Seeing more than whitespace — Tokenisation and disambiguation in a North Sámi grammar checker

Linda Wiechetek **Kevin Brubeck Unhammer** **Sjur Nørstebø Moshagen**
UiT The Arctic University of Norway Trigram AS UiT The Arctic University of Norway
linda.wiechetek@uit.no kevin@trigram.no sjur.n.moshagen@uit.no

Abstract

Communities of lesser resourced languages like North Sámi benefit from language tools such as spell checkers and grammar checkers to improve literacy. Accurate error feedback is dependent on well-tokenised input, but traditional tokenisation as shallow preprocessing is inadequate to solve the challenges of real-world language usage. We present an alternative where tokenisation remains *ambiguous* until we have linguistic context information available. This lets us accurately detect sentence boundaries, multiwords and compound error detection. We describe a North Sámi grammar checker with such a tokenisation system, and show the results of its evaluation.

1 Introduction

Bilingual users frequently face bigger challenges regarding literacy in the lesser used language than in the majority language due to reduced access to language arenas (Outakoski, 2013; Lindgren et al., 2016). However, literacy and in particular writing is important in today's society, both in social contexts and when using a computer or a mobile phone. Language tools such as spellcheckers and grammar checkers therefore play an important role in improving literacy and the quality of written text in a language community.

North Sámi is spoken in Norway, Sweden and Finland by approximately 25,700 speakers (Simons and Fennig, 2018), and written in a number of institutions like the daily Sámi newspaper (*Ávvir*[1]), a few Sámi journals, websites and social media of the Sámi radio and TV (e.g. *YleSápmi*[2]). In addition, the Sámi parliaments, the national governments, and a Sámi university college produce North Sámi text.

An open-source spellchecker for North Sámi has been freely distributed since 2007 (Gaup et al., 2006).[3] However, a spellchecker is limited to looking only at one word contexts. It can only detect non-words, i.e. words that cannot be found in the lexicon. A grammar checker, however, looks at contexts beyond single words, and can correct misspelled words that are in the lexicon, but are wrong in the given context. In addition, a grammar checker can detect grammatical and punctuation errors.

A common error in North Sámi and other compounding languages is to spell compound words as separate words instead of one. The norm typically requires them to be written as one word, with the non-final components being in nominative or genitive case if they are nouns. This reflects a difference in meaning between two words written separately and the same two words written as a compound. Being able to detect and correct such compounding errors is thus important for the language community.

This paper presents and evaluates a grammar checker framework that handles ambiguous tokenisation, and uses that to detect compound errors, as well as improve sentence boundary detection after abbreviations and numeral expressions. The framework is completely open source, and completely rule-based. The evaluation is done manually, since gold standards for North Sámi tokenisation have not been developed prior to this work.

2 Background

The system we present is part of a full-scale grammar checker (Wiechetek, 2017, 2012). Before this work, there were no grammar checkers for the Sámi languages although some grammar checker-like work has been done in the language learning platform *Oahpa* (Antonsen, 2012). However, there have been several full-scale grammar checkers for

[1] https://avvir.no/ (accessed 2018-10-08)

[2] https://yle.fi/uutiset/osasto/sapmi/ (accessed 2018-10-08)

[3] In addition to that, there are syntactic disambiguation grammars, machine translators, dictionaries and a tagged searchable online corpus.

other Nordic languages, most of them implemented in the rule-based framework Constraint Grammar (CG). *Lingsoft* distributes grammar checkers for the Scandinavian languages,[4] some of which are or have been integrated into *MS Word*; a stand-alone grammar checker like *Grammatifix* (Arppe, 2000) is also available from *Lingsoft*. Another widely used, mostly rule-based and free/open-source system is *LanguageTool* (Milkowski, 2010), though this does not yet support any Nordic languages. Other CG-based checkers are *OrdRet* (Bick, 2006) and *DanProof* (Bick, 2015) for Danish.

2.1 Framework

The central tools used in our grammar checker are *finite state transducers* (FST's) and *CG* rules. CG is a rule-based formalism for writing disambiguation and syntactic annotation grammars (Karlsson, 1990; Karlsson et al., 1995). The *vislcg3* implementation[5] we use also allows for dependency annotation. CG relies on a bottom-up analysis of running text. Possible but unlikely analyses are discarded step by step with the help of morpho-syntactic context.

All components are compiled and built using the *Giella* infrastructure (Moshagen et al., 2013). This infrastructure helps linguists coordinate resource development using common tools and a common architecture. It also ensures a consistent build process across languages, and makes it possible to propagate new tools and technologies to all languages within the infrastructure. That is, the progress described in this paper is immediately available to all languages in the *Giella* infrastructure, barring the necessary linguistic work.

The North Sámi CG analysers take morphologically ambiguous input, which is the output from analysers compiled as FST's. The source of these analysers is written in the Xerox *twolc*[6] and *lexc* (Beesley and Karttunen, 2003) formalisms, compiled and run with the free and open source package HFST (Lindén et al., 2011).

We also rely on a recent addition to HFST, *hfst-pmatch* (Hardwick et al., 2015) (inspired by Xerox pmatch (Karttunen, 2011)) with the runtime tool *hfst-tokenise*. Below we describe how this lets us analyse and tokenise in one step, using FST's to identify regular words, multiword expressions and potential compound errors.

It should be noted that the choice of rule-based technologies is not accidental. The complexity of the languages we work with, and the general sparsity of data, makes purely data-driven methods inadequate. Additionally, rule-based work leads to linguistic insights that feed back into our general understanding of the grammar of the language. We chose a Constraint Grammar rule-based system since it is one we have long experience with, and it has proven itself to be competitive both in high- and low-resource scenarios. For example, *DanProof* (Bick, 2015, p.60) scores more than twice that of *Word2007* on the F1 measure (72.0% vs 30.1%) for Danish grammar checking. CG also compares favourably to modern deep learning approaches, e.g. *DanProof*'s F0.5 (weighting precision twice as much as recall) score is 80.2%, versus the 72.0% reported by Grundkiewicz and Junczys-Dowmunt (2018).

In addition, most current approaches rely very much on large-scale manually annotated corpora,[7] which do not exist for North Sámi. It makes sense to reuse large already existing corpora for training language tools. However, in the absence of these, it is more economical to write grammars of hand-written rules that annotate a corpus linguistically and/or do error detection/correction. As no other methods for developing error detection tools exist for North Sámi or similar languages in comparable situations (low-resourced in terms of annotated corpus, weak literacy, higher literacy in the majority languages), it is impossible for us to provide a comparison with other technologies.

2.2 Motivation

This section describes some of the challenges that lead to the development of our new grammar checker modules.

A basic feature of a grammar checker is to correct spelling errors that would be missed by a spell checker, that is, orthographically correct words that are nevertheless wrong in the given context.

(1) Beroštupmi **gáktegoarrun|gursii**
interest costume.sewing|course.ILL
'An interest in a costume sewing course'

[4] http://www2.lingsoft.fi/doc/swegc/errtypes.html (accessed 2018-10-08)

[5] http://visl.sdu.dk/constraint_grammar.html (accessed 2018-10-08), also Bick and Didriksen (2015)

[6] Some languages in the Giella infrastructure describe their morphophonology using Xfst rewrite rules; both *twolc* and *rewrite rules* are supported by the Giella infrastructure.

[7] "Automatic grammatical error correction (GEC) progress is limited by corpora available for developing and evaluating systems." (Tetreault et al., 2017, p.229)

In the North Sámi norm, generally (nominal) compounds are written as one word; it is an error to insert a space at the compound border. Ex. (1) marks the compound border with a pipe.

(2) *Beroštupmi **gáktegoarrun gursii**

If the components of a compound are separated by a space as in ex. (2) (cf. the correct spelling in ex (1)), the grammar checker should detect a compound spacing error.

Compound errors can not be found by means of a non-contextual spellchecker, since adjacent nominals are not automatically compound errors. They may also have a syntactic relation. Our lexicon contains both the information that *gáktegoarrun-gursii* would be a legal compound noun if written as one word, and the information needed to say that *gáktegoarrun gursii* may have a syntactic relation between the words, that is, they are independent tokens each with their own analysis.[8] We therefore assume ambiguous tokenisation. In order to decide which tokenisation is the correct one, we need context information.

In addition, there is the issue of combinatorial explosion. For example, the bigram *guhkit áiggi* 'longer time' may be a compound error in one context, giving an analysis as a single noun token. But it is also ambiguous with *sixteen* two-token readings, where the first part may be adjective, adverb or verb. We want to include these as alternative readings.

A naïve solution to getting multiple, ambiguous tokenisations of a string like *guhkit áiggi* would be to insert an optional space in the compound border in the entry for dynamic compounds, with an error tag. But if we analyse by longest match, the error reading would be the only possible reading. We could make the error tag on the space be optional, which would make the entry ambiguous between adjective+noun and compound, but we'd still be missing the adverb/verb+noun alternatives, which do not have a compound border between them. To explicitly encode all correct alternatives to compound errors in the lexicon, we would need to enter readings for e.g. verb+noun bigrams simply because they happen to be ambiguous with an error reading of a nominal compound.

Manually adding every bigram in the lexicon that happens to be ambiguous with an error would be extremely tedious and error-prone. Adding it automatically through FST operations turns out to quickly exhaust memory and multiply the size of the FST. Our solution would need to avoid this issue.

(3) **omd.** sámeskuvllas
for.example Sámi.school.LOC
'for example in the Sámi school'

(4) **omd.** Álttás sámeskuvllas
for.example Alta.LOC Sámi.school.LOC
'for example in Alta in the Sámi school'

In the fragment in ex. (3)–(4) above, the period after the abbreviation *omd.* 'for example' is ambiguous with a sentence boundary. In the first example, we could use the local information that the noun *sámeskuvllas* 'Sámi school (Loc.)' is lowercase to tell that it is not a sentence boundary. However, the second sentence has a capitalised proper noun right after *omd.* and the tokenisation is less straightforward. We also need to know that, if it is to be two tokens instead of one, the form splits before the period, and the tags belonging to `"<omd>"` go with that form, and the tags belonging to `"<.>"` go with that form. That is, we need to keep the information of which substrings of the form go with which readings of the whole, ambiguously-tokenised string.

As this and the previous examples show, we need context information to resolve the ambiguity; this means we need to *defer the resolution of ambiguous tokenisation* until after we have some of the morphological/syntactic/semantic analysis available.

(5) **Itgo** don muitte
not.SG2.Q you remember
'Don't you remember'

(6) **It** **go** don muitte
not.SG2 Q you remember
'Don't you remember'

Ex. (5) and (6) above are equivalent given the context – the space is just a matter of style, when used in this sense – but *go* appearing as a word on its own is locally ambiguous, since the question particle *go* may in other contexts be a conjunction (meaning 'when, that'). We want to treat *Itgo* 'don't you' as two tokens *It*+*go*; having equal analyses for the equal alternatives (after disambiguation) would simplify further processing. This can be encoded in the lexicon as one entry which we might be able to

[8] The non-head noun sometimes has an epenthetic only when used as a compound left-part, information which is also encoded in the lexicon.

split with some postprocessing, but before the current work, our tools gave us no way to show what parts of the form corresponded to which tokens. A typical FST entry (here expanded for simplicity) might contain

```
ii+V+Sg2+TOKEN+go+Qst:itgo
```

Now we want to encode that the form splits between 'it' and 'go', and that 'ii+V+2Sg' belongs to 'it', and that 'go+Qst' belongs to 'go'. But inserting a symbol into the form would mean that the form no longer analyses; we need to somehow mark the split-point.

Our system solves all of the above issues – we explain how below.

3 Method

Below we present our grammar checker pipeline, and our method to analyse and resolve ambiguous tokenisation. We first describe the system architecture of the North Sámi grammar checker, then our morphological analysis and tokenisation method, and finally our method of finding errors by disambiguating ambiguous tokenisation.

3.1 System architecture

The North Sámi grammar checker consists of different modules that can be used separately or in combination, cf. Figure 1.

The text is first tokenised and morphologically analysed by the descriptive morphological analyser *tokeniser-gramcheck-gt-desc.pmhfst*, which has access to the North Sámi lexicon with both error tags and lexical semantic tags. The following step, *analyser-gt-whitespace.hfst*, detects and tags whitespace errors. It also tags the first words of paragraphs and other whitespace delimited boundaries, which can then be used by the boundary detection rules later on, which enables detecting e.g. headers based on their surrounding whitespace.

The valency annotation grammar *valency.cg3* adds valency tags to potential governors. Then follows the module that disambiguates ambiguous tokenisation, *mwe-dis.cg3*, which can select or remove compound readings of multi-word expressions based on the morpho-syntactic context and valencies. It can also decide whether punctuation is a sentence boundary or not. The next module, *divvun-cgspell*, takes unknown words and runs them through our spell checker, where suggestions include morphological analyses.

The next module is the CG grammar *grc-disambiguator.cg3*, which performs morpho-syntactic analysis and disambiguation, except for the speller suggestions, which are left untouched. The disambiguator is followed by a CG module, *spellchecker.cg3*, which aims to reduce the suggestions made by the spellchecker by means of the grammatical context. The context is now partly disambiguated, which makes it easier to decide which suggestions to keep, and which not.[9]

The last CG module is *grammarchecker.cg3*, which performs the actual error detection and correction – mostly for other error types than spelling or compound errors. The internal structure of *grammarchecker.cg3* is more complex; local case error detection takes place after local error detection, governor-argument dependency analysis, and semantic role mapping, but before global error detection.

Finally, the correct morphological forms are generated from tag combinations suggested in *grammarchecker.cg3* by means of the normative morphological generator *generator-gt-norm.hfstol*, and suggested to the user along with a short feedback message of the identified error.

3.2 Ambiguous tokenisation

A novel feature of our approach is the support for different kinds of ambiguous tokenisation in the analyser, and how we disambiguate ambiguous tokens using CG rules.

We do tokenisation as part of morphological analysis using the *hfst-tokenise* tool, which does a left-to-right longest match analysis of the input, where matches are those given by a *pmatch* analyser. This kind of analyser lets us define tokenisation rules such as "a word from our lexicon may appear surrounded by whitespace or punctuation". The *pmatch* analyser imports a regular lexical transducer, and adds definitions for whitespace, punctuation and other tokenisation hints; *hfst-tokenise* uses the analyser to produce a stream of tokens with their morphological analysis in CG format.

As an example, *hfst-tokenise* will turn the input *ii, de* 'not, then' into three CG cohorts:

[9] In later work done after the submission, we tried using *grc-disambiguator.cg3* again after applying *spellchecker.cg3*, this time allowing it to remove speller suggestions. Given that the context was now disambiguated, and problematic speller suggestion cases had been handled by *spellchecker.cg3*, it disambiguated the remaining speller suggestions quite well, and left us with just one or a few correct suggestions to present to the user.

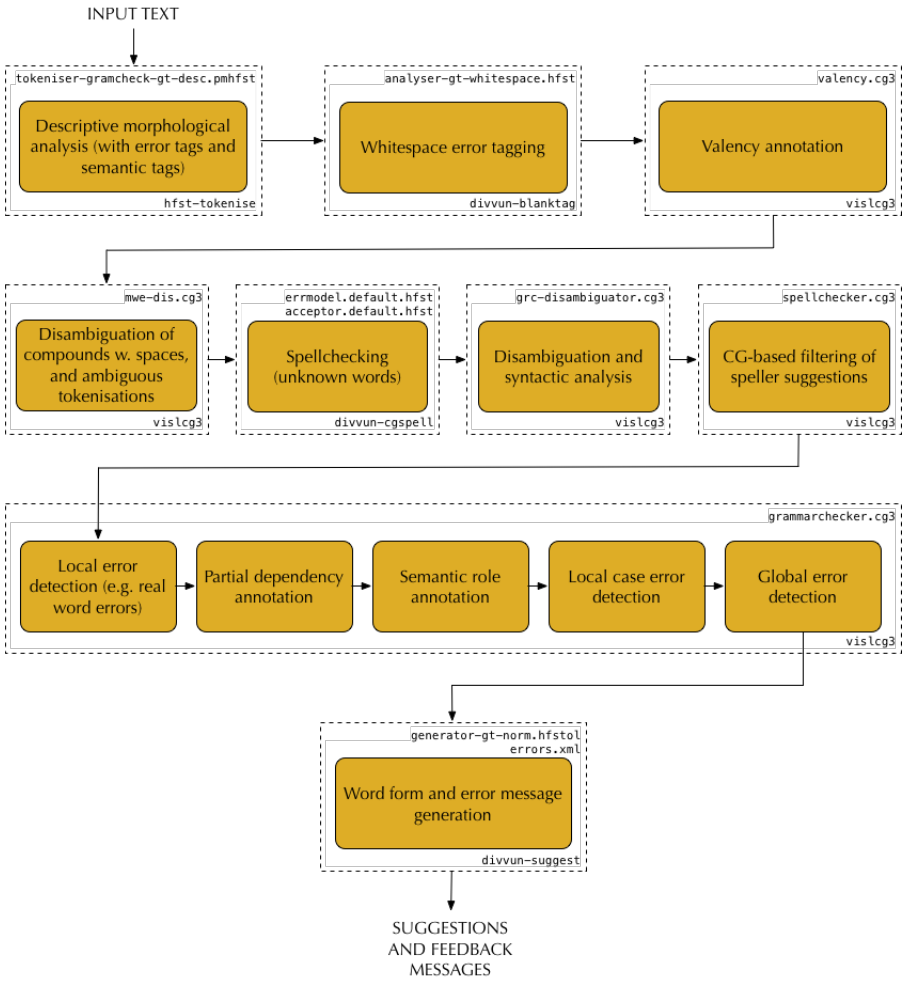

Figure 1: System architecture of the North Sámi grammarchecker

```
"<ii>"
        "ii" V IV Neg Ind Sg3 <W:0>
"<,>"
        "," CLB <W:0>
:
"<de>"
        "de" Adv <W:0>
        "de" Pcle <W:0>
```

The space between the words is printed after the colon. The analyses come from our lexical transducer.

```
Define morphology @bin"analyser.hfst" ;
Define punctword  morphology &
                  [ Punct:[?*] ] ;
Define blank      Whitespace |
                  Punct ;
Define morphoword morphology
                  LC([blank | #])
                  RC([blank | #]) ;
regex [ morphoword | punctword ];
```

The above *pmatch* rules say that a word from the lexicon (*analyser.hfst*) has to be surrounded by a "blank", where a blank is either whitespace or punctuation. The `LC/RC` are the left and right context conditions. We also extract (intersect) the subset of the lexicon where the form is punctuation, and allow that to appear without any context conditions.

We insert *re-tokenisation* hints in the lexicon at places where we assume there is a possible tokenisation border, and our changes to *hfst-tokenise* let the analyser backtrack and look for other tokenisations of the same input string. That is, for a given longest match tokenisation, we can force it to *redo* the tokenisation so we get other multi-token readings with shorter segments alongside the longest match. This solves the issue of combinatorial explosion.

As a simple example, the ordinal analysis of *17.* has a backtracking mark between the number and the period. If the lexicon contains the symbol-pairs/arcs:

```
1:1 7:7 ϵ:@PMATCH_BACKTRACK@
ϵ:@PMATCH_INPUT_MARK@ .:A ϵ:Ord
```

then, since the form-side of this analysis is `17.`, the input *17.* will match, but since there was a backtrack-symbol, we trigger a retokenisation. The input-mark symbol says where the form should be split.[10] Thus we also get analyses of *17* and . as two separate tokens.

```
"<17.>"
        "17" A Ord Attr
        "." CLB "<.>"
                "17" Num "<17>"
```

To represent tokenisation ambiguity in the CG format, we use *vislcg3 subreadings*,[11] where deeper (more indented) readings are those that appeared first in the stream, and any reading with a word-form-tag (`"<.>"` above) should (if chosen by disambiguation) be turned into a cohort of its own. Now we may run a regular CG rule to pick the correct reading based on context, e.g. `SELECT (".") IF (1 some-context-condition) ...;` which would give us

```
"<17.>"
        "." CLB "<.>"
                "17" Num "<17>"
```

Then a purely mechanical reformatter named *cg-mwesplit* turns this into separate tokens, keeping the matching parts together:

```
"<17>"
        "17" Num
"<.>"
        "." CLB
```

We also handle possible compound errors with the above scheme. When compiling the lexical transducer, we let all compound boundaries optionally be realised as a space. Two successive nouns like *illu sáhka* 'happiness news (i.e. happy news)' will be given a compound analysis which includes an error tag. We also insert a backtracking symbol with the space, so that the tokenisation tool knows that the compound analysis is not necessarily the only one (but without having to explicitly list all possible alternative tokenisations). If the retokenisation finds that the nouns can be analysed and tokenised independently, then those tokens and analyses are also printed.

```
"<illu sáhka>"
        "illusáhka" N Sg Nom Err/SpaceCmp
        "sáhka" N Sg Nom "< sáhka>"
                "illu" N Sg Nom "<illu>"
```

Given such an ambiguous tokenisation, CG rules choose between the compound error and the two-token readings, using context information from the rest of the sentence. If the non-error reading was chosen, we get:

```
"<illu sáhka>"
        "sáhka" N Sg Nom "< sáhka>"
                "illu" N Sg Nom "<illu>"
```

which *cg-mwesplit* reformats to two cohorts:

```
"<illu>"
        "illu" N Sg Nom
"< sáhka>"
        "sáhka" N Sg Nom
```

3.3 Rule-based disambiguation of ambiguous tokenisation

As mentioned above, disambiguation of ambiguous tokenisation is done after morphological analysis. Consequently, this step has access to undisambiguated morphological (but not full syntactical) information. In addition, lexical semantic tags and valency tags are provided. The rules that resolve sentence boundary ambiguity are based on transitivity tags of abbreviations, lexical semantic tags, and morphological tags. Some of them are specific to one particular abbreviation.

Bi- or trigrams given ambiguous tokenisation can either be misspelled compounds (i.e. in North Sámi typically two-part compounds are the norm) or two words with a syntactic relation. The assumption is that if a compound is lexicalised, two or more adjacent words may be analysed as a compound and receive an errortag (*Err/SpaceCmp*), using a CG rule such as the following:

```
SELECT SUB:* (Err/SpaceCmp) IF (NEGATE
    0/* Err/MissingSpace OR Ess);
```

This rule selects the error reading unless any subreading of this reading (`0/*`) already has another error tag or is an essive case form.

This is the case unless any other previously applied rule has removed the error reading. Version r172405 of the tokenisation disambiguation grammar *mwe-dis.cg3* has 40 REMOVE rules and 8 SELECT rules.

Compound errors are ruled out for example if the first word is in genitive case as it can be the first part of a compound but also a premodifier. The simplified CG rule below removes the compound

[10] This also means we cannot reshuffle the input/output side of the FST. In practice, we use a flag diacritic in the lexicon, which will keep its place during minimisation, and after the regular lexicon is compiled, we turn the flag into the `ϵ:@PMATCH_INPUT_MARK@` symbol-pair.

[11] https://visl.sdu.dk/cg3/chunked/subreadings.html (accessed 2018-10-10)

error reading if the first component is in genitive unless it receives a case error reading (nominative/accusative or nominative/genitive) or it is a lesser used possessive reading and a non-human noun. The rule makes use of both morphological and semantic information.

```
REMOVE (Err/SpaceCmp) IF (0/1 Gen -
    Allegro - Err/Orth-nom-acc - Err/
    Orth-nom-gen - PX-NONHUM);
```

(7) **Gaskavahku eahkeda**
Wednesday.GEN evening.ACC
'Wednesday evening'

(8) **áhpehis nissonolbmuid**
pregnant woman.ACC.PL
'pregnant women'

In ex. (7), *gaskavahku* 'Wednesday' is in genitive case. The context to rule out a compound error is very local. In ex. (8), *áhpehis* 'pregnant' the first part of the potential compound is an attributive adjective form. Also here compound errors are categorically discarded.

(9) Paltto lea **riegádan jagi** 1947
Paltto is born.PRFPRC year.ACC 1947
'Paltto was born in 1947'

(10) galggai buot báikkiin **dárogiella**
should all place.LOC.PL Norwegian.NOM
oahpahusgiellan
instructing.language in all places
'Norwegian had to be the instructing language'

Other cases of compound error disambiguation, however, are more global. In ex. (9), *riegádan jagi* 'birth year (Acc.)' is a lexicalized compound. However as it is preceded by a finite verb, which is also a copula, i.e. *lea* 'is', the perfect participle form *riegádan* 'born' is part of a past tense construction ('was born'), and the compound error needs to be discarded.

In ex. (10), on the other hand, the relation between the first part of the bigram (*dárogiella* 'Norwegian') and the second part (*oahpahusgiellan* 'instructing language (Ess.)') is that of a subject to a subject predicate. The disambiguation grammar refers to a finite copula (*galggai* 'should') preceding the bigram.

4 Evaluation

In this section we evaluate the previously described modules of the North Sámi grammar checker. Firstly, we evaluate the disambiguation of compound errors in terms of precision and recall. Then we compare our system for sentence segmentation with an unsupervised system. Since a corpus with correctly annotated compound and sentence boundary tokenisation for North Sámi is not available, all evaluation and annotation is done from scratch. We use the *SIKOR* corpus (SIKOR2016)),[12] a descriptive corpus which contains automatic annotations for linguistic research purposes, but no manually checked/verified tags. We selected a random corpus of administrative texts for two reasons. We had a suspicion that it would have many abbreviations and cases of ambiguous tokenisation. Secondly, administrative texts stand for a large percentage of the total North Sámi text body, and the genre is thus important for a substantial group of potential users of our programs.

4.1 Compound error evaluation

For the quantitative evaluation of the disambiguation of potential compound errors we calculated both precision (correct fraction of all marked errors) and recall (correct fraction of all errors). The corpus used contains 340,896 space separated strings, as reported by the Unix tool `wc`. The exact number of tokens will vary depending on tokenisation techniques, as described below.

The evaluation is based on lexicalised compounds as potential targets of ambiguous tokenisation. A previous approach allowed ambiguous tokenisation of dynamic compounds too, solely using syntactic rules to disambiguate. However, this led to many false positives (which would require more rules to avoid). Since our lexicon has over 110,000 lexicalised compounds (covering 90.5 % of the compounds in the North Sámi *SIKOR* corpus) coverage is acceptable without the riskier dynamic compound support.[13]

Table 1 contains the quantitative results of the compound error evaluation. Of the 340.895 running bigrams in the text, there were a total of 4.437 potential compound errors, i.e. 1.30 % of running bigrams are analysed as possible compounds by our lexicon. On manually checking, we found 458 of these to be true compound errors (0.13 % of running bigrams, or 10.3 % of potential compound errors as marked by the lexicon). So the table

[12] *SIKOR* contains a range of genres; the part used for evaluation contains bureaucratic texts.

[13] For less developed lexicons, the trade-off may be worth it.

True positives	360
False positives	110
True negatives	3,869
False negatives	98
Precision	76.6%
Recall	78.6%

Table 1: Qualitative evaluation of CG compound error detection

indicates how well our Constraint Grammar disambiguates compound errors from words that are supposed to be written apart, and tells nothing of the work done by the lexicon in selecting possible compound errors (nor of possible compound errors missed by the lexicon).[14]

Precision for compound error detection is well above the 67% threshold for any error type in a commercial grammar checker mentioned by Arppe (2000, p.17), and the F0.5 (weighting precision twice as much as recall) score is 77.0%, above e.g. Grundkiewicz and Junczys-Dowmunt (2018)'s 72.0%.[15]

False positives occur for example in cases where there is an internal syntactic structure such as in the case of ex. (11), where both *bálvalus* 'service' and *geavaheddjiide* 'user (Ill. Pl.)' are participants in the sentence's argument structure. Since there is no finite verb, the syntactic relation could only be identified by defining the valency of *bálvalus* 'service'.

(11) Buoret **bálvalus** **geavaheddjiide**
Better service.NOM.SG user.ILL.PL
'Better service to the users'

A number of the false negatives (cf. ex. (12)) are due to frequent expressions including *lágan* (i.e. *iešguđetlagan* 'different', *dánlágan* 'this kind of', etc.), which need to be resolved by means of an idiosyncratic rule. *Dan* and *iešguđet* are genitive or attributive pronoun forms and not typically part of a compound, so a syntactic rule only does not resolve the problem.

(12) *__iešguđet lágan__ molssaeavttut
different kinds alternative.PL
'Different kinds of alternatives'

(13) *__Láhka__ __rievdadusaid__
law.NOM;lacquer.GEN changing.ACC.PL
birra
about
'About the law alterations'

In ex. (13), there is a compound error. However, one of the central rules in the tokeniser disambiguation grammar removes the compound error reading if the first part of the potential compound is in the long genitive case form. However, in this case *láhka* can be both the genitive form of *láhkka* 'lacquer' and the nominative form of *láhka* 'law'. This unpredictable lexical ambiguity had not been taken into account by the disambiguation rule, which is why there is a false negative. In the future it can be resolved by referring to the postposition *birra* 'about', which asks for a preceding genitive.

4.2 Sentence boundary evaluation

A common method for splitting sentences in a complete pipeline (used e.g. by *LanguageTool*) is to tokenise first, then do sentence splitting, followed by other stages of linguistic analysis. Here a standalone tokeniser would be used, e.g. PUNKT (Kiss and Strunk, 2006), an unsupervised model that uses no linguistic analysis, or GATE[16] which uses regex-based rules. The Python package SpaCy[17] on the other hand trains a supervised model that predicts sentence boundaries jointly with dependency structure. Stanford CoreNLP[18] uses finite state automata to tokenise, then does sentence splitting.

In contrast, our method uses no statistical inference. We tokenise as the first step, but the tokenisation remains ambiguous until part of the linguistic analysis is complete.

Below, we make a comparison with PUNKT[19], which, although requiring no labelled training data, has been reported[20] to perform quite well compared to other popular alternatives.

As with the above evaluation, we used bureaucratic parts of the *SIKOR* corpus. We trained the PUNKT implementation that comes with NLTK on

[14] We have also not calculated the number of actual compounds in the evaluation corpus, so the ratio of compound errors to correct compounds is unknown.

[15] We would like to compare performance on this task with a state-of-the-art machine learning method, but have seen no references for this particular task to use as an unbiased baseline. However, the gold data set that was developed for evaluating our method is freely available to researchers who would like to experiment with improving on the results.

[16] http://gate.ac.uk/ (accessed 2018-10-08)

[17] https://spacy.io/ (accessed 2018-10-08)

[18] http://www-nlp.stanford.edu/software/corenlp.shtml (accessed 2018-10-08)

[19] https://www.nltk.org/_modules/nltk/tokenize/punkt.html (accessed 2018-10-08)

[20] https://tech.grammarly.com/blog/how-to-split-sentences (accessed 2018-10-08)

System	PUNKT	Divvun
True pos.	1932	1986
False pos. (split mid-sent)	39	29
True neg.	474	484
False neg. (joined sents)	55	1
Precision	98.02%	98.56%
Recall	97.23%	99.95%

Table 2: Sentence segmentation errors per system on 2500 possible sentences.[22]

287.516 "words" (as counted by `wc`), and manually compared the differences between our system (named *divvun* below) and PUNKT. We used a trivial `sed` script `s/[.?:!] */&\n/g` to create a "baseline" count of possible sentences, and ran the evaluation on the first 2500 potential sentences given by this script (as one big paragraph), counting the places where the systems either split something that should have been one sentence, or treated two sentences as one; see table 2.

Of the differences, we note that PUNKT often treats abbreviations like *nr* or *kap.* as sentence boundaries, even if followed by lower-case words or numbers (*st. meld. 15* as three sentences). Our system sometimes makes this mistake too, but much more rarely. Also, PUNKT never treats colon as sentence boundaries. The colon in Sámi is used for case endings on names, e.g. *Jönköping:s*, but of course also as a clause or sentence boundary. Thus many of the PUNKT errors are simply *not* marking a colon as a sentence boundary. On the other hand, our system has some errors where an unknown word led to marking the colon (or period) as a boundary. This could be fixed in our system with a simple CG rule.

There are also some odd cases of PUNKT not splitting on period even with following space and title cased word, e.g. *geavahanguovlluid. Rádjegeassin*. Where the baseline *sed* script creates the most sentence boundaries in our evaluation test set (2500), our system creates 2015 sentences, and *PUNKT* 1971.

Our system is able to distinguish sentence boundaries where the user forgot to include a space, e.g. *buorrin.Vuoigatvuođat* is correctly treated as a sentence boundary. This sort of situation is hard to distinguish in general without a large lexicon. Our system does make some easily fixable errors, e.g. *kap.1* was treated as a sentence boundary due to a wrongly-written CG rule (as such, this evaluation has been helpful in uncovering silly mistakes). Being a rule-based system, it is easy to support new contexts when required.

5 Conclusion

We have introduced the North Sámi grammar checker presenting its system architecture and described its use and necessity for the North Sámi language community. Tokenisation is the first step in a grammar checker when approaching frequent spelling error types that cannot be resolved without grammatical context. We are questioning the traditional concept of a token separated by a space, not only in terms of multiwords, but also in terms of potential compound errors. Our experiment showed that our system outperforms a state-of-the-art unsupervised sentence segmenter. Disambiguation of compound errors and other two-word combinations give good results both in terms of precision and recall, i.e. both are above 76%. Our method of ambiguous tokenisation and ambiguity resolution by means of grammatical context allows us to improve tokenisation significantly compared to the standard approaches. The integration of the grammar checker framework in the *Giella* infrastructure ensures that this approach to tokenisation is directly available to all other languages using this infrastructure.

Acknowledgments

We especially would like to thank Thomas Omma for testing rules and checking examples within the above discussed modules, and our colleagues in *Divvun* and *Giellatekno* for their daily contributions to our language tools and the infrastructure.

References

Lene Antonsen. 2012. Improving feedback on l2 misspellings - an fst approach. In *Proceedings of the SLTC 2012 workshop on NLP for CALL; Lund; 25th October; 2012*, 80, pages 1–10. Linköping University Electronic Press; Linköpings universitet.

Antti Arppe. 2000. Developing a grammar checker for Swedish. In *Proceedings of the 12th Nordic Conference of Computational Linguistics (NoDaLiDa 1999)*, pages 13–27, Department of Linguistics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Studies in Computational Linguistics. CSLI Publications, Stanford.

Eckhard Bick. 2006. A constraint grammar based spellchecker for Danish with a special focus on dyslexics. In Mickael Suominen, Antti Arppe, Anu Airola, Orvokki Heinämäki, Matti Miestamo, Urho Määttä, Jussi Niemi, Kari K. Pitkänen, and Kaius Sinnemäki, editors, *A Man of Measure: Festschrift in Honour of Fred Karlsson on his 60th Birthday*, volume 19/2006 of *Special Supplement to SKY Jounal of Linguistics*, pages 387–396. The Linguistic Association of Finland, Turku.

Eckhard Bick. 2015. DanProof: Pedagogical spell and grammar checking for Danish. In *Proceedings of the 10th International Conference Recent Advances in Natural Language Processing (RANLP 2015)*, pages 55–62, Hissar, Bulgaria. INCOMA Ltd.

Eckhard Bick and Tino Didriksen. 2015. CG-3 – beyond classical Constraint Grammar. In *Proceedings of the 20th Nordic Conference of Computational Linguistics (NoDaLiDa 2015)*, pages 31–39. Linköping University Electronic Press, Linköpings universitet.

Børre Gaup, Sjur Moshagen, Thomas Omma, Maaren Palismaa, Tomi Pieski, and Trond Trosterud. 2006. From Xerox to Aspell: A first prototype of a north sámi speller based on twol technology. In *Finite-State Methods and Natural Language Processing*, pages 306–307, Berlin, Heidelberg. Springer Berlin Heidelberg.

Roman Grundkiewicz and Marcin Junczys-Dowmunt. 2018. Near human-level performance in grammatical error correction with hybrid machine translation. *arXiv preprint arXiv:1804.05945*.

Sam Hardwick, Miikka Silfverberg, and Krister Lindén. 2015. Extracting semantic frames using hfstpmatch. In *Proceedings of the 20th Nordic Conference of Computational Linguistics, (NoDaLiDa 2015)*, pages 305–308.

Fred Karlsson. 1990. Constraint Grammar as a Framework for Parsing Running Text. In *Proceedings of the 13th Conference on Computational Linguistics (COLING 1990)*, volume 3, pages 168–173, Helsinki, Finland. Association for Computational Linguistics.

Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila. 1995. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter, Berlin.

Lauri Karttunen. 2011. Beyond morphology: Pattern matching with FST. In *SFCM*, volume 100 of *Communications in Computer and Information Science*, pages 1–13. Springer.

Tibor Kiss and Jan Strunk. 2006. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

Krister Lindén, Miikka Silfverberg, Erik Axelson, Sam Hardwick, and Tommi Pirinen. 2011. Hfst—framework for compiling and applying morphologies. In Cerstin Mahlow and Michael Pietrowski, editors, *Systems and Frameworks for Computational Morphology*, volume Vol. 100 of *Communications in Computer and Information Science*, pages 67–85. Springer-Verlag, Berlin, Heidelberg.

Eva Lindgren, Kirk P H Sullivan, Hanna Outakoski, and Asbjørg Westum. 2016. Researching literacy development in the globalised North: studying trilingual children's english writing in Finnish, Norwegian and Swedish Sápmi. In David R. Cole and Christine Woodrow, editors, *Super Dimensions in Globalisation and Education*, Cultural Studies and Transdiciplinarity in Education, pages 55–68. Springer, Singapore.

Marcin Milkowski. 2010. Developing an open-source, rule-based proofreading tool. *Softw., Pract. Exper.*, 40(7):543–566.

Sjur N. Moshagen, Tommi A. Pirinen, and Trond Trosterud. 2013. Building an open-source development infrastructure for language technology projects. In *NODALIDA*.

Hanna Outakoski. 2013. Davvisámegielat čálamáhtu konteaksta [The context of North Sámi literacy]. *Sámi dieđalaš áigečála*, 1/2015:29–59.

SIKOR2016. 2016-12-08. SIKOR UiT The Arctic University of Norway and the Norwegian Saami Parliament's Saami text collection. **URL:** *http://gtweb.uit.no/korp (Accessed 2016-12-08)*.

Gary F. Simons and Charles D. Fennig, editors. 2018. *Ethnologue: Languages of the World*, twenty-first edition. SIL International, Dallas, Texas.

Joel R. Tetreault, Keisuke Sakaguchi, and Courtney Napoles. 2017. JFLEG: A fluency corpus and benchmark for grammatical error correction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, pages 229–234.

Linda Wiechetek. 2012. Constraint Grammar based correction of grammatical errors for North Sámi. In *Proceedings of the Workshop on Language Technology for Normalisation of Less-Resourced Languages (SALTMIL 8/AFLAT 2012)*, pages 35–40, Istanbul, Turkey. European Language Resources Association (ELRA).

Linda Wiechetek. 2017. *When grammar can't be trusted – Valency and semantic categories in North Sámi syntactic analysis and error detection*. PhD thesis, UiT The arctic university of Norway.