics*). The construction of different forms does not involve as many variants as it is the case of the gender, but we can also consider a certain number of models: `rojo`, `rojos` (*red*); `luz`, `luces` (*light*); `lord`, `lores` (*lord*) or `frac`, `fraques` (*dress coat*). We have implemented 10 variation groups for number.

This complexity suggests the necessity to interface the tagging process in order to verify easily the properties demanded, as well as to facilitate the maintenance. To deal with, we propose the fields for spanish tokens, together with their possible values, i.e. the tag set represented in table 1. As an example, let's consider the word `sobre`. This word has three possible meanings in Spanish: preposition (*on*, *upon*, *over*, *about*), noun (*envelope*) and verb (*to exceed*, *to be unnecessary*). When it is a verb, there are two possible values for the person: first and third. So, the output of the morphological analyzer should contain four taggings:

```
Word: "sobre"
      Preposition, "sobre"
      Common Noun, Masculine, Singular, "sobre"
      Verb, Subjunctive Present, First, Singular, "sobrar"
      Verb, Subjunctive Present, Third, Singular, "sobrar"
```

3 The system at work

On the basis of a classic compilation process from a set of morphological rules, our goal is to make transparent for the user the generation of these rules. In this manner, the user can turn his attention around the linguistic information, leasing to the system most of the problems

Field	Values	
Word	*The citation form present in the input text.*	
Lemma	*The canonical form of the word.*	
Category	Adjective	*With no type.*
	Adverb	Exclamatory, modifier, nuclear, nuclear & modifier *and* relative.
	Conjunction	Coordinate, coordinate & subordinate, subordinate *and* que.
	Determiner	Alterizer, article, cardinal, combinable quantifier, comparative, demonstrative, interrogative, non combinable quantifier, ordinal, possessive, relative *and* totalizer.
	Idiom	*With no type.*
	Preposition	*With no type.*
	Pronoun	Alterizer, atonic, cardinal, combinable quantifier, comparative, demonstrative, enclitic atonic personal, enclitic tonic personal, interrogative, non combinable, quantifier, ordinal, possessive, relative, tonic *and* totalizer.
	Punctuation Mark	*With no type.*
	Noun	Common *and* proper.
	To be	*With no type.*
	To have	*With no type.*
	Unknown	*With no type.*
	Verb	*With no type.*
Gender	Masculine, feminine, masculine & feminine *and* neutral.	
Number	Singular, plural *and* singular & plural.	
Mode	Indicative, subjunctive, imperative, infinitive, gerund *and* participle.	
Verbal tense	Present, imperfect in "ra", imperfect in "se", simple perfect, future, conditional, perfect past, pluperfect in "ra", pluperfect in "se", anterior past, perfect future *and* perfect conditional.	
Person	First, second *and* third.	
Determination	Definite *and* indefinite.	
Case	Nominative, accusative, dative, accusative & dative *and* case preposition.	
Comparison	Equality, superiority & inferiority *and* non comparative.	

Table 1: Tag set

imposed by the programming task. From the computational point of view, this implies a gain of safety, as well as a more friendly user-interface.

3.1 Tracing facilities

Due to the complexity and great size of current systems, it is not possible, in practice, to correct errors and even detect them without help. A crucial feature of our proposal is to establish valid mechanisms to make it possible to observe the behavior during tagging.

The tracing facility lets the user obtain, from the global system, the set of states visited by the tagger for a given word and check its correctness. Furthermore, we call these sequences of states *paths*. Path recovery can be small enough to verify simple properties related to the correction of the recognition units for inflection or derivation.

For example, in the case of the word `ténselo` (*hold it for him, her or them* or *tauten it*), the tagger could return a first tagging formed by three tokens:

```
Word: "t'en"
      Verb, Imperative Present, Second, Singular, 2 Enclitic Pronouns, "tener"
Word: "se"
      Enclitic Pronoun, Atonic, Feminine & Masculine, Third, Singular & Plural, "'el"
Word: "lo"
      Enclitic Pronoun, Atonic, Masculine, Third, Singular, Accusative, "'el"
```

and a second one formed by two tokens:

```
Word: "t'ense"
      Verb, Imperative Present, Second, Singular, 1 Enclitic Pronoun, "tensar"
Word: "lo"
      Enclitic Pronoun, Atonic, Masculine, Third, Singular, Accusative, "'el"
```

It seems strange that this word can correspond to two verbs which are so different, the former with one enclitic pronoun, and the latter with two.

However, by passing the word through the tagger with the debugging option, we obtain the following two paths:

```
0 st1                                    0 st1
-- t --> 1 st431                         -- t --> 1 st431
-- ACCENT --> 3 st1626                   -- ACCENT --> 3 st1626
-- e --> 5 st2874                        -- e --> 5 st2874
-- n --> 8 CLIT_IMP_SING2_st251          -- n --> 7 st4404
-- s --> 12 st1121                       -- s --> 11 CLIT_IMP_CONJ1_st285
-- e --> 18 CLIT_IMP_SING22_st253        -- e --> 15 CLIT_IMP_CORT12_st279
-- l --> 23 CLITGEN2_st331               -- l --> 21 CLITGEN1_st329
-- o --> 26 st1318                       -- o --> 24 st1314
```

both equivalents to the reduced FA in Fig. 1. Briefly, in the trace, each character in the token is associated to a transition over an state in the FA, denoted by expressions of the form `*st*`. When some of these states have a well defined meaning from a linguistic point of view, they are denoted by a phrase indicating their functionality. In the example, the system has used the following set of keywords: `CLIT` (by enclitic pronoun), `IMP` (by imperative), `SING` (by singular), `GEN` (by gender), `CONJ` (by conjugation), and `CORT` (by courtesy). In this case, the trace produced by the query let us check that the tagging is correct, and also validate that the involved treatment units are working correctly.

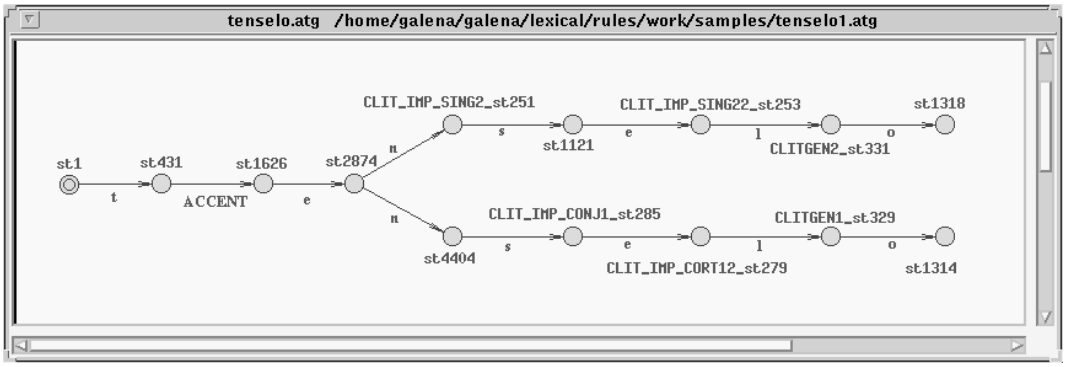

Figure 1: Reduced FA for the query `ténselo`

3.2 Updating facilities

Our goal now is to provide a mechanism to verify the correctness of incremental developing of taggers. This may be helpful to minimize the set of errors present in the new releases of the system by assuring the compatibility with previous ones. The verification method we want to advocate for is based on reductions of a global FA. These reductions collapse states of the automaton to reach sizes reasonable enough to be outprinted and well understood. So, we can center our attention only around relevant information that can be easily manipulated.

To illustrate this point, we assume a new version at the tagger have been implemented. Despite of the increasing of power, the update has changed a correct pattern recognition in the former release. Due to this, the word `ténselo` is not recognized by the new release. Our goal is to detect these kinds of bugs in compile time.

A way to do that is to compare patterns. So, we can automatically take out them from the old tagger[1] and verify whether they are present or not, in the new model. When this process deals with the case of the pattern corresponding to `ténselo`, which we shall reference as `pattern`, the verifier produces the following output:

```
@ obseqd (new-model, pattern);;
error outgoing labels:
no states in automaton-2 with same outgoing labels than states in it1
number of iteration(s): 1
False : Bool
```

[1] That we assume correct.

which indicates that `it1` contains the list of problematic states. We can now see them:

```
@ show it1;;
{5} : List of Integer
```

From here, we deduce that the fifth state in the path represents somewhere an erroneous option in the new model. By showing the transitions in the pattern explored, we can visualize this state as follows:

```
@ explore (new-model);;
State 0
st1
-- t -->  1: st329
# ? 5
State 5
st759
-- i -->  6: CLIT_IMP_SING22_st458
```

which locates the bug in the state `st759` for the transition labeled `i`, as it is also shown in Fig. 2. This puts into evidence that we have implemented a pattern recognition for `ténsilo`, a word with no meaning in Spanish.

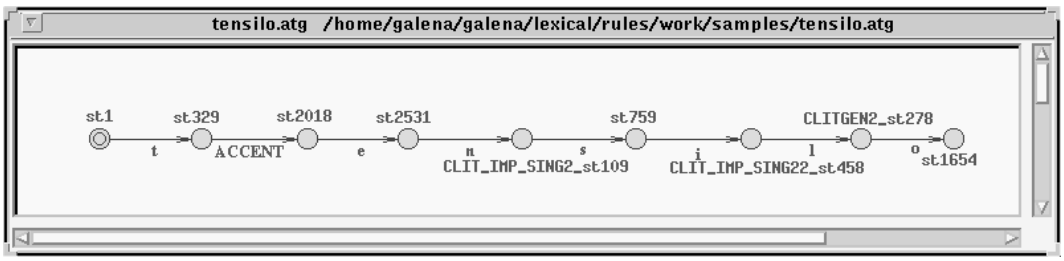

Figure 2: Reduced FA for the query `ténsilo`

4 Experimental results

To illustrate performance we give both information on the current version of our analyzer, and information on the evolution process we expect. As physical support for tests we have taken a *Sun SPARCStation 10*.

Actually, we are able to reconize and tag the most common 4000 lemmas of Spanish. The corresponding automaton has more than 32000 states and the average speed is 1400 words tagged per second. Number of states in the automaton is high, but it will grow slowly because main inflectional phenomena are already implemented. That is, the only task that remains undone is the introduction of more and more stems, and it is checked that this process yields an average increase of only 2 states for each new lemma. However, unfortunately, compilation time can be very high. This is the cost you must pay when what you really want is high performance. The only solution for this obstacle is to implement incremental building processes for automata. This point is part of our future work.

In order to have a better idea of future sizes and times, we built several analyzers from a great amount of patterns. These patterns were generated with a random process, but keeping the same level of ambiguity as in Spanish words, and the results are in figure 3. These tests show that the proposed architecture for the tagger presents a linear time and space complexity.

5 Conclusion

The design of tagging systems should respond to constraints of efficiency, safety and maintenance that we have considered from a practical point of view.

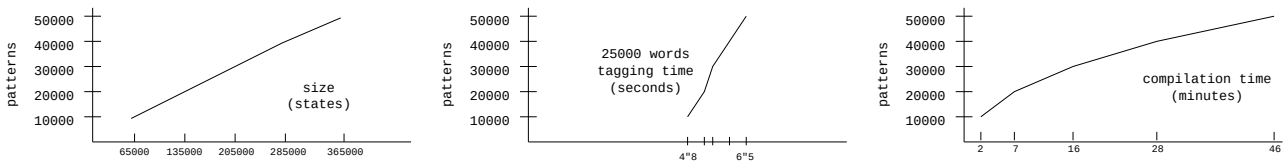

Figure 3: Experimental results

The choice of the FA model as operational formalism assures the computational efficiency. Safety is guaranteed by the separation between this operational kernel, and the high-level descriptive formalism.

However, one of the major services of every lexicon ought to be to provide as much information as possible about errors, because of the complexity of actual implementations, and the natural evolution suffered by these kinds of systems. The goal is to minimize the time dedicated to debug the system. So, our discussion has a practical sense.

As an additional advantage, our proposal is based on the capability to reduce general FA's. This implies that it is independent of the particular implementation of the system, which solves the problem of portability and reduces costs.

The described work is not closed. It represents only a first approach to the problem of verification in tagging, but preliminary results seem to be promising and the operational formalism well adapted to deal with more complex problems as to consider unrestricted error recovery algorithms, and development of disambiguation techniques.

References

[1] K. Koskenniemi. Compilation of automata from morphological two-level rules. In *Proc. of the* 5^{th} *Scandinavian Conference of Computational Linguistics*, pages 143–149, Helsinki, Finland, 1985.

[2] G. Ritchie. On the generative power of two-level morphological rules. In *European Chapter of the ACL*, pages 51–57, Manchester, 1989.

[3] G. Ritchie, D. Pulman, Stephen, A.W. Black, and G.J Russell. *Computational Morphology.* The MIT Press, Cambridge, Massachusetts, U.S.A., 1991.

[4] E. Roche and Y. Schabes. Deterministic part-of-speech tagging with finite-state transducers. *Computational Linguistics*, 21(2):227–253, 1995.