Syntactic Error Detection and Correction in Date Expressions using Finite-State Transducers

Arantza Díaz de Ilarraza, Koldo Gojenola, Maite Oronoz, Maialen Otaegi and Iñaki Alegria

Department of Computer Languages and Systems
University of the Basque Country
P.O. box 649, E-20080 Donostia
{jipdisaa, jibgogak, jiporanm, acpalloi}@si.ehu.es

July 31, 2007

Abstract

This paper presents a system for the detection and correction of syntactic errors. It combines a robust morphosyntactic analyser and two groups of finite-state transducers specified using the Xerox Finite State Tool (XFST). One of the groups is used for the description of syntactic error patterns while the second one is used for the correction of the detected errors. The system has been tested on a corpus of real texts, containing both correct and incorrect sentences, with good results.

1 Introduction

In this work we present a research carried out to detect and correct syntactic errors in date expressions using finite-state transducers (FSTs). Finite-state constraints, encoded in the form of automata and transducers, have been applied to the linguistic analysis. We have used XFST for the definition of complex linguistic patterns over morphosyntactic information.

We chose to deal with date expressions due to the fact that they contain morphologically and syntactically rich enough phenomena where several types of errors can be found. They can be considered representative of the errors that are detectable by examining local syntactic contexts. Besides, and based on copy-editors' and language teachers' opinion, date expressions in Basque are one of the most frequent source of errors in both, language learners and native speakers.

Basque is an agglutinative language, and as a consequence, most of the elements appearing in date expressions (year numbers, months and days) must inflect, i.e. the corresponding article and case morphemes must be attached to them. Moreover, each different date format requires that the elements involved appear in fixed combinations

of, for example, cases (see table 1), so different types of agreement are needed. These require a previous linguistic analysis before applying the FSTs for detection and correction.

Finite-state techniques have been used to create most of the NLP tools for linguistic analysis for Basque (Aduriz and Díaz de Ilarraza 2003). Following a previous experience in the construction of a robust spelling checker based on FSTs, XUXEN[1] (Agirre, Alegria, Arregi, Artola, Díaz de Ilarraza, Urkia, Maritxalar, and Sarasola 1992), we have faced the task of syntactic error detection and correction in the same way.

The remainder of this paper is organised as follows. Section 2 reviews several related works. After commenting on the linguistic resources we have used in section 3, we give a general overview of the system in section 4. Section 5 describes the error detection process, while section 6 presents the correction procedure. Then, we evaluate the system in section 7, to conclude in section 8.

2 Related work

The problem of syntactic error detection and correction has been addressed since the early years of natural language processing. For the treatment of the significant amount of errors (typographic, phonetic, cognitive and grammatical) that result in valid words (Weischedel and Sondheimer 1983; Heidorn, k. Jensen, Miller, Byrd, and Chodorow 1982) different techniques have been proposed:

- Grammar-based techniques. These systems use the results of a parser as input. Techniques that use chart-based methods (Min and Wilson 1998) or the relaxation of syntactic constraints (Douglas and Dale 1992) could be categorised into this group. In general, these methods share the problem of incomplete coverage of the underlying grammars. Manually written grammars are often unable to analyse the full range of sentences in running text. Moreover, when dealing with ill-formed sentences, the systems should accept not only correct sentences, but also the much wider spectrum of incorrect ones. On the other hand, statistical parsers induced from treebanks are able to analyse any sentence, but they can not easily distinguish correct sentences from incorrect ones.

- Error patterns (Kukich 1992; Golding and Schabes 1996; Mangu and Brill 1997), which are either hand-coded rules or are automatically learned using statistical techniques. Most of these approaches are implemented using finite-state techniques, for example the Constraint Grammar (CG) formalism (Karlsson, Voutilainen, Heikkila, and Anttila 1995) is used in (Arppe 2000; Birn 2000; Badia, Gil, Quixal, and Valentín 2004) for error detection in Swedish and Catalan, or the Xerox Finite State Tool (XFST)(Karttunen, Gaál, and Kempe 1997) for finding grammar errors in Swedish texts written by children (Hashemi, Cooper, and Andersson 2003).

Kukich (Kukich 1992) surveys the state of the art in syntactic error detection. She estimates that between 25% and over 50% of the total errors are in fact are valid words.

[1] http://ixa.si.ehu.es

On the other hand, (Atwell and Elliot 1987) made a manual study concluding that 55% of the errors are detectable by an examination of the local syntactic context, 18% are due to global syntactic errors (involving long-distance syntactic dependencies, which need a full parse of the sentence), and 27% are semantic errors.

Errors in date expressions can be deemed as a representative of local syntactic errors. A work similar to the one presented here is that of Karttunen (Karttunen 2006), who describes a system that mapped numbers to numerals in Finnish. This language has in common to Basque that the created linguistic structures are inflected, and some of their components must agree in case. That makes the transduction process of these languages more complex than in languages like English, with a simpler morphology.

Regarding the treatment of Basque date expressions, (Gojenola and Oronoz 2000) presented a system that detected some types of errors using an unification based partial parser. This work extends that system with a more comprehensive set of error types and also including the task of error correction.

Error type	Example
0. If the place name is inflected in inessive case (*Donostian*), the day number must be inflected in inessive case. If the place name is inflected in absolutive case (*Donostia*), the day number must be inflected in absolutive case.	Donostia**[n]**, 2007ko maiatzaren 27a**[]** *27th May, 2007* Donostia, 2007ko maiatzaren 27a**[n]**
1. The year number cannot be inflected using a hyphen	Donostian, 1995**[-]**eko maiatzaren 14an
2. The month (*maiatza*) must appear in lowercase	1999ko **[M]**aiatzaren 2an
3. The optional place name preceding dates (*Frantzia*) must be followed by a comma	Frantzia 1997ko maiatzaren 8an
4. The day number after a month in genitive case (*maiatzaren*) must have a case mark	Donostian, 1995eko maiatzaren 22**[]**
5. The day number after a month in ergative case (*maiatzak*) cannot have a case mark	1998.eko maiatzak 14**[ean]** argitaratua
6. The month (*maiatza*) must be inflected in genitive or absolutive case	Donostian, 1995eko Maiatza**[ren]** 14an
7. The dot that makes a number ordinal (*1995.eko*) cannot appear after the year number except when the word *urte* (*'year'*) follows it	Donostian 1997**[.]**eko Maiatzan 28an
8. Numbers 11 and 31 can not take the absolutive singular.	1997-ko maiatzaren 31**[a]**

Table 1: Most frequent error types in dates (errors marked in boldface).

3 Linguistic Resources

For the analysis of the input text, we use part of the Basque shallow syntactic analyser (Aduriz and Díaz de Ilarraza 2003), mainly based on finite-state technology (Aduriz, Aldezabal, Alegria, Arriola, Díaz de Ilarraza, Ezeiza, and Gojenola 2003). Although information at chunk or syntactic levels could be used for the treatment of other error phenomena, morphosyntactic information is enough for the recognition of errors in date expressions.

Figure 1 shows the morphosyntactic analyser and the modules for disambigua-

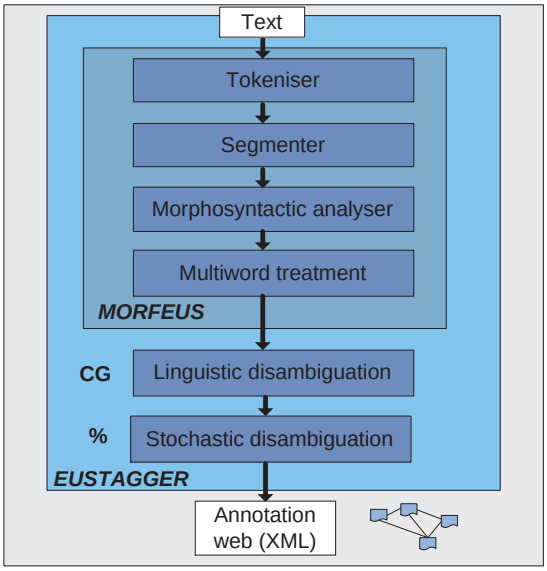

Figure 1: Morphosyntactic analysis and disambiguation.

tion. The process starts with the outcome of the morphosyntactic analyser (MORFEUS), which was created following the two-level morphology (Koskenniemi 1983), and deals with all the lexical units of a text, both simple words and multi-word units. The tagger/lemmatiser EUSTAGGER not only obtains the lemma and category of each form but also performs disambiguation using for this task information about part of speech, fine grained part of speech or case. The disambiguation process is carried out by means of linguistic rules using CG and stochastic rules based on Hidden Markov Models (Ezeiza 2003), which reduces the high word-level ambiguity to a limited amount of remaining interpretations.

All the information in the analysis chain is exchanged by means of standardised XML files (Artola, Díaz de Ilarraza, Ezeiza, Gojenola, Labaka, Sologaistoa, and Soroa 2005) and a class library for the management of all the linguistic information. The full system provides a robust basis, essential not only for any treatment based on corpora but also for error detection.

4 General Overview of the System

The process for error detection and correction starts after analysing the input text. The system (see figure 2) is composed of two groups of FSTs, one for error detection (see section 5) and the other one for the generation of correct dates (see section 6). Two filters prepare the input for each of these FST groups.

Take, for example, the date expression *"1995eko maiatzaren 15"* (15th of May, 1995). It is incorrectly written because in Basque the day number after a month in

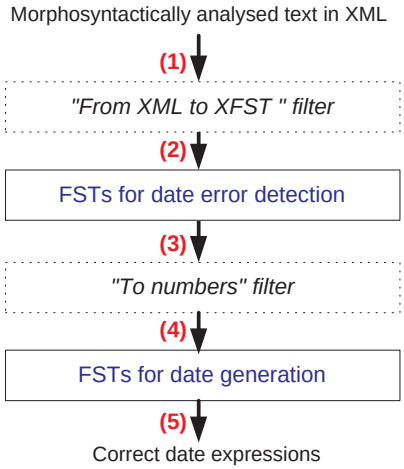

Figure 2: General architecture of the system.

genitive case must take a case mark. Given this text as input, the date expression will go through the following modules:

1. *"From* XML *to* XFST*" filter*. In a first step, the preprocessing filter changes the morphosyntactic information in XML to a more suitable format for the FSTs. Figure 3 shows the feature structures that gather the lemma and morphosyntactic information about the incorrect date example, including POS, FPOS (fine grained part of speech), CASE, NUM and MUG (definite/indefinite article). Figure 4 represents the corresponding simplified format.

$$\begin{bmatrix} \text{form} & \text{'1995eko'} \\ \text{lemma} & \text{'1995'} \\ \text{morph} & \begin{bmatrix} \text{pos} & \text{NOUN} \\ \text{fpos} & \text{NUMBER} \\ \text{case} & \text{GEN-L} \\ \text{num} & \text{S} \\ \text{mug} & \text{M} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \text{form} & \text{'Maiatzaren'} \\ \text{lemma} & \text{'maiatz'} \\ \text{morph} & \begin{bmatrix} \text{pos} & \text{NOUN} \\ \text{fpos} & \text{COMMON} \\ \text{case} & \text{GEN} \\ \text{num} & \text{S} \\ \text{mug} & \text{M} \end{bmatrix} \end{bmatrix} \begin{bmatrix} \text{form} & \text{'15'} \\ \text{lemma} & \text{'15'} \\ \text{morph} & \begin{bmatrix} \text{pos} & \text{NOUN} \\ \text{fpos} & \text{NUMBER} \end{bmatrix} \end{bmatrix}$$

Figure 3: Feature structures representing a date expression (see label 1 in figure 2)

2. FSTs *for date error detection*. For error detection in date expressions, we have sequentially applied nine finite-state transducers, one for each kind of error defined (see table 1), creating a cascade of FSTs. In the output of each of the FSTs, the incorrect linguistic structures are surrounded by tags describing each type of

error. Figure 4 shows how the incorrect structure is surrounded by two error tags (BEGINERRORTYPE4 and ENDERRORTYPE4)[2] .

```
{ WS { AS +form +1995eko +lemma +1995 +morph +POS +NOUN +FPOS +NUMB
    +CASE +GEN-L +NUMBER +S +MUG +M AE } WE }
BEGINERRORTYPE4
{ WS { AS +form +Maiatzaren +lemma +maiatz +morph +POS +NOUN +FPOS +COMMON
    +CASE +GEN +NUM +S +MUG +M AE } WE }
{ WS { AS +form +15 +lemma +15 +morph +POS +NOUN +FPOS +NUMBER AE } WE }
ENDERRORTYPE4
```

Figure 4: Output of the error detection grammar (labeled 3 in figure 2).

3. *"To numbers" filter*. Once the errors in date expressions are tagged (it is frequent to find more than one error in each date expression), the correction process starts. The FSTs used for error correction were not created specifically for this purpose but for helping Basque language learners to write date expressions[3]. The group of FSTs *for date generation* obtain the corresponding text equivalences from numbers representing date expressions. The *"To numbers"* filter obtains a numbered expression with the format *"year/month/day"*[4] (see figure 5) for each date expression tagged with an error.

Donostia *LocPn* 1995/05/15

Figure 5: Result after the application of the "To numbers" filter (labeled 4 in figure 2).

4. FSTs *for date generation*. As we have previously mentioned, the correction module uses FSTs that change numbers representing date expressions to their corresponding full-text equivalences. Figure 6 shows two correction candidates created for correcting the error in the example.

Donostia, 1995eko maiatzaren 15a
Donostian, 1995eko maiatzaren 15ean

Figure 6: Corrected date expressions (labeled 5 in figure 2).

5 Error Detection

Inflection in date expressions is a common source of errors, not detectable by a spelling checker, as each isolated word-form is correct. Figure 7 shows one of the formats of a valid date expression:

[2]The following tags are added to the morphosyntactic information to facilitate the regular expression definition in the XFST grammar: WS (word starts), WE (word ends), AS (analysis starts) and AE (analysis ends).

[3]http://kantauri.eleka.net/neh and http://sisx04.si.ehu.es/~ iniebla001/idazlagun/

[4]In Basque numerically written dates follow the format year/month/day.

Durangon,	*1999ko*	*martxoaren*	*7an*
Durango, inessive, sing	1999, genitive	martxoa, genitive, sing	7, inessive, sing
In Durango,	1999,	March	the 7th

Figure 7: Format of a valid date expression.

After examining different instances of errors, we chose the nine most frequent error types (see table 1). Some of these errors belong to idiosyncratic facts of date expressions (errors 0, 3, 4, 5 and 6), while four of them must be considered linguistically incorrect facts that can be reused in other general contexts (errors 1, 2, 7 and 8). A group of error detection patterns has been defined in XFST for each of the error types, and after compiling them, a cascade of FSTs is applied to the input text.

We adopted a kind of "gradual relaxation" approach, considering that several mistakes could co-occur, as quite often two or three errors might appear in the same expression. We had to design error patterns bearing in mind not only the correct expression, but also its erroneous versions. This relaxation on what could be considered a correct date had the risk of increasing the number of false alarms.

```
1.  define Month_Gen ...
2.  define Incorrect_Month_Gen_in_Upper ...
3.  define Correct_Year ...
4.  define Incorrect_Year_with_Hyphen ...
5.  define Year  [ Correct_Year | Incorrect_Year_with_Hyphen ]
6.  define Error_Type_4
           [Month_Gen | Incorrect_Month_Gen_in_Upper ] Not_Inflec_Numb;
7.  define Mark_Error_Type_4 [ Error_Type_4 ]
           @-> BEGINERRORTYPE4 ... ENDERRORTYPE4 || Year _ ;
```

Figure 8: Regular expressions for an error pattern.

The error pattern for the fourth kind of error (the day number after a month in genitive case must have a case mark) is defined in two steps (see figure 8). First, the syntactic pattern of the error is defined (a correct month or a month incorrectly written in uppercase followed by a non inflected number, see definitions 1 through 6), and named `Error_Type_4`. Second, a longest-match left-to-right replace operator (`@ - >`) is used (`Mark_Error_Type_4`) to surround the incorrect pattern (represented by ...) by two error tags (`BEGINERRORTYPE4` and `ENDERRORTYPE4`). To further restrict the application of the rule, left and right contexts for the error can be defined, mostly to assure that the rule is only applied to dates, thus avoiding false alarms. In this case, a year must be found to the left of the month. The year could be a correctly written year or a mispelled one (with a hyphen). As we can see, the error-description pattern considers the possibility that previous error patterns occur.

6 Error Correction

For the correction task, we took a group of already defined XFST transducers that was created to map numbers representing date, time and number expressions to text (Otaegi 2006), and adapted a subset of them in order to correct date expressions.

According to The Royal Academy of the Basque Language[5], the most appropriate ways to express a date are those in which the locative (place name) and the declension of the day agree in absolutive or inessive cases (in figure 6 the first date expression agrees in absolutive case and the second one in inessive case), so, we create date expressions in this format.

A FST has been used for each of the cases. These transducers, nevertheless, do not create the word indicating location and the comma after it. A FST for morphological generation created using *lexc* (Beesley and Karttunen 2003) is used to generate the locative in inessive case. The comma is generated after checking that a proper name indicating a locative (*LocPn* in figure 5) precedes the date.

The process of creating a full-text date is simple. Let us explain the rules for specifying the FST that generates dates in inessive case (see figure 9). The input is divided into three groups separated by slashes: year, month and day. When a year is found, a genitive locative[6]) morpheme (*"-ko"*) is added to the year number (`Translate_Year`, rule number 1). Months are mapped from numbers to text by means of replacement operators that are restricted to the date context (`Translate_Month`, rule number 3). Finally, the inessive singular morpheme (*"-an"*) is added to the day (`Translate_Day`, rule number 4). There are several exceptions to these mappings: when the year or day number finishes in a consonant, an epenthetic *"-e"* is added to the genitive locative case in the year (*"-e"* + *"-ko"* = *"-eko"*), and to the inessive case (*"-e"* + *"an"* = *"-ean"*) in the day (`Add_E_Day`, rule number 6).

```
1.  define Translate_Year [  "/" -> "ko" || _ Number Number "/" ];
2.  define Translate_Month05 [ "0" 5 "/ " -> " maiatzaren " || "ko" _ ];
3.  define Translate_Month [ Translate_Month01 .o.  Translate_Month02
      .o. ... .o.  Translate_Month12 ];
4.  define Translate_Day [ [ .. ]  -> "a" "n" || Number _ .#. ];
5.  define Translate [ Translate_Year .o. Translate_Month .o.
       Translate_Day ];
6.  define Add_E_Day [ "a" "n" -> "e" ... ||
       [ [ "0" | 2 | 4 | 6 | 8 ] 1 | [ 1 | 3 | 5 | 7 | 9 ] "0" | 5 ] _ ];
                ...
n-1.  define Clean [ Add_E_Day .o. Add_E_Year ];
n.  define Translate_Clean [ Translate .o. Clean ];
```

Figure 9: Regular expressions for date generation.

This method, based on the generation of correct date expressions, guarantees the

[5]http://www.euskaltzaindia.net, 37th rule

[6]The genitive locative case "-ko" ("of") is attached to phrases that denote location, or to phrases that denote a property.

correction of all the errors in the expression even if not all of them were detected. For example, if only 2 errors out of 3 are detected, all of them are properly corrected.

7 Evaluation

The evaluation corpus (development + test) is composed of 267 essays written by students (with a high proportion of errors) and texts from newspapers and magazines, more than 500,000 words altogether. From them we chose 658 sentences, including correct dates, incorrect dates, and also structures similar to dates. It was relatively easy to obtain test data compared to other kinds of errors. Although the data must be obtained mostly manually, date expressions contain several cues (month names, year numbers) that help in the process of finding semiautomatically test sentences.

All the corpus was inspected looking for false alarms (see table 2), that is, correct dates or sentences similar to dates that could be flagged as erroneous. The problem of false alarms is one of the biggest challenges we must face when dealing with unrestricted texts. As a result of the selection procedure, the proportion of errors was higher than in normal texts. Therefore, we divided our data into two groups. One of them was used for development and we left the second one for the final test. The proportion of correct dates was higher in the case of test data with respect to those in the development corpus, so that the effect of false alarms would be evaluated with more accuracy.

	Development corpus		**Test corpus**	
Number of test items	411		247	
Correct dates	51		35	
Structures "similar" to dates	263		173	
Incorrect dates	97		38	
Incorrect dates with 1 error	48	49.6 %	9	23.7 %
Incorrect dates with 2 errors	35	36.0 %	25	65.8 %
Incorrect dates with 3 errors	10	10.3 %	3	7.9 %
Incorrect dates with 4 errors	4	4.1 %	1	2.6 %

Table 2: Test data.

	Development corpus		**Test corpus**	
Number of test items	411 (97 errors)		247 (38 errors)	
Undetected date errors	4	4.1 %	3	7.9 %
Detected date errors	93	95.9 %	35	92.1 %
False alarms	2		4	

Table 3: Evaluation results.

Table 3 shows the results of the evaluation. As the development corpus could be inspected during the refinement of the patterns, the results in the second and third columns

can be understood as an upper limit of the system in its current state, with 95.9% recall[7] and 97.8% precision[8] (93 detected errors/95 error proposals, that is, 2 false alarms).

The system obtains 92.1% recall over the corpus of previously unseen 247 test items. Regarding precision the system correctly detects 35 errors, giving 39 proposals (89.7%). If the false alarms are divided by the number of test items (4/247) of the test corpus, we can estimate the false alarm rate to be around 1.6% over the number of dates in real texts. Table 4 examines some of the false alarms and their cause. Although the results are good, more corpus data will be needed in order to maximize precision.

The correction guarantees that all the errors in date expressions were corrected even when some of them could not be detected. That is, even when a sentence contains more than one error, once one is detected, it is transformed to the numerical format. As correct date expressions are generated from this format, all the errors are corrected.

Example	Cause of the error
1998ko abenduak 20. Bizkaiko → 1998ko abenduak 25. *20th December, 1998. From Bizcay 25th December, 1998*	The analyser does not detect the line end and analyses the *Bizkaiko* place name as it was immediately preceding the date expression. If it was the case, the comma is missing.
Primakovek 1998ko irailaren 11n hartu zuen ... *Primakov took it on the 11th of September 1998*	The unknown word *Primakov* is interpreted as a place name.

Table 4: False alarms.

8 Conclusions and Future Work

This work shows an application of XFST for syntactic error detection and correction in date expressions. The reported experiment is based on a corpus, and tested on real examples of both correct and incorrect sentences. This approach implies the existence of big corpora and manual annotation for most of the errors.

Two of the most successful methods for error detection, i.e., relaxation of syntactic constraints and error patterns, have been combined in our system with good results. Relaxation has not been dynamically applied at parsing time, but it has been manually coded. This implies a considerable amount of work, as we had to consider the formats for valid sentences as well as for all their incorrect variants. Regular expressions in the form of automata and transducers are suitable for the definition of complex error patterns based on linguistic units.

We are currently exploring extensions to the system to detect new kinds of errors by combining rule-based error detection and automatic acquisition of error patterns. We think that this could help to smooth the scaling-up problem associated to the increase in the number of rules, and the amount of work in the process of hand-coding them. Using either hand-coded rules or automatically learned ones, both methods have still the problem of obtaining and marking big test corpora. Some experiments with the

[7] recall = correctly detected errors/all errors

[8] precision = correctly detected errors/(correctly detected errors + false alarms)

automatic creation and tagging of errors (Sjöbergh and Knutsson 2005;Wagner, Foster, and van Genabith) seem to be a possible solution to this bottleneck.

We plan to extend the error detection/correction system to other qualitatively different types of errors, such as those involving agreement between the main components of the sentence, which is very rich in Basque, errors due to incorrect use of subcategorization and errors in post-positions. Errors in post-positions, determiner-noun agreement errors, ... could be treated using XFST, but a deeper study must be made if we want to deal with errors involving long-distance dependencies in the sentence (e.g. agreement between verb and subject, object or indirect object). Although the number of potential syntactic errors is huge, we think that the treatment of the most frequent kinds of error with high recall and precision can result in useful grammar-checking tools.

Acknowledgments. This research is supported by the University of the Basque Country (GIU05/52) and the Ministry of Industry of the Basque Government (ANHITZ project, IE06-185). We would like to thank Ruben Urizar for his collaboration in this work.

References

Aduriz, I., I. Aldezabal, I. Alegria, J. M. Arriola, A. Díaz de Ilarraza, N. Ezeiza, and K. Gojenola (2003). Finite State Applications for Basque. In *EACL 2003 Workshop on Finite-State Methods in Natural Language Processing*.

Aduriz, I. and A. Díaz de Ilarraza (2003). Morphosyntactic Disambiguation and Shallow Parsing in Computational Processing of Basque. *Inquiries into the lexicon-syntax relations in Basque*, 1–21.

Agirre, E., I. Alegria, X. Arregi, X. Artola, A. Díaz de Ilarraza, M. Urkia, M. Maritxalar, and K. Sarasola (1992). XUXEN: A Spelling Checker/Corrector for Basque Based on Two-Level Morphology. In *Proceedings of ANLP'92*, Povo Trento, pp. 119–125.

Arppe, A. (2000, December 9-10). Developing a Grammar Checker for Swedish. In *Proceedings from the 12th Nordiske datalingvistikkdager*, Department of Linguistics, Norwegian University of Science and Technology (NTNU). Nordgard.

Artola, X., A. Díaz de Ilarraza, N. Ezeiza, K. Gojenola, G. Labaka, A. Sologaistoa, and A. Soroa (2005). A Framework for Representing and Managing Linguistic Annotations Based on Typed Feature Structures. In *Proceedings of Recent Advances on NLP (RANLP05)*, Borovets, Bulgaria.

Atwell, E. and S. Elliot (1987). *The Computational Analysis of English: a Corpus-Based Approach*, Chapter Dealing with Ill-Formed English Text. De Longman.

Badia, T., A. Gil, M. Quixal, and O. Valentín (2004). NLP-enhanced Error Checking for Catalan Unrestricted Text. In *Proceedings of the fourth international conference on Language Resources and Evaluation, LREC 2004*, Lisbon, Portugal, pp. 1919–1922.

Beesley, K. R. and L. Karttunen (2003). *Finite State Morphology*. CSLI Publications.

Birn, J. (2000, December 9-10). Detecting Grammar Errors with Lingsofts Swedish Grammarchecker. In *Proceedings from the 12th Nordiske datalingvistikkdager*, Department of Linguistics, Norwegian University of Science and Technology (NTNU). Nordgard.

Douglas, S. and R. Dale (1992). Towards Robust PATR. In *COLING*, pp. 468–474.

Ezeiza, N. (2003). *Corpusak ustiatzeko tresna linguistikoak. Euskararen etiketatzaile sintaktiko sendo eta malgua. PhD thesis*. Donostia: University of the Basque Country.

Gojenola, K. and M. Oronoz (2000, April 30). Corpus-based Syntactic Error Detection Using Syntactic Patterns. In *NAACL-ANLP00,Student Research Workshop*.

Golding, A. R. and Y. Schabes (1996). Combining Trigram-Based and Feature-Based Methods for Context-Sensitive Spelling Correction. In A. Joshi and M. Palmer (Eds.), *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, San Francisco, pp. 71–78. Morgan Kaufmann Publishers.

Hashemi, S. S., R. Cooper, and R. Andersson (2003). Positive Grammar Checking: A Finite State Approach. In *Computational Linguistics and Intelligent Text Processing, 4th International Conference, CICLing 2003, Mexico City, Mexico, February 16-22*, Volume 2588 of *Lecture Notes in Computer Science*, pp. 635–646. Springer.

Heidorn, G., k. Jensen, L. Miller, R. Byrd, and M. Chodorow (1982). The EPISTLE Text-Critiquing System. *IBM Systems Journal 21*(3).

Karlsson, F., A. Voutilainen, J. Heikkila, and A. Anttila (1995). *Constraint Grammar: Language-independent System for Parsing Unrestricted Text*. Berlin: Prentice-Hall.

Karttunen, L. (2006). Numbers and Finnish Numerals. *In A Man of Measure Festschrift in Honour of Fred Karlsson on his 60th Birthday, a special supplement to SKY Journal of Linguistics 19*, 407–421.

Karttunen, L., T. Gaál, and A. Kempe (1997). Xerox Finite State Tool. Technical report, Xerox Research Centre Europe.

Koskenniemi, K. (1983). *Two-Level Morphology: a General Ccomputational Model for Word-form Recognition and Production*. Helsinki: University of Helsinki.

Kukich, K. (1992, December). Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys 24*(4), 377–439.

Mangu, L. and E. Brill (1997). Automatic Rule Acquisition for Spelling Correction. In *Proceedings of the 14th International Conference on Machine Learning*, pp. 187–194. Morgan Kaufmann.

Min, K. and W. H. Wilson (1998). Integrated Control of Chart Items for Error Repair. In *COLING-ACL*, pp. 862–868.

Otaegi, M. (2006). Datak, orduak eta zenbakiak euskaraz. Technical report, University of the Basque Country.

Sjöbergh, J. and O. Knutsson (2005). Faking Errors to Avoid Making Errors: Very Weakly Supervised Learning for Error Detection in Writing. In *Proceedings of RANLP 2005*, Borovets, Bulgaria, pp. 506–512.

Wagner, J., J. Foster, and J. van Genabith. A Comparative Evaluation of Deep and Shallow Approaches to the Automatic Detection of Common Grammatical Errors. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 112–121.

Weischedel, R. and N. Sondheimer (1983). Meta-rules as a Basis for Procedding Ill-Formed Input. *American Journal of Computational Linguistics 9*(3-4), 161–177.