VERIFICATION OF MORPHOLOGICAL ANALYZERS

Manuel Vilares Ferro
vilares@dc.fi.udc.es

Jorge Graña Gil
grana@dc.fi.udc.es

Fidel Cacheda Seijo
Computer Science Department
University of Corunna, Campus de Elviña s/n
15071 A Coruña, Spain
cacheda@dc.fi.udc.es

Abstract

This paper is a reflection on the use of verification tools in morphological analyzers. The growing complexity of taggers poses serious problems for the software engineer in order to verify the correctness of the tagging procedure. In this context, our goal is to integrate this kind of facility to check for the exact properties of the system.

Key Words: Morphological Analysis, Verification Tools, Safety.

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 for verify the properties of a morphological analyzer, leads to more safe 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 of practical studies on verification tools are related with the concept of finite automaton (FA) [2, 3, 6]. This represents an adequate starting point for our work, since most of authors bet on 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 context that probably continuously evolves. In effect, they represent the operational interpretation of a set of morphological rules to decide how to tag each word encountered. This implies a lost of declarative power and make the study of segmentation phenomena difficult, which is of interest when 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 [3]. AUTO computes small-scale models of finite transition systems, as it is the case for FA's. These reduced systems are quotiens of the one under study, up to generalized bisimulation [4, 5]. The parameter of the reduction is a user-defined abstraction criterion [1], which embodies a particular point of view on a system. So, one is able to build a variety of quotiens of a same system, which are small enough to verify particular properties.

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

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. Most representative features are 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, 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.

This complexity suggests the necessity to interface the morphological analysis with a formal proof system which allows us to verify easily the properties demanded, as well as to recovery 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, showed in Table 1.

Field	Values	
Word		
Lemma		
Category	Adjective	
	Adverb	Exclamatory, modifier, nuclear, relative nuclear & modifier.
	Conjunction	Coordinate & subordinate, subordinate que, coordinate.
	Determiner	Alterizer, article, cardinal, demonstrative, comparative, possesive, interrogative, ordinal, non combinable quantifier, relative totalizer, combinable quantifier.
	Idiom	
	Preposition	
	Pronoun	Alterizer, atonic,
		combinable quantifier,
		comparative, relative, enclitic atonic personal, enclitic tonic personal, interrogative, ordinal, non combinable, cardinal, quantifier, possessive, tonic totalizer demonstrative.
	Punctuation Mark	
	Noun	Common proper.
	To be	
	To have	
	Unknown	
	Verb	
Gender	Masculine, feminine, masculine & feminine neutral.	
Number	Singular, plural singular & plural.	
Mode	Indicative, subjunctive, imperative, infinitive, gerund participle.	
Verbal tense	Present, imperfect in "ra", imperfect in "se", simple perfect, future, conditional, perfect past, pluperfect in "ra", pluperfect in "se", perfect future perfect conditional, anterior past.	
Person	First, second third.	
Determination	Definite indefinite.	
Case	Nominative, accusative, dative, accusative & dative case preposition.	
Comparison	Equality, superiority & inferiority non comparative,	

Table 1: Tag set

The tagger proposed shows a linear time complexity. As a reference, taking as physical support a *Sun SPARCstation 10*, the middle speed has been of 2700 words taggered per second.

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 well understood. So, we can center our attention only around relevant information.

Tracing facilities

A crucial feature of our proposal is to establish valid mechanisms to make it possible to observe the behavior during tagging [3]. Due to complexity and great size of current systems, it is not possible, in practice, to correct errors and even detect them without help. So, for example, the FA implementing our running example for Spanish has more than 9000 states.

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:

```
  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 also

```
  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, first with one enclitic pronoun, and after with two. However, by passing the word through the analyzer with the debugging option, we obtain the following paths:

```
0 st1
-- t --> 1 st431
-- ACCENT --> 3 st1626
-- e --> 5 st2874
-- n --> 8 CLIT_IMP_SING2_st251
-- s --> 12 st1121
-- e --> 18 CLIT_IMP_SING22_st253
-- l --> 23 CLITGEN2_st331
-- o --> 26 st1318
```

and

```
0 st1
-- t --> 1 st431
-- ACCENT --> 3 st1626
-- e --> 5 st2874
-- n --> 7 st4404
-- s --> 11 CLIT_IMP_CONJ1_st285
-- e --> 15 CLIT_IMP_CORT12_st279
-- l --> 21 CLITGEN1_st329
-- o --> 24 st1314
```

both equivalents to the reduced FA in Fig. 1. The partial view produced by this query let us check that the tagging is correct, and also validate that the treatment units involved are working correctly.

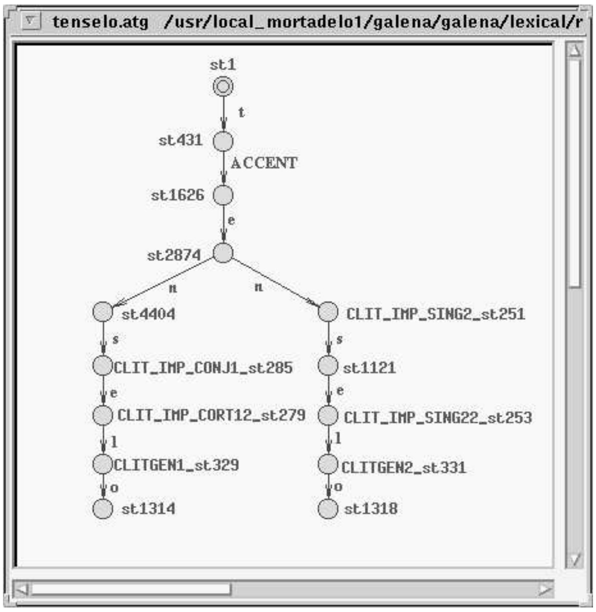

Figure 1: Reduced FA for the query `ténselo`

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

Improving maintenance

To illustrate this aspect, we assume that we have implemented a new version of the morphological analyzer. This last one should increase the power of the previous system, but the updating has unconsciously introduced an erroneous pattern. 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, which may be of helpful for the incremental developing of taggers.

A way to do that is to compare patterns. So, we can take automatically 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`, that

[1]That we assume correct.

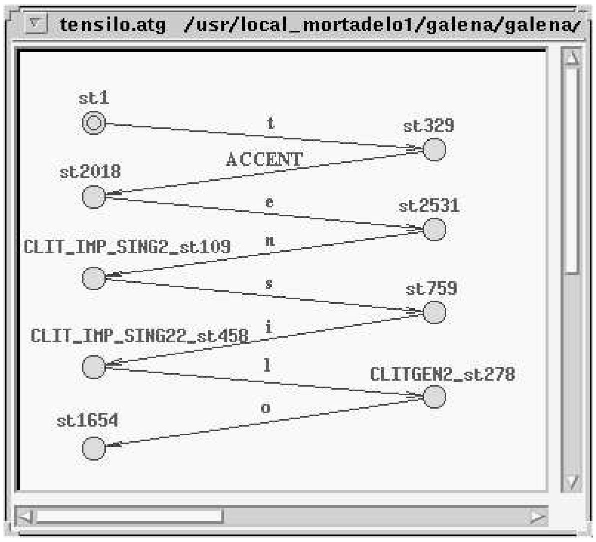

Figure 2: Reduced FA for the query `ténsilo`

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 to see them:

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

from which we deduce that the fifth state in the path, the state `st759`, represents somewhere 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 it is shown in Fig. 2. This puts into evidence that we have implemented a pattern recognition for `ténsilo`, a word with no meaning in Spanish.

TRACING A GUESSER

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

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 it would not be strange that the user ignores 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 to 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 on 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, that we have baptized `lexgalena`:

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

We recover `V1`, the initial state for the first conjugation, in order to obtain a reduction of the glogal FA that we call `conj1`[2]:

```
@ 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 resulted correct. 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 precedent set of endings.

```
@ catenate-stem("JUG",get-endings-list(conj1));;
{ ... ; JUG'E; JUGASTE; JUG'O; ...} : List of String
```

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 derivated forms, applicable to any kinds of words. In the same manner, we can consider a similar reasoning to solve another aspects of the question. This is, for example, the case of error recovery during the transmission of a text, when some characters are lost or ill-formed. At this point, the system must assure completely recovering, and resuming the recognition process at the point of each error, so as not to miss detecting any subsequent errors. In effect, often an error in the morphological analysis of a text involves the skipping of large portions of it for subsequent treatment. This means that any additional errors that were skipped over will go undetected until future analysis of the same text, which surcharges the time required for debugging.

[2] By first conjugation.

CONCLUSION

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 aspect he wants to put in evidence, for which the correctness of the recognition procedure must be verified.

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.

References

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

[2] G. Boudol, V. Roy, R. de Simone, and D. Vergamini. Process calculi, from theory to practice: Verification tools. In *Automatic Verification Methods for Finite State Systems, LNCS 407*, pages 1–10. Springer-Verlag, 1990.

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

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

[5] D. Park. Concurrency and automata on finite sequences. *Lecture Notes in Computer Science*, 104:167–183, 1981.

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