Finite-State Morphology and Formal Verification[†]

Manuel Vilares Ferro, Jorge Graña Gil
Computer Science Department, University of Corunna,
Campus de Elviña s/n, 15071 La Coruña, Spain
{vilares,grana}@dc.fi.udc.es

Pilar Alvariño Alvariño
Spanish Philology Department, University of Santiago de Compostela,
Burgo de las Naciones s/n, 15705 Santiago de Compostela, Spain
fepili@usc.es

(*Received 27 December 1996*)

Abstract

This paper describes an environment for the generation of non-deterministic taggers, currently used for the development of a Spanish lexicon. In relation to previous approaches, our system includes the use of verification tools in order to assure the robustness of the generated taggers. A wide variety of user defined criteria can be applied for checking the exact properties of the system.

1 Introduction

There are few things more frustating than spending a great deal of time debugging errors in an application. The existence of a tool which allows verification of the properties of a morphological analyzer leads to safer systems at the same time that modeling effort is saved. This kind of facility is specially useful in the implementation of taggers for inflectional languages with non-trivial morphology.

Most practical studies on verification tools are related with the concept of finite automaton (FA) (Boudol *et al.* 1990; Madelaine *et al.* 1989; Roy 1990). This represents an adequate starting point for our work, since most authors consider FA's as the most efficient and general way to deal with the problem of tagging computationally. However, the use of FA's poses some problems in relation with the maintenance of the system in a continuously evolving context. In effect, they represent the operational interpretation of a set of morphological rules in order to decide how to tag each word encountered. This implies a loss of declarative power and makes the study of segmentation phenomena difficult, which is of interest when

† This work was partially supported by the Autonomous Government of Galicia under projects XUGA20403B95 and XUGA10505B96.

some kind of unexpected behavior is detected. So, we are interested in verification methods combining modularity in the description of the system and flexibility in the verified properties, such as AUTO (Madelaine *et al.* 1989). AUTO computes small-scale models of finite transition systems, as is the case for FA's. These reduced systems are quotiens of that under study, up to generalized bisimulation (Milner 1980; Park 1981). The parameter of the reduction is a user-defined abstraction criterion (Boudol 1985), which embodies a particular point of view of a system. One is therefore able to build a variety of quotiens of a same system, which are small enough to verify particular properties.

2 Spanish as a guideline 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, particularly non-concatenative ones, which make it adequate for our purposes. The most representative of this type of features can be found in verbs. 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, all of which have 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 38 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 even implemented forms with three enclitic pronouns, like `tráetemelo` (*bring it for you and me*). Here, the analysis has to segment the word and return four tokens.

This complexity suggests the need to interface the morphological analysis with a formal proof system which allows us to verify easily the properties demanded, as well as to recover the system from unexpected states.

Taking up the introduction of the tagger, we propose the fields for token, together with their possible values, i.e. the tag set, shown in Table 1.

3 Verification by reduction

The verification method we want to advocate in this paper is based on reductions of a global FA. These collapse states of the automaton to reach sizes reasonable enough to be outprinted and easily understood. In this way, our attention is centered only around relevant information.

Table 1. *Tag set*

Field	Values	
Word	*The citation form present in the input text.*	
Lemma	*The canonical form of the word.*	
Category	Adjective	*With no type.*
	Adverb	Exclamative, modifier, nuclear, relative, interrogative *and* nuclear & modifier.
	Article	*With no type.*
	Conjunction	Coordinate *and* subordinate.
	Demonstrative	*With no type.*
	Indefinite	*With no type.*
	Interjection	*With no type.*
	Interrogative	*With no type.*
	Numeral	Cardinal, ordinal, partitive *and* multiple.
	Peripheral	Foreign word, formula, symbol, abbreviation, acronym *and* other.
	Preposition	*With no type.*
	Personal Pronoun	Tonic, proclitic atonic *and* enclitic atonic.
	Possessive	*With no type.*
	Punctuation Mark	Dot, comma, colon, semicolon, dash, quotes, open/close question mark, open/close exclamation mark, open/close parenthesis *and* dots.
	Relative	*With no type.*
	Substantive	Common *and* proper.
	Verb	*With no type.*
Subtype	Determiner, non-determiner *and* both.	
Gender	Masculine, feminine, both, neutral *and* non-applicable.	
Number	Singular, plural, both *and* non-applicable.	
Degree	Comparative *and* non-applicable.	
Person	First, second, third,first & third *and* non-applicable.	
Case	Nominative, accusative, dative, accusative & dative, prepositional case *and* nominative & prepositional case.	
Verbal tense	Present, preterite, copreterite, copreterite in “se”, future, postpreterite, antepresent, antepreterite, antecopreterite, antecopreterite in “se”, antefuture *and* non-applicable.	
Mode	Indicative, subjunctive, imperative, infinitive, compound infinitive, gerund, compound gerund *and* participle.	

3.1 Tracing facilities

A crucial feature of our proposal is to establish valid mechanisms to make it possible to observe the behavior during tagging (Madelaine *et al.* 1989). 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.

From the global FA, the verification process lets the user obtain the path, that is, the set of states visited by the tagger, for a given word, and check its correctness. For example, in the case of the word `ténselo` (*hold it for him, her or them* or *tauten it*), a morphological analyzer without additional contextual information could return two possible taggings:

```
=> ["ten", (V2spm02), Verb, second, sing, present, imperative,
    gender non-applicable, 2 enclitic pronouns, "tener"]
=> ["se", (Re3yyy), Personal Pronoun enclitic atonic, third,
    sing & plur, accusative & dative, masc & fem, "'el"]
=> ["lo", (Re3sam), Personal Pronoun enclitic atonic, third, sing,
    accusative, masc, "'el"]
```

and also

```
=> ["t'ense", (V2spm01), Verb, second, sing, present, imperative,
    gender non-applicable, 1 enclitic pronoun, "tensar"]
=> ["lo", (Re3sam), Personal Pronoun enclitic atonic, third, sing,
    accusative, masc, "'el"]
```

It seems strange that this word can correspond to two verbs which are so different, the first with two enclitic pronouns, and the second with one. However, by passing the word through the analyzer with the debugging option, we obtain the following 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 equivalent to the reduced FA in Fig. 1. The partial view produced by this query allows us to check that the tagging is correct, and also to validate that the treatment units involved are working correctly.

The goal is not only to guarantee the correctness of any of the current treatment units for the inflections, but also any new ones introduced by the user. In this way, we can minimize the set of errors present in the final application.

3.2 Improving maintenance

To illustrate this aspect, we assume that we have implemented a new version of the morphological analyzer. This latest version should increase the power of the previous system, but the updating has unconsciously introduced an erroneous pattern. Due

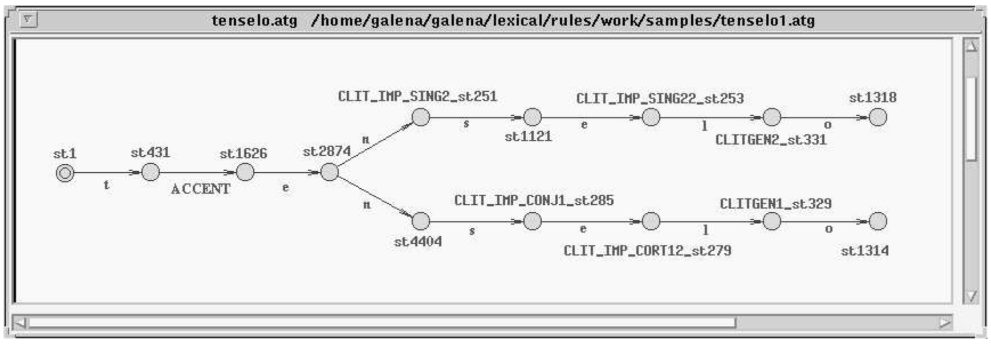

Fig. 1. Reduced FA for the query `ténselo`

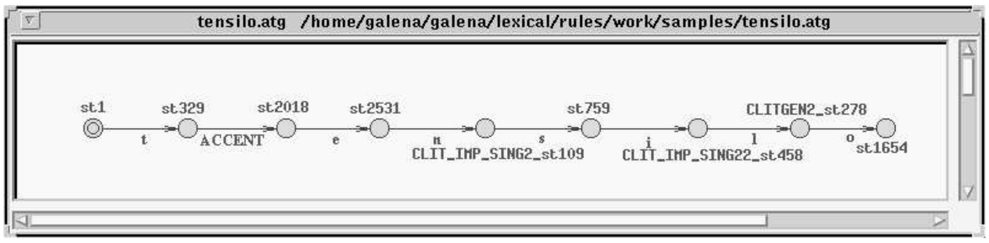

Fig. 2. Reduced FA for the query `tènsilo`

to this, the word `ténselo` is not recognized by the new release. Our goal is to detect these kinds of bugs in compile time, which may be helpful for the incremental developing of taggers.

One way of doing this is to compare patterns. So, we can automatically take them out from the old tagger, which we assume to be correct, 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`, that 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
```

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

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

from which we deduce that the fifth state in the path, the state `st759`, represents an erroneous option in the new model. We can make this evident by showing the transitions in the pattern explored:

```
@ 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 transition labeled `i`, as is shown in Fig. 2. This indicates that we have implemented a pattern recognition for `ténsilo`, a word with no meaning in Spanish.

3.3 Devising a guesser

Let's assume, for example, that we want to introduce a new verb into the lexicon. As we have seen, the verbal paradigm in Spanish is not trivial, and the user may well be unaware of the group to which the verb belongs. At this point, it would be desirable to integrate in the system a facility to guide the user in such a task. Using AUTO, we can explore all the paths between two different states, as well as accede to relevant states. So, it is easy for the user to start from the initial state corresponding to a given verbal model, and recover the labels in the paths corresponding to the verbal endings. Concatenating the stem of the verb to those endings, we automatically obtain the set of all the verbal forms against which the user can contrast the requested information.

The following example proves that the verb `amar` (*to love*) is regular, and belongs to the first conjugation in Spanish. In the same manner, we shall prove that the verb `jugar` (*to play*) is not regular, since the form `jugé` is not correct. We shall show the verification process step by step. First, we load the FA containing the morphological analyzer, which we call `lexer`:

```
@ set aut = include-fc2-automaton "lexer";;
aut : Automaton
```

We recover `V1`, the initial state for the first conjugation, in order to obtain a reduction of the global FA that we call `conj1`:

```
@ set conj1 =
     subautomaton (aut , car (structure (aut, "V1")));;
conj1 : Automaton
```

Finally, we capture the labels in the paths from `V1` to all the final states in the first conjugation, and catenate them to the stem `am` of the verb.

```
@ catenate-stem ("am", get-endings-list (conj1));;
{ ... ; am'e; amaste; am'o; ...} : List of String
```

where `catenate-stem` and `get-endings-list` are functions implemented by using the resources of AUTO, and "..." has been used to abbreviate the output for this paper. In this case, all verbal forms have been correctly produced. As a consequence, `amar` is a regular verb of the first conjugation.

Following a similar process, we catenate the stem `jug` of `jugar` to the preceding set of endings.

```
@ catenate-stem ("jug", get-endings-list(conj1));;
{ ... ; jug'e; jugaste; jug'o; ...} : List of String
```

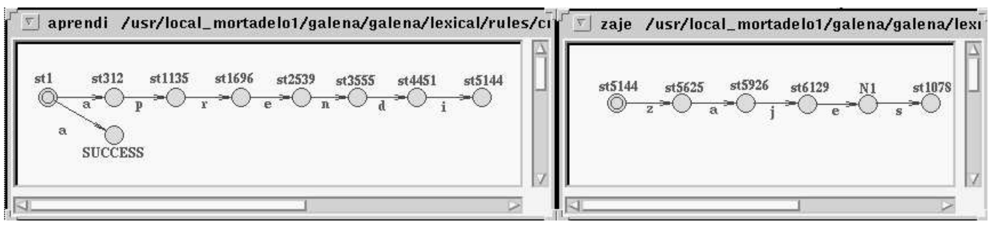
aprendi /usr/local_mortadelo1/galena/galena/lexical/rules/ci
st1
st312
st1135
st1696
st2539
st3555
st4451
st5144
a
p
r
e
n
d
i
a
SUCCESS
zaje /usr/local_mortadelo1/galena/galena/lexi
st5144
st5625
st5926
st6129
N1
st1078
z
a
j
e
s

Fig. 3. Error recovery for the word `aprendizaje`

In this case, the form `jug'e` is not correct, which implies that `jugar` does not belong to this model.

Although the preceding example could be qualified as naive, it illustrates a simple approach to implement an automatic generator of derived forms, applicable to any kind of words.

3.4 Automatic error recovery

In the same manner, we can consider a similar reasoning to solve another aspect of the question. This is, for example, the case of error recovery during the analysis or transmission of a text, when some characters are lost or ill-formed.

A verification tool for finite transition systems is necessarily capable of manipulating paths in FA's. This allows us to model a simple protocol to deal with situations where the lack of information obstructs the normal development of the morphological recognition process.

An error in the morphological analysis of a text often involves the skipping of large portions of it for subsequent treatment[1]. This means that any additional errors that were skipped over will go undetected until future analysis of the same text, which increases the time required for debugging. So, one of the major services of every lexicon ought to be to provide as much information as possible about errors in order to minimize the time dedicated to debugging the system. This means completely recovering, and resuming, the recognition process at the point of each error, so as not to miss detecting any subsequent errors.

The advantages of a system having a good error recovery method are obvious. The disadvantages are that it may be either very costly to develop or very inefficient to use. We introduce an automatic error recovery method which is based on the capability of AUTO to reduce FA's, and therefore to manipulate paths in the finite state machine associated with a morphological analyzer. This implies that our proposal is independent of the set of morphological rules which form basis of the analyzer, thus solving the first of the two problems.

To illustrate the technique, let's consider `aprendicaje`, an incorrect word in Spanish. The minimal equally plausible correct version of the above string is the word `aprendizaje` (*learning*). We shall detail explicitly the recovery process, although in practice it would be performed automatically. Let's assume that the

[1] for example, in the case of parsing.

tagging process is currently located in the word `anprendicaje`. Once the error has been detected, the system calls the verifier with the erroneous string. This looks for the minimal unit involved in the recognition from the criterion `anprendicaje` representing the string. We call the resulting reduced automaton `reduced`. Formally, this is recovered by the function `context` as follows:

```
@ set reduced = context (aut, aprendicaje);;
reduced : Automaton
```

We can now locate precisely the state in the automaton where the error was detected. To do this, the system first recovers the final states in the reduced model by using the function `dead`

```
@ dead aprendicaje;;
{8} : List of Integer
```

which implies that the only final state is numbered 8. From this point, we recover the path for `anprendicaje` in the reduced model leading to this state. We call this path `road`, and the system applies the function `path` to achieve this goal

```
@ set road = path (aprendicaje, 8);;
road : Path
```

It is now possible to get a trace of the transitions followed by the tagging process for the original erroneous string by applying the function `show` as follows:

```
@ show road;;
0 st1
-- a --> 1 st312
-- p --> 3 st1135
-- r --> 4 st1696
-- e --> 5 st2539
-- n --> 6 st3555
-- d --> 7 st4451
-- i --> 8 st5144
: Path
```

which indicates that the error state is `st5144`. The system is now in position to begin with the automatic error correction process from the skeleton represented by the left-hand-side in Fig. 3. To do this, the algorithm extracts from the global FA the automaton beginning at the error state using the function `subautomaton`. We call the resulting automaton `continue`, which is shown in the right-hand-side of Fig. 3.

```
@ set continue = subautomaton (aut, car (number (aut, "st5144")));;
continue : Automaton
```

Finally, the system automatically catenates the portion of the erroneous string that we have identified as correct, `aprendi`, with the endings corresponding to paths in `continue`. As result, we obtain the set of possible corrections proposed by the verifier:

```
@ catenate-stem ("aprendi", get-endings-list (continue));;
{aprendizaje, aprendizajes} : List of String
```

It is clear that, finding a *bridge transition* when an error makes the automaton stop the recognition process is a strategy that solves only the *change* error hypothesis. The error could of course be an *insertion* or a *deletion*, even in another position. Therefore, this method represents a first approach to the problem of automatic error recovery.

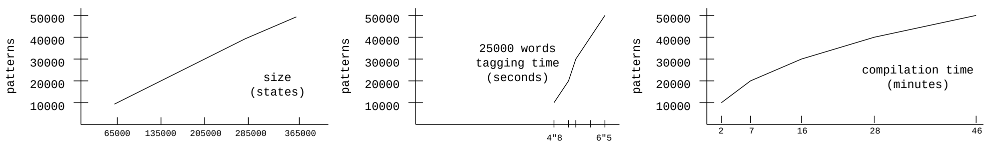

Fig. 4. Experimental results

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 Hyper Sparc Station.*

At present, we are able to reconize and tag the most common 7000 lemmas of Spanish. The corresponding automaton has more than 35000 states and the average speed is 1400 words tagged per second. The number of states in the automaton is high, but it will grow slowly because the main inflectional phenomena are already implemented. That is, the only task that remains to be performed is the introduction of more and more stems, and it has been proved that this process yields an average increase of only 2 states for each new lemma. However, unfortunately, compilation time can be very high, which is the price that must be paid when the desired result is high performance. The only way to overcome this obstacle is to implement incremental building processes for automata. This feature is part of our future work.

In order to achieve a better understanding of future sizes and times, we built several analyzers from a large quantity 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 shown in Fig. 4. 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. The choice of the FA model as operational formalism assures computational efficiency. Safety is guaranteed by the separation which exists 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 debugging the system. For this reason, our discussion has a practical sense.

The approach presented in this paper allows verification of morphological analyzers by computing reductions. These reductions are parametrized by criteria chosen by the user, reflecting the specific aspect to be worked on, for which the correctness of the recognition procedure must be verified.

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 work described above 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 such as considering unrestricted error recovery algorithms, and the development of disambiguation techniques.

References

Boudol, G. 1985. Logics and Models for Concurrent Systems. *Notes on Algebraic Calculi of Processes.* Springer-Verlag.

Boudol, G., Roy, V., de Simone, R. and Vergamini, D. 1990. Process calculi, from theory to practice: Verification tools. In *Automatic Verification Methods for Finite State Systems, Lecture Notes in Computer Science* **407**:1–10, Springer-Verlag.

Madelaine E. and Vergamini, D. 1989. AUTO: A verification tool for distributed systems using reduction of finite automata networks. In *Proc. FORTE'89 Conference*, Vancouver.

Milner, R. 1980. A calculus of communicating systems. *Lecture Notes in Computer Science* **92**, Springer-Verlag.

Park, D. 1981. Concurrency and automata on finite sequences. *Lecture Notes in Computer Science* **104**: 167–183, Springer-Verlag.

Roy, V. 1990. AUTOGRAPH: Un outil de visualisation pour les calculs de processus. *PhD thesis, Université de Nice-Sophia Antipolis, France.*