Natural Language Engineering **17** (2): 145–161. © Cambridge University Press 2011
doi:10.1017/S1351324911000088

Syntactic error detection and correction in date expressions using finite-state transducers

ARANTZA DÍAZ DE ILARRAZA, KOLDO GOJENOLA, MAITE ORONOZ and IÑAKI ALEGRIA

Department of Computer Languages and Systems, University of the Basque Country
P.O. box 649, E-20080 Donostia, the Basque Country, Spain
emails: a.diazdeilarraza@ehu.es, koldo.gojenola@ehu.es, maite.oronoz@ehu.es, i.alegria@ehu.es

(*Received 15 June 2010; revised 24 September 2010; accepted 22 October 2010*)

Abstract

This paper presents a set of experiments for the detection and correction of syntactic errors, exploring two alternative approaches. The first one uses an error grammar which combines a robust morphosyntactic analyser and two groups of finite-state transducers (one for the description of syntactic error patterns and the other for the correction of the detected errors). We have also experimented an alternative approach using a positive date grammar where deviations are detected by applying edit-distance techniques. The system has been tested on a corpus of real texts which contained both correct and incorrect sentences. Although the experiment was limited to one language, the results show that attainable performance is not the only criterion for preferring one solution over another.

1 Introduction

In this work, we present a research carried out to detect and correct syntactic errors in date expressions using local grammars which are compiled into finite-state transducers (FSTs). Finite-state constraints compiled into transducers have been applied to linguistic analyses. We have used two general-purpose libraries for the creation of automata and transducers: the Xerox Finite State Tool XFST (Karttunen, Gaál and Kempe 1997) and Foma (Hulden 2009b). Both allow the definition of complex linguistic patterns applied to morphosyntactic information.

We chose to deal with date expressions because they contain morphologically and syntactically rich phenomena where several types of errors can be found. They can be considered representative of the types of syntactic errors that are detectable when examining local syntactic contexts (that is, a window of five or six consecutive words). According to copy editors and language teachers (Zubiri 1994), date expressions in Basque are one of the most frequent source of errors in both language learners and native speakers. Date expressions are also an example of a practical problem

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

Table 1. *Most frequent error types in dates (errors marked in brackets)*

Error type	Example
0. If the place name is inflected in inessive case (*Donostian*), the day number must be inflected in inessive case.	Donostia**[n]**, 2007ko maiatzaren 27a**[]** *27th May, 2007*
If the place name is inflected in absolutive case (*Donostia*), the day number must be inflected in absolutive case.	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]**

that must be solved in several contexts, such as Named Entity Recognition, text normalisation, or semantic mapping of time expressions.

Basque is an agglutinative language and, as a consequence, most of the elements that appear in date expressions (year numbers, months and days) must inflect, i.e. the corresponding article and case morphemes need to 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). As a result, different types of agreement are needed. These require a linguistic (morphological analysis and morphosyntactic disambiguation) analysis prior to applying the FSTs for detection and correction, as many date errors are syntactic rather than morphological.

The remainder of this paper is organised as follows. Section 2 reviews several related works. After describing in Section 3 the linguistic resources we have used, we explain our error grammar approach in Section 4. Section 4.1 describes the error detection process, while Section 4.2 presents the correction procedure. In Section 5 we experiment with an alternative approach for detecting errors using a positive grammar, detecting errors by means of edit-distance techniques. Then, we present the evaluation results in Section 6, and the final conclusions in Section 7.

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

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

amount of errors that result in valid words (Heidorn *et al.* 1982; Weischedel and Sondheimer 1983) different techniques have been proposed:

- Grammar-based techniques that use the results of a parser as input. Two kinds of parsers can be used for this purpose:
  - — Parsers based on manually developed grammars that describe correct structures. Special methods have been devised for error detection: chart-based methods (Min and Wilson 1998), relaxation of syntactic constraints (Douglas and Dale 1992) or edit-distance methods (Gusfield 1997).
  - — Statistical parsers can analyse any sentence, but they do not easily distinguish correct and incorrect sentences, although they can make a probabilistic approximation of grammaticality (Wagner and Foster 2009).
- Error patterns (Kukich 1992; Golding and Schabes 1996; Mangu and Brill 1997), which are either hand-coded rules or automatically learned ones. Most of these approaches use finite-state techniques, for example the Constraint Grammar (CG) formalism (Karlsson *et al.* 1995) is used in Arppe (2000), Birn (2000), and Badia *et al.* (2004) for error detection in Swedish and Catalan, or the Xerox Finite State Tool (XFST; Karttunen *et al.* 1997) for finding grammar errors in Swedish texts written by children (Hashemi, Cooper and Andersson 2003).

Kukich (1992) surveyed the state of the art in syntactic error detection. She estimates that between 25% and over 50% of the total errors in written texts are in fact valid words. On the other hand, Atwell and Elliot (1987) made a manual study concluding that 55% of these errors are detectable examining 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. Since then, new technologies have extended the possibilities of other types of texts, such as e-mails or chats, where the chance of finding errors increases considerably, and so does the applicability of error detection systems (Foster 2010).

Errors in date expressions can be deemed as representative of local syntactic errors. A similar work to the one presented here is that of Karttunen (2006), which describes a system that maps numbers to numerals in Finnish. Finnish has in common with Basque the inflection of linguistic structures, and agreement in case of some of their components that makes the transduction process more complex than in languages like English, with a simpler morphology. Constant (2009) gives a similar account of a finite-state grammar describing measurement phrases in French.

Recognisers for date expressions are used as a part of the task of Named Entity Recognition (NER; Friburger and Maurel 2004; Traboulsi 2009). However, these applications either do not take the problem of incorrect dates into account or otherwise they define a loose description of dates that covers incorrect dates as well.

Regarding the treatment of Basque date expressions, Gojenola and Oronoz (2000) present a system that detects errors using a unification based partial parser. This work extends that system using a finite state approach instead of unification, with a more comprehensive set of error types and including also the task of error correction.

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

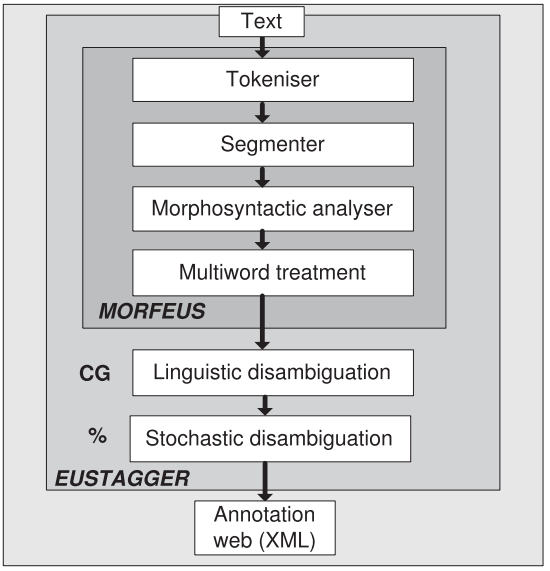


Fig. 1. Morphosyntactic analysis and disambiguation.

There are several general-purpose libraries for the creation of transducers. While some of them are proprietary software, such as XFST (Karttunen *et al.* 1997), there are also alternatives based on free software, like Nooj (http://www.nooj4nlp.net/NooJManual.pdf, based on INTEX (Gross 1997)) or Foma (Hulden 2009b). XFST is the result of long-term research at the Xerox Palo Alto Research Center since the early 1980s. Foma is a free software tool to specify finite-state automata and transducers in a similar fashion to AT&T's FSM Library (Mohri, Pereira and Riley 2000), XFST and the SFST toolkit (Schmid 2005). One of Foma's main design goals was its compatibility with XFST. In an earlier experiment in Machine Translation transfer (transferring verb chains), we evaluated the Nooj-FSA tool, concluding that it was less time-efficient than XFST and Foma. This is the reason why we selected them for this work.

The problem of using either a positive or a negative approach is a well studied problem (Laporte 1998) whose practical consequences are still not settled. We will evaluate both methods in the context of syntactic error detection and correction.

3 Linguistic resources

For the analysis of the input text we use the Basque shallow syntactic analyser (see Figure 1), based mainly on finite-state technology (Aduriz *et al.* 2003). The recognition of errors in date expressions only needs morphosyntactic information (lemma, POS category and features like syntactic case and number).

Figure 1 shows the morphosyntactic analyser together with the modules in charge of disambiguation. The process starts with the output 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

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

multi-word units. The tagger/lemmatiser EUSTAGGER obtains the lemma, category and morphosyntactic features of each form and also performs disambiguation using information on 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 2002), which reduces the high word-level ambiguity to a limited amount of interpretations per wordform.

All the information in the analysis chain is exchanged by means of standardised XML files (Artola *et al.* 2009). The full system provides a robust basis, essential not only for any NLP treatment based on corpora, but also for error detection.

4 The error grammar approach

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: (i) for error detection (see Section 4.1) and (ii) for the generation of correct dates (see Section 4.2). Let us consider the date expression *'1995eko maiatzaren 15'* (15th of May, 1995). It is incorrectly written because in Basque the day number after a month in genitive case must take a case marker. The process will go through the following modules:

(1) *'From* XML *to* XFST*' filter*. This preprocessing filter (a C++ program) changes the morphosyntactic information to the format suitable for the FSTs. The text labelled (1) in Figure 2 shows the result of the analysis represented by means of feature structures that gather the lemma and morphosyntactic information of the incorrect date example, including POS, FPOS (fine-grained part of speech), CASE, NUM (number) and MUG (definite/indefinite article). The text labelled (2) in Figure 2 represents the corresponding simplified format[1] needed for the next step.

(2) FST*s for date error detection.* We have sequentially applied nine finite-state transducers, one for each kind of error defined (see Table 1), creating a cascade of FSTs. The beginning and end of the incorrect linguistic structures are marked by means of tags describing each type of error. The text labelled (3) in Figure 2 shows how the incorrect structure is surrounded by two error tags (BEGINERROR4 and ENDERROR4). Figure 5 presents the set of transducers defined to detect the error number four (in Table 1).

(3) *'To numbers' FST*. 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 *'To numbers'* FST obtains a numbered expression with the format *'YYYY/MM/DD'* for each date expression tagged with an error. The 'To numbers' transducer is very simple because it only has to match the numbers corresponding to the year and the day and translate the name of the months to the corresponding number.

(4) FST*s for date generation.* The correction module uses FSTs that change numbers representing date expressions to their corresponding full-text equivalences. The

[1] The following tags are added to facilitate the definition of FSTs: WS (word starts), WE (word ends), AS (analysis starts) and AE (analysis ends).

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

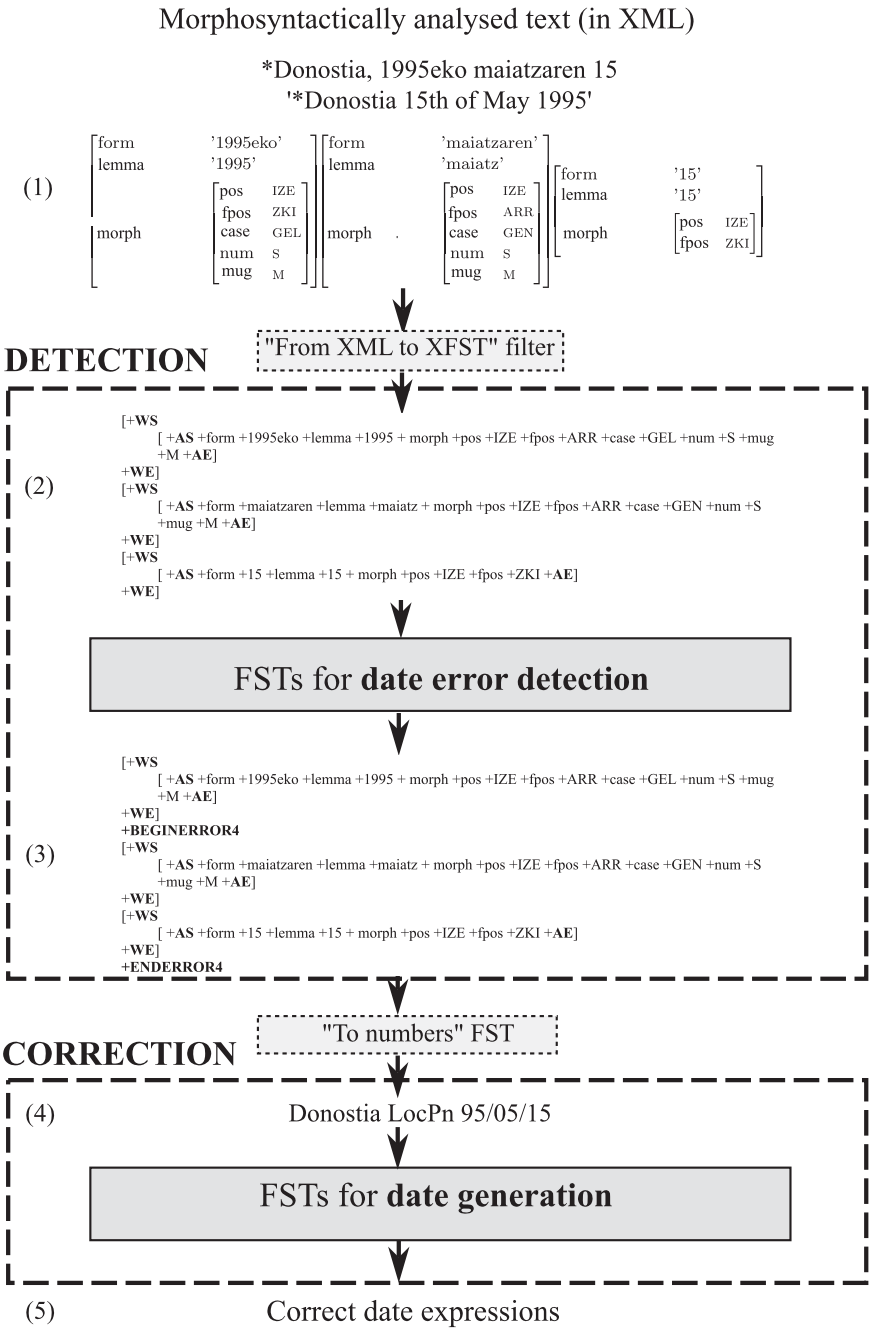


Fig. 2. General architecture of the system.

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

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

Fig. 3. Format of a valid date expression.

text numbered (5) in Figure 2 shows two correction candidates for correcting the example. There are two standard forms (with variants) for Basque date expressions, and the input date is corrected to one of them. In this respect, this process could better be defined as normalisation instead of correction. Figure 6 presents the set of transducers defined for date generation.

4.1 Error detection

Inflection in date expressions is a common source of errors, not detectable by a spelling checker. Figure 3 shows one of the formats of a valid date expression.

After a preliminary study , we chose the most frequent error types (see Table 1). Errors 0, 3, 4, 5 and 6 are idiosyncratic errors of date expressions, while errors 1, 2, 7 and 8 can be considered general errors that can be found in other contexts. A group of error detection patterns has been defined for each of the error types, and after compiling them, a cascade of FSTs is applied to the input text.

The error pattern rules were devised considering that several mistakes could co-occur. As a result, we designed error patterns bearing in mind not only the correct expression, but also its erroneous versions. This strategy had associated the risk of increasing the number of false alarms. This means that when writing rules for the detection of one kind of error, we took the transducers for correct dates as a basis, but we had to relax these definitions for correct dates, so that other kinds of date errors could also have co-occurred together with the currently treated error. This softening on what can be considered a correct date has the problem of marking as erroneous text units that do not correspond to dates (for example, expressions containing case-marked months and years but not corresponding to a date). As a consequence, error detecting rules were considerably more complex when adapting them to the possible co-occurrence of other errors. For example, Figure 5 shows the error detecting rules for the fourth kind of error, where the rules take both correct and possibly incorrect definitions of years and months. As the interactions among errors grow considerably with the number of errors, we based our error patterns on the combinations actually found in the corpus, so that in practice that number can be considerably reduced.

Figure 5 shows the error pattern for the fourth kind of error: the day number after a month in genitive case must have a case mark. We have used the FST operators presented in Figure 4. Rules 1–4 define regular expressions describing years and months in both their correct and incorrect variants. Rules number 5 and 6 define two regular expressions describing a year (`Year`) and a month (`Month`) (both the correct and incorrect variants are defined, using the disjunction (|) operator). This way, the error-description patterns will consider the possibility that

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

```
[ ]                    The empty-string language/identity relation
.#.                    The beginning/end of a string
(A)                    Union of A with the empty string (option)
~A                     Complement (negation)
$A                     Contains (all strings containing at least one A)
A*                     Kleene star
A+                     Kleene plus
A B                    Concatenation
A | B                  Union
A .o. B                Composition. Maps strings in the upper language
                       of A to strings in the lower language of B
A -> B ... C || L _ R  Conditional replace operator. It marks
                       expression A with tags B and C, when left (L)
                       and right (R) contexts are met
@-> B ... C || L _ R   Conditional longest-match left-to-right replace
A.l                    Lower-side language of A
```

Fig. 4. List of the main FST operators.

```
1. define Correct_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 Month       [ Correct_Month_Gen  |  Incorrect_Month_Gen_in_Upper ]
7. define Err_Type4  Month   Non_Inflected_Number;
8. define Mark_Err4 [ Err_Type4 ] @-> BEGINERROR4 ... ENDERROR4 || Year _ ;
```

Fig. 5. Regular expressions and relations for an error pattern.

other error patterns co-occur. The error pattern is defined in rule 7 (a month (`Month`) followed by a non inflected number), and named `Err_Type4`. Finally, rule 8 describes a transducer (`Mark_Err4`) that uses the longest-match left-to-right replace operator (`@->`), to surround the incorrect pattern by two error tags (BEGINERROR4 and ENDERROR4). To further restrict the application of the rule, the left context for the error (`Year`) is defined, to assure that the rule is only applied to dates, thus avoiding false alarms.

4.2 Error correction

For correction we took a group of already defined two-tape 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, 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 the text labelled (5) in Figure 2 the first date expression agrees in absolutive and the second one in inessive), so we decided that the system should create date expressions in this format.

An FST has been used for each of the cases. These transducers, however, do not create the word indicating the location and the comma after it, but an FST is used to generate the locative in inessive case, using morphological generation by means

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

```
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 ];
```

Fig. 6. Regular relations for date generation.

of *lexc* (Beesley and Karttunen 2003). The comma is generated after checking that a proper name indicating a locative (*LocPn* in Figure 2) 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 6). The input comes in the format: year/month/day. Rule number 1 replaces the slash after the year number (using the conditional replacement operator ->) by a genitive locative[2] morpheme (*'-ko'*). Months are mapped from numbers to text (actually, we make use of multi-character symbols such as *'maiatzaren'*) by means of replacement operators that are restricted to the date context (rule 2 shows an example). Rule 3 composes transducers of different months by means of the composition operator (.o.). 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). The last two rules compose the defined transducers.

This method, based on the generation of correct date expressions, guarantees the correction of all the errors in the expression even if not all of them were detected. For example, if only two errors out of three are detected, all of them are properly corrected.

5 An alternative: a recognition-based method

A simpler, more general, and also less-labour-intensive alternative than developing incorrect date error patterns, is to develop a simpler FST for date identification and to use edit-distance based techniques to detect errors. In this case the type of error will not be identified (the diagnosis capability of the system is lost) but simple errors will be detected and corrected. For this purpose we have used an FST based on the ideas from Karttunen (2006) which can identify correct dates obtained from the low-side language in a transducer that converts dates from number to text. So, we

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

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

```
define Date [ YearDigits "/" MonthDigits "/" DayDigits ];

# Tags: transduction from 2006/06/04 to 2006Y06M04D
# The epenthesis rule has the form [..] -> A || L _ R, and
# inserts a new symbol A into a string.
# .#. represents the beginning/end (left/right context) of a string
define Tags [ [..] -> D  ||  Digit _ .#. ] .o.
            [ "/" -> M  ||  Digit _ Digit Digit D ] .o.
            [ "/" -> Y  ||  Digit _ Digit Digit M ];

# Declension: attaches suffixes "ko" and "a" to year and day
define Declension [ Y -> k o ] .o. D -> a ;
# String and declension for months
define MonthTrans [ "0" 1 M ->  urtarrilaren  .o. [ "0" 2 M -> ...] ... ;
# Epenth adds an epenthetic e depending on the spellings
define Epenth [[..]-> e  ||
                 [ EvenDigit 1 | OddDigit "0"  |  5  |  .#. 1 ] _ k o];
define DateConversion [ Date .o. Declension .o. MonthTrans .o. Epenth ];

# Take the lower-side language using the .l operator
define DateId DateConversion.l ;

# Optional place name before the date
define DateStr [ (Place (n) "," " " ) DateId (n) ].l ;
```

Fig. 7. Regular relations for date identification.

```
define ForbidN ~$[ n "," " " DateId n ] ;
# $A denotes the strings containing A, and ~A denotes the complement,
# that is, ForbidN denotes the strings not containing the expression
```

Fig. 8. Enforcing case agreement.

will use the lower-side of a number-to-text transducer. The simplified grammar of this date recogniser is shown in Figure 7. The grammar follows the syntax already presented in the previous figures, so we have explained it by means of comments. The grammar accepts expressions such as *Donostia, 2010eko apirilaren 10a.*

Related to agreement in the inessive case (the 'n' suffix appearing in error number 0 in Table 1) two rules have been designed in order to constrain its correctness. If the suffix appears attached to the place name, it should also appear attached to the day. Figure 8 shows one of the rules, which forbids the second 'n' (day) if it is not linked to a corresponding 'n' attached to the place name.

For identification and correction of errors we can use the *med* (minimum edit distance) option in the Foma compiler (Hulden 2009a, 2009b). The *med* option applies a fast algorithm for finding approximate matches of a string in a finite-state automaton (which can also be seen as an identity transducer), given some metric of similarity. The algorithm can be adapted to use a variety of metrics to determine the distance between two words. Without any parameter, the same cost is assigned to all the possible variations (deletion, insertion and substitution), but a matrix can be defined when different costs are assigned.

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

```
Insert 2
Substitute 2
Delete 2
Cost 1
:n n: ,: :a a: :e e: a:A e:E i:I o:O u:U m:M
```

Fig. 9. Matrix for defining the minimum edit-distance costs.

For example, in the matrix in Figure 9, a lower cost (1 against 2 for the general changes) is assigned to the changes corresponding to errors 0, 1, 2, 3 and 8 in Table 1. The change corresponding to the 7th error type can not be included because the character (hyphen) is not in the defined language.

The three first lines in Figure 9 express that general insertions, substitutions and deletions have a cost of two. The other two lines define a cost of one for well-known errors. Each of these errors is specified by a pair of characters separated by a colon. Additions are represented by pairs without the left character, and deletions by pairs without the right character. Looking at the last line, we can distinguish:

- 6 substitutions. The last six pairs specify substitutions among lower-case and upper-case.
- 3 additions. ':n,' ':e' and ':a' specify the addition of 'n,' 'e' and 'a'.
- 4 deletions: 'n:," ',:', 'e:' and 'a:'; deletion of 'n', ',', 'e' and 'a'.

As an example, when analysing the wrong date expression *Donostia, 2010eko apirilaren 10* using *med* it will give as alternative the correct *Donostia, 2010eko apirilaren 10a* date with a cost of 1. So, using the *med* option, erroneous dates can be identified and corrected. The *med-cutoff* option can limit the cost terminating a search. Based on the matrix in Figure 9 with the *med-cutoff* parameter set to 3, expressions with a single change (insertion, deletion or substitution) and a special change (one of the lower cost) will be corrected. For example in *Donostia, 2010eko apirillaren 10* there are two errors, the deletion of the *a* (cost 1) and one general insertion (apirilaren → apirillaren) (cost 2). Depending on the value of these parameters, different trade-offs for precision/recall/time can be obtained.

6 Evaluation

The evaluation corpus (development + test, see Table 2) 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. Out of them we chose 658 sentences, including correct dates, incorrect dates and also structures similar to dates. Compared to other kinds of errors, it was relatively easy to obtain test data because date expressions contain several cues (month names, year numbers) that help in the process of semiautomatically finding test sentences.

The corpus was compiled with the double objective of both finding errors and also testing 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

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

Table 2. *Test data*

	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 3. *Evaluation results (error pattern grammar)*

	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	0		1	

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 than in the development corpus, so that the effect of false alarms could be evaluated with more accuracy.

6.1 Error grammar

Table 3 shows the results. As the development corpus could be inspected during the refinement of the patterns, the results in the number of detected and undetected errors in the development corpus can be understood as an upper limit of the system in its current state, with 95.9% recall[3] and 100% precision[4] (93 error proposals).

The system obtains 92.1% recall over the corpus of previously unseen 247 test items. Regarding precision the system correctly detects 35 errors. If the false alarms are divided by the number of test items (1/247) of the test corpus, we can estimate the false alarm rate to be around 0.4% over the number of dates in real texts. Figure 10 shows the unique false alarm and its cause.

The correction guarantees that all the errors in dates are 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 its corresponding number format and, as the correct date is generated, all the errors are corrected.

[3] recall = correctly detected errors/all errors.
[4] precision = correctly detected errors/(correctly detected errors + false alarms).

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

Example: Primakovek 1998ko irailaren 11n hartu zuen ...
'*Primakov took it on the 11th of September 1998*'

Fig. 10. False alarm. The unknown word *Primakov* is interpreted as locative.

Table 4. *Size of the implemented transducers for error detection*

Transducer Error number (in Table 1)	Recognition		Marking	
	Arcs	States	Arcs	States
0	444	18,332	5,940	775,439
1	50	434	843	30,557
2	20	524	3,841	176,816
3	53	1,784	15,703	757,119
4	122	4,337	1,330	61,827
5	158	6,119	1,753	86,697
6	121	4,426	2,289	113,704
7	118	3,897	5,598	270,190
8	128	4,761	323	14,835

6.2 Positive grammar

The recognition of date errors using a general, positive FST gives competitive results when compared to the first method. Correct dates are recognised correctly and, in the case of incorrect dates, the main difference is that it is not possible to give a diagnosis. Nevertheless, and similarly to the system based on error patterns, most of the errors are identified and the corresponding right spelling proposed.

After doing different tests on the development corpus the *med* cutoff value was set to 4, resulting in a good trade-off between coverage and precision. In addition to this, bigger values would significantly slow the correction process.

The only difference between the two systems is that there are five undetected date errors (compared to three with the specific error transducer presented in Section 3). All the rest of the incorrect dates are identified and corrected. The number of false alarms on the test data is the same (one false alarm) as in the previous approach.

Table 4 gives a measure of the complexity of the nine transducers for the errors in Table 1, in terms of their number of states and transitions. It is worth noting that the error-marking transducers mark both the beginning and end of the error, which considerably increases the size of the corresponding transducer. This can be clearly seen in the table, where the error-marking transducers are sometimes two orders of magnitude bigger than their recognising counterparts. Table 5 shows that the transducer for error identification and the date-generation transducer are relatively simple compared to the size of the error-marking transducers.

Regarding the speed of the transducers, we measured the application of the 9 error detection FSTs to 1,000 date expressions, obtaining the result in 41 seconds, that is, 0.041 seconds per expression. This test was made in an Unix machine

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

Table 5. *Size of the finite state machines for date generation and recognition*

	States	Arcs
FST for date generation	57	1,136
FSA for date recognition	200	880

(1xSPARC64 VII + Quad Core) with a processor of 2.75 GHz and 8 Gigas of RAM memory.

7 Conclusions and future work

This work shows an application of finite state transducers (FST) 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 needs the existence of big corpora and manual annotation.

We have applied two alternative methods and two tools, XFST and Foma, to the task of error detection. As a first method, we have experimented a negative approach, using a set of error patterns. In order to detect several types of errors in the same expression, the error patterns have been written considering the formats for valid sentences as well as their incorrect variants. The results show that regular expressions and relations compiled to automata and transducers are suitable for the definition of complex error patterns based on linguistic units. The second approach was based on the definition of correct dates, and the application of edit-distance techniques to detect errors. As the results have shown (see Section 6) both systems obtain good results in terms of precision/recall, although the first one allows the identification of the error type(s), which can be more explanatory in contexts like an error correction system or a language learning environment. This second approach (see Section 5) can be useful when diagnosis is not necessary and rapid development is an issue. In both methods we are able to obtain the corrections without having to detect all the errors occurring in the ungrammatical date expression.

From our experiments we can conclude that if a diagnosis is not necessary for the checker the second solution is better because it is a less labour-intensive option and because the solution is less dependent on the language. However when a diagnosis is necessary only the first solution is suitable.

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. This could counteract the scaling-up problem associated with the increase in the number of rules, as well as the amount of work involved in the process of hand-coding them. Using either hand-coded rules or automatically learned ones, both methods still pose the problem of obtaining and marking big test corpora. Some experiments with the automatic creation of errors (Sjöbergh and Knutsson 2005; Foster and Andersen 2009) seem to be a possible solution to this bottleneck.

In the future, we plan to extend the error detection system to other qualitatively different types of errors such as those involving for example, determiner-noun

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

agreement errors that could be treated using FSTs. Although the number of potential syntactic errors is huge, in our opinion the treatment of the most frequent kinds of errors with high recall and precision can result in useful grammar-checking tools.

Acknowledgments

This research is supported by the Basque Government (IT344-10), the Ministry of Industry of the Basque Government (Berbatek project, IE09-262) and the Spanish Ministry of Science and Innovation (IMLT project, TIN2007-63173). We would like to thank the editor and three anonymous reviewers for their constructive suggestions and comments.

References

Aduriz, I., Aldezabal, I., Alegria, I., Arriola, J., Díaz de Ilarraza, A., Ezeiza, N., and Gojenola, K. 2003. Finite state applications for Basque. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL 2003). Workshop on Finite-State Methods in Natural Language Processing*, Budapest, Hungary.

Arppe, A. 2000. Developing a grammar checker for Swedish. In *Proceedings of the 12th Nordiske datalingvistikkdager*, Department of Linguistics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

Artola, X., Díaz de Ilarraza, A., Soroa, A., and Sologaistoa, A. 2009. Dealing with complex linguistic annotations within a language processing framework. *IEEE Transactions on Audio, Speech and Language Processing* **17**(5): 904–15.

Atwell, E., and Elliot, S. 1987. Dealing with Ill-Formed english text. In R. Garside, G. Sampson, and G. Leech (eds.), *The Computational Analysis of English: a Corpus-Based Approach*. London, UK: De Longman.

Badia, T., Gil, A., Quixal, M., and Valentín, O. 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. 1919–22.

Beesley, K. R., and Karttunen, L. 2003. *Finite State Morphology*. Stanford, CA, USA: CSLI Studies in Computational Linguistics.

Birn, J. 2000. Detecting grammar errors with Lingsoft's Swedish grammar-checker. In *Proceedings of the 12th Nordiske datalingvistikkdager*, Department of Linguistics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

Douglas, S., and Dale, R. 1992. Towards robust PATR. In *Proceedings of the 14th Conference on Computational Linguistics (COLING 1992)*, Nantes, France, pp. 468–74.

Ezeiza, N. 2002. *Corpusak Ustiatzeko Tresna Linguistikoak. Euskararen Etiketatzaile Sintaktiko Sendo eta Malgua*. Ph. D. thesis, University of the Basque Country, Donostia-San Sebastin, Spain.

Foster, J. 2010. 'cba to check the spelling': investigating parser performance on discussion forum posts. In *Proceedings of the 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics and Human Language Technologies (NAACL-HLT 2010)*, CA, USA.

Foster, J., and Andersen, I. 2009. GenERRate: generating errors for use in grammatical error detection. In *Proceedings of the NAACL Workshop on Innovative Use of NLP for Building Educational Applications*, CO, USA.

Friburger, N., and Maurel, D. 2004. Finite-state transducer cascades to extract named entities in text. *Theoretical Computer Science* **313**(1): 94–104.

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

Gojenola, K., and Oronoz, M. 2000. Corpus-based syntactic error detection using syntactic patterns. In *Proceedings of The NAACL-ANLP00, Student Research Workshop*, Seattle, WA, USA.

Golding, A. R., and Schabes, Y. 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*, Santa Cruz, CA, pp. 71–8.

Gross, M. 1997. *Finite-State Language Processing*, Chapter The Construction of Local Grammars, pp. 329–54. Cambridge, MA, USA: The MIT Press.

Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, UK: Cambridge University Press.

Hashemi, S. S., Cooper, R., and Andersson, R. 2003. Positive grammar checking: a finite state approach. In *Proceedings of the 4th International Conference in Computational Linguistics and Intelligent Text Processing (CICLing 2003)*, Volume 2588 of *Lecture Notes in Computer Science*, pp. 635–46. Springer-Verlag Berlin Heidelberg.

Heidorn, G., Jensen, K., Miller, L., Byrd, R., and Chodorow, M. 1982. The EPISTLE text-critiquing system. *IBM Systems Journal* **21**(3).

Hulden, M. 2009a. *Finite-State Machine Construction Methods and Algorithms for Phonology and Morphology*. Ph. D. thesis, University of Arizona.

Hulden, M. 2009b. Foma: a finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguitics*, Athens, Greece, pp. 29–32.

Karlsson, F., Voutilainen, A., Heikkila, J., and Anttila, A. 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**: 407–21.

Karttunen, L., Gaál, T., and Kempe, A. 1997. Xerox Finite State Tool. Manual, Xerox Research Centre Europe, Grenoble, Maylan, France.

Koskenniemi, K. 1983. *Two-level Morphology: A General Computational Model for Word-form Recognition and Production*. Helsinki, Finland: University of Helsinki.

Kukich, K. 1992. Techniques for automatically correcting words in text. *ACM Computing Surveys* **24**(4): 377–439.

Laporte, E. 1998. Lexical disambiguation for fine-grained tagsets. In J. G. *et al*. (ed.), *The Tbilisi Symposium in Logic, Language and Computation: Selected Papers*, Studies in Logic, Language and Information, pp. 203–10. Cambridge, UK: Cambridge University Press; and Stanford, CA: CSLI and FoLLI.

Mangu, L., and Brill, E. 1997. Automatic rule acquisition for spelling correction. In *ICML '97: Proceedings of the 14th International Conference on Machine Learning*, San Francisco, CA, USA, pp. 187–94. Morgan Kaufmann.

Min, K., and Wilson, W. H. 1998. Integrated control of chart items for error repair. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics(ACL-COLING 1998)*, pp. 862–68.

Mohri, M., Pereira, F. C. N., and Riley, M. 2000. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science* **231**(1): 17–32.

Otaegi, M. 2006. Datak, orduak eta zenbakiak euskaraz. Technical report, University of the Basque Country, Donostia-San Sebastin, Spain.

Schmid, H. 2005. A programming language for finite-state transducers. In A. Yli-Jyrä, L. Karttunen, and J. Karhumäki (eds.), *Proceedings of Finite-State Methods and Natural Language Processing (FSMNLP 2005)*, Volume 4002. Springer-Verlag, Berlin Heidelberg.

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press

Sjöbergh, J., and Knutsson, O. 2005. Faking errors to avoid making errors: very weakly supervised learning for error detection in writing. In *Proceedings of Recent Advances in Natural Language Processing (RANLP 2005)*, Borovets, Bulgaria, pp. 506–12.

Traboulsi, H. 2009. Arabic named entity extraction: a local grammar-based approach. In *Proceedings of the 2009 International Multiconference on Computer Science and Information Technology (IMCSIT 2009)*, Mragowo, Polland, pp. 139–43.

Wagner, J., and Foster, J. 2009. The effect of correcting grammatical errors on parse probabilities. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, Paris, France.

Weischedel, R., and Sondheimer, N. 1983. Meta-rules as a basis for processing Ill-formed input. *American Journal of Computational Linguistics* **9**(3–4): 161–77.

Zubiri, I. 1994. *Gramática Didáctica del Euskara*. Bilbo: Didaktiker.

https://doi.org/10.1017/S1351324911000088 Published online by Cambridge University Press