Drawing Blue Lines – What can Constraint Grammar do for GEC?

Linda Wiechetek
Divvun/ UiT Norgga árktalaš universitehta
firstname.lastname@uit.no

Kevin Brubeck Unhammer
Trigram AS
firstname@trigram.no

Abstract

This paper presents the application of rule-based methods for Grammatical Error Correction (GEC) across multiple low-resource languages. We describe new functionality using the Constraint Grammar (CG) formalism, designed for detecting and correcting different types of complex grammatical errors in a range of morphologically complex languages. These errors require transformations such as reordering, word additions/deletions, and alternative choices for multiword suggestions. New perspectives are gained from end-to-end-testing – this work aims to clarify the relationship between the command-line interface used by developers and the user interfaces of our grammar checker plug-in for common word processors. We present challenges and solutions in correcting complex errors, with examples from languages like Lule Sámi, Irish, and Greenlandic, enabling linguists to adapt these methods in order to provide accurate and context-aware proofing tools for their own languages in mainstream word processors like Microsoft Word, Google Docs or LibreOffice.

1 Introduction

This work builds on Fred Karlsson's introduction to CG / Constraint Grammar (Karlsson, 1990b) (CG2), Eckhard Bick's higher doctoral dissertation (Bick, 2000) and Tino Didriksen's user manual for VISL CG3 (Didriksen, 2010). Those descriptions focus mainly on the use of the command line. However, things can look very different in popular word processing programs, where sentences are not displayed from top to bottom with each word's analysis. As programmers, the command line is our daily workplace and when things work we can easily forget that this is not the place where most writers see their texts displayed. The purpose of this article is to implement and describe how complex errors can be marked and displayed in text processing programs in a way that will be intuitive, useful and pedagogical to the writer. We will focus on proofing tool applications which are commonly used by writers, in particular grammar checkers that are substantially more complex than spellcheckers, not only from a linguistic, but also a technical point of view. The blue lines in MS Word typically stretch over more than one word, sometimes the whole sentence. Correction suggestions can include not only different spellings, but entirely different constructions. We also need ways to display correction feedback and possibilities to refer to all parts of the error in the text shown to the user.

Figure 1 illustrates how the *Divvun*[1] grammar checker works in MS Word. The text itself receives red lines for default English spellchecking, which any user of non-English text needs to turn off. The relevant grammar checking takes place in the right-hand side-bar since Microsoft does not let third-party software show feedback inside the document, even though they do not provide any alternative for e.g. Sámi speakers. Therefore the blue lines marking simple or – in this case – complex grammatical errors are found to the right of the document.

This article shows different types of complex errors in a number of languages, along with technical solutions within the CG-formalism, most of which came into being as a result of working with grammar checking. We aim to give a thorough description so that the full potential of it can be used by whoever would like to model a correction

[1] https://divvun.no/

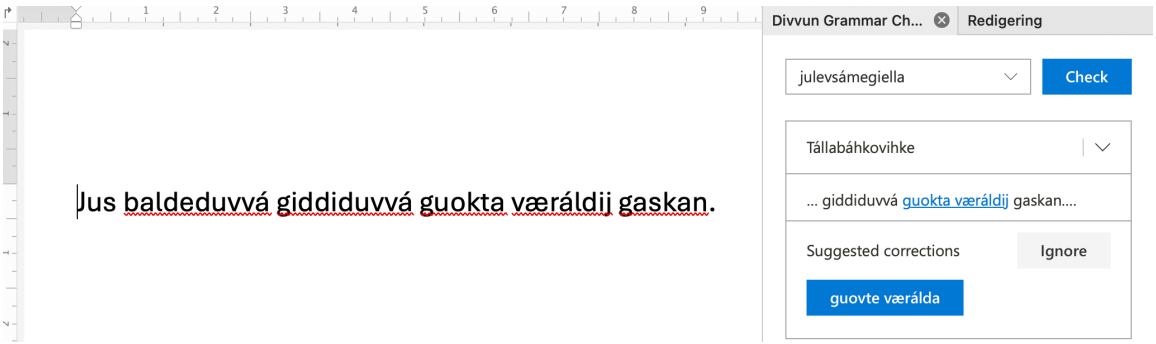

Figure 1: Divvun Grammar checking in MS Word

```
"<guokta>"
        "guokta" Num Sg Nom <W:0.0> &msyn-numphrase-sggen ID:4
        "guokta" Num <W:0.0> Sg Gen SUGGEST SUGGEST ID:4
guokta+Num+Sg+Gen       guovte
        "guokta" Num v2 Sg Acc <W:0.0> &msyn-numphrase-sggen ID:4
:
"<væráldij>"            guokta væráldij →  guovte værálda
        "várált" Area/NO N <smj> <smj> Sem/Plc Pl Gen <W:0.0> &msyn-numphrase-sggen ID:5 R:$2:4 R:LEFT:4
msyn-numphrase-sggen
        "várált" Area/NO N <smj> <smj> Sem/Plc <W:0.0> Sg Gen SUGGEST SUGGEST ID:5 R:$2:4 R:LEFT:4
várált+Area/NO+N+Sg+Gen várálda,várálda,várálda
:
"<gaskan>"
        "gaskan" Po <smj> <smj> <W:0.0>
"<.>"
        "." CLB <W:0.0>
```

Figure 2: Divvun Grammar checking from the developer's perspective on the command line

grammar of their language.[2]

2 Motivation

In our work with grammatical errors, we found that our formalism lacked certain functionalities required for error detection and correction. One of them was handling multiple possible corrections of a complex error, where we have to be careful that corrections are consistent, e.g. **as the tree grow* could be corrected to *as the **trees** grow* or *as the tree **grows*** but not **as the **trees grows***. We also need to be able to reorder, add or delete words.

A large portion of grammatical errors we have encountered are *complex*, in that they require marking up several words, and corrections may have to change several words at once. For example, in North Sámi, the biggest class of grammatical errors are compound errors, where two or more adjacent words should be written as one word. The second biggest class is subject-verb agreement errors, another complex error. In Lule Sámi, we have focused on NP-internal number and case agreement, where the most complex rules target numeral phrases. These behave differently from the majority language paradigm and are hence the cause of many errors for bilingual speakers/writers. This class is relevant for all Sámi languages. Gender agreement errors in noun phrases are common errors in Irish (and in a number of other Indo-European languages as well). Greenlandic has been the showcase for word order errors, but these also appear in for example Lule Sámi, a typical SOV language that is under influence from the SVO languages Swedish and Norwegian. Greenlandic (standard SOV), faces similar pressure since speakers are typically bilingual with Danish (standard SVO). The noun phrase internal word order differs as well; Greenlandic uses N A where Danish uses A N.

3 Background

3.1 Previous work

Unlike the common assumption that all language tasks are solved by machine learning applications, much grammar checking has had and still has a rule-based foundation, even in applications that do not say this outright.[3] Bick (2015) has made

[2]A technical reference for the system is available at `https://github.com/divvun/libdivvun`

[3]The authors have personally experienced that their own work on rule-based systems has been presented as "machine

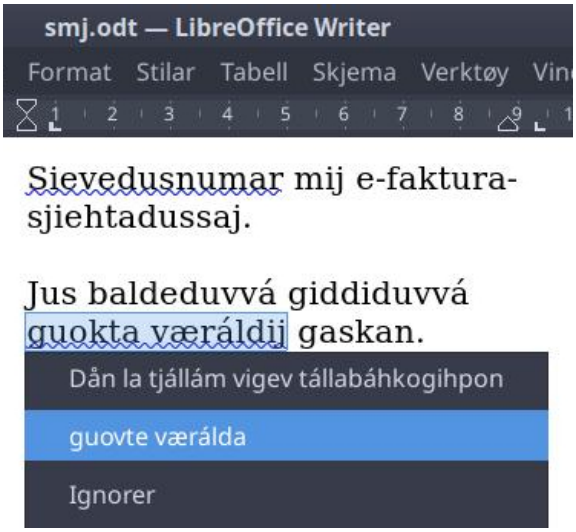

Figure 3: Divvun Grammar checking in LibreOffice (which gives writers autonomy over the choice of spelling/grammar checking providers)

several CG-based grammar checkers for Danish. *Grammatifix* (Arppe, 2000), based on CG, has been part of the Swedish MS Word since 2000. *Grammarly* is a grammar checker for English; little is published about their methods, but as Dyomkin (2015) reveals it was at that time using rules on dependency parse-trees.

3.2 Technical background

Below we describe a Constraint Grammar (CG) module for handling linguistic grammar errors. Constraint Grammar is a rule-based formalism originally developed by Karlsson (1990a); Karlsson et al. (1995). In our work, we use the free and open source implementation VISL CG-3 (Bick and Didriksen, 2015). This module is one component of a larger pipeline displayed in Figure 4. In this pipeline, the text is first tokenized and morphologically analysed with *finite-state automata* (Beesley and Karttunen, 2003) using the free and open-source toolkit HFST (Lindén et al., 2013), in particular `hfst-tokenise`, with morphologies from the Divvun/Giellatekno infrastructure (Wiechetek et al., 2022).

As described in Wiechetek et al. (2019), this step also parses consecutive words as possible compound errors (words written with extraneous spaces). A `divvun-blanktag` step adds useful tags like "start of sentence". The following Constraint Grammar tags words with valency classes, the next one disambiguates ambiguous multi-words. Then a purely mechanical step `cg-mwesplit` turns the disambiguated multiword analyses into separate CG cohorts.[4] A second `divvun-blanktag` step tags some whitespace errors. Then we run the spellchecker – this may add readings to previously unknown words. The new readings are also given valency tags by CG. Next we run a CG disambiguator and syntax tagger – this should ideally leave us with one reading per word. A second CG is used to filter out some unlikely spelling suggestions. Then we arrive at the focus of this article, the grammar checker CG (marked green in the Figure). This module contains the rules which tag grammatical errors, create suggestions and expand underlines.

The final module of the pipeline, `divvun-suggest`, uses the information given by the previous steps to look up word forms of suggestions, create readable error messages from error tags, and format it in a way that is readable to word processor plug-ins and other user interfaces.

The developer of grammar checker rules typically sees CG-formatted output when working with rules, which includes a lot of information that plug-ins do not need to see. The plug-ins need a programmatic interface which unambiguously tells them which words need underlines, and with what suggestions. This interface needs to use a format with good library support across programming languages. So `divvun-suggest` can output in two different formats: human-readable CG for the developer, and the popular data interchange format `JSON` for the plug-ins. The `JSON` format is used to communicate suggestions to plug-ins for LibreOffice, MS Word, Google Docs, web sites and newspaper publishing systems.

4 Building blocks of grammar checker rules

Before we get into the complex error examples, we give an introduction to the way grammar checker rules are written in our system. Here we describe the necessary components required to *tag* something as an error, to *suggest* a correction, to *underline* the related words of a context and to *refer* to the different parts of an error in an explanation.

We will here write a simplified rule for the common Norwegian Nynorsk agreement error in ex-

learning" or "AI" for press releases.

[4] A cohort in CG is a wordform with all its possible readings.

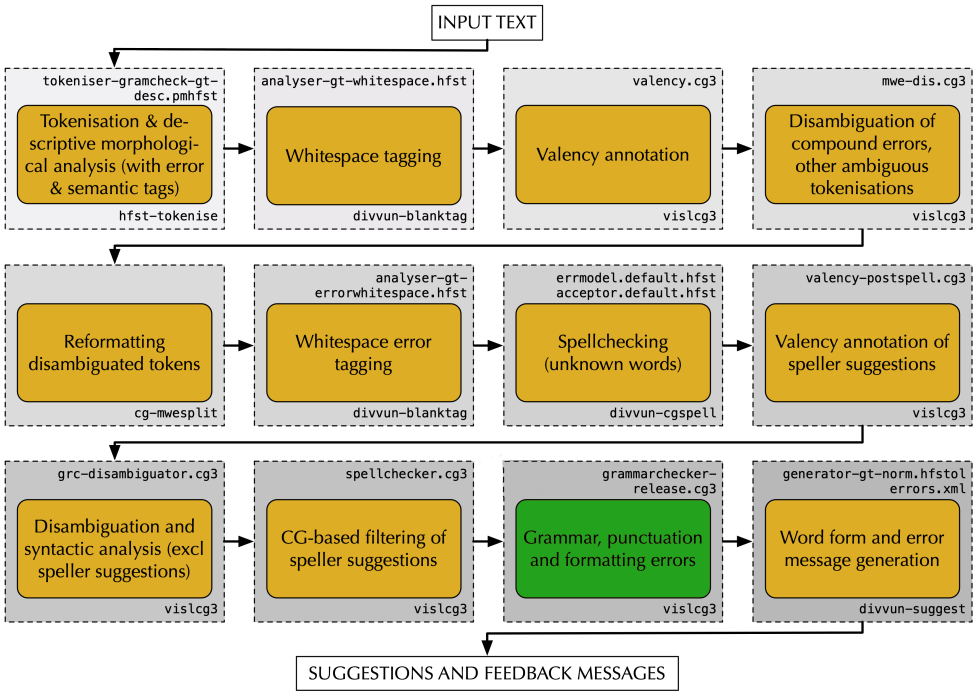

Figure 4: Modular structure of the grammar checkers

ample (1), where the wrong determiner gender is used, with correction in example (2):

(1) *å føre **eit** rekneskap
to keep an.NT account.M

(2) å føre ein rekneskap
to keep an.M account.M
'to keep accounts'

To tag this as an error, we could target the neuter determiner and give it a tag like `&det-n-nt/m-agr` if it precedes a masculine noun with the following rule, simplified for this example:

```
ADD (&det-n-nt/m-agr)
 TARGET (det nt)
 IF (1 (n m)) (NEGATE 1 (n nt)) ;
```

Tags beginning with `&` are interpreted as error tags by the `divvun-suggest` module. This rule alone will ensure the determiner is underlined in the user's word processor, but does not yet lead to a suggestion. To suggest the form 'ein', we can create a new reading with the correct tags belonging to that form:

```
COPY (m SUGGEST) EXCEPT (nt)
 TARGET (&det-n-nt/m-agr) ;
```

The special tag `SUGGEST` indicates to `divvun-suggest` that this is a suggestion; given the right lemma and tags, the morphological generator will give the correct form for readings tagged `SUGGEST`. With this in place, the user will also see a drop-down with the suggestion 'ein'.

We may also want to extend the underline to the noun, to indicate the relevant context of the error. We do this by adding a relation `RIGHT` to the noun:

```
ADDRELATION (RIGHT) (&det-n-nt/m-agr)
 TO (1 (n)) ;
```

This will both extend the underline to include the noun, and ensure suggestions will include the noun, so *'eit rekneskap' gets the suggestion 'ein rekneskap'.

The relation is called `RIGHT` since we're extending the right-hand end of the underline. We can also add a `LEFT` relation if we want to extend the underline to the left of the main error cohort (the word with the `&` tag).

Plug-ins may show helpful explanations for why something is considered an error. These explanations are written in the file `errors.xml`, and may refer to word of the main error cohort with the string `$1` – the module `divvun-suggest` will replace `$1` with the corresponding word form (here 'eit'). To refer to another word in an explanation, we can

add the `$2` (or `$3` etc.) relation to it in the same way as `RIGHT`, and then use `$2` in the XML description (which here would be replaced with 'rekneskap'), turning a template message like `'$1' is neuter, but '$2' is masculine` into *'eit' is neuter, but 'rekneskap' is masculine* in the plugin.

5 Adding particles in Lule Sámi

Some times we need to add new words in corrections, for example missing particles:

(3) *Boahtá sån sijddaj?
come.PRS.3SG PRON.3SG siida.ILL

(4) Boahtá gus sån
come.PRS.3SG PCLE.QST PRON.3SG
sijddaj?
siida.ILL
'Is s/he coming home?'

To achieve this, we use the regular CG feature `ADDCOHORT`, along with a tag `&ADDED` to signify to `divvun-suggest` that this cohort did not exist in the input. Since the word did not exist in the input, we put the error tag on the preceding word, and give that a relation `RIGHT` to the added word.[5] The user will then see a correction from 'Boahtá' to 'Boahtá gus'.

6 Linking errors to each other in Irish

Errors may be spread across several words that need to be fixed as one. We may have a lone noun with the wrong case, but if it has an article in front, both may be wrong and need to be fixed.

Consider the Irish phrase below, where (5) has two words that need changing, example (6) being the correction:

(5) *Parlaimint **an Eoraip**
Parliament the.SG.DEF Europe.FEM.COM.SG

(6) Parlaimint **na hEorpa**
Parliament the.GEN.SG.DEF.FEM

Europe.FEM.GEN.SG.DEFART
'Parliament of Europe'

Here we want our suggestion to include changes to both the noun and the article as in Figure 5. (As the screenshot shows, the missing human-readable error explanation becomes glaringly obvious when testing in a plug-in UI.)

[5]The added word should also get the same error tag, in case there are several possible errors in the context.

Figure 5: Divvun Grammar checking for Irish (Microsoft by default shows the red underlines of English grammar checking).

The below rules will correct the noun:

```
ADD (&msyn-gen-case-nouns)
 TARGET (Noun Com)
 IF (*-1 (Noun Com) BARRIER (*) - Art) ;


COPY (Gen Sg DefArt SUGGEST)
 EXCEPT (Com Sg &msyn-gen-case-nouns)
 TARGET (Com Sg &msyn-gen-case-nouns)
 IF (-1 (Art Def)) (NEGATE 0 DefArt);
```

But we also want to correct its possible determiner. We could add the below rules, which will underline the determiner (in this context) and give it a suggestion:

```
ADD (&msyn-gen-case-nouns)
 TARGET ("an" Art Sg Def)
 IF (NEGATE 0 Gen)
    (1 (Noun &msyn-gen-case-nouns)) ;

COPY ("na" Gen Sg Def Fem SUGGEST)
 EXCEPT ("an" Sg Def &msyn-gen-case-nouns)
 TARGET (Art &msyn-gen-case-nouns)
 IF (1 (Noun Fem)) ;
```

However, with these rules alone, the system will see two separate, independent errors, leading to two disconnected underlines, each with their suggestion. This is not ideal. We want the user to be able to pick both corrections at once, since they belong together. To tie them together, we (somewhat arbitrarily) call the noun the "main" word of the error complex, and add a `LEFT` relation from the noun to the determiner:

```
ADDRELATION ($2 LEFT) (&msyn-gen-case-nouns)
 TO (-1 (Art &msyn-gen-case-nouns)) ;
```

We use the term *co-errors* for any words of the error complex that we do not identify as the *main* error cohort. The difference between main and co-errors *may* coincide with syntactic relations, but does not need to – the main error word is often chosen by the grammar writer based on what is

most stably wrong in this type of error.

Strictly speaking, we do not need an error tag at all for the article in this case. However, as we shall see below there may be several competing ways of correcting a complex error, and so we need to tell the system that *this* correction of the article belongs with *this* correction of the noun. We use relations to tie together the cohorts, but relations are cohort-to-cohort, not reading-to-reading. So we need a matching error tag[6] as well to pick out which *readings* of the two cohorts are connected.

7 Suggesting several things for one error in Lule Sámi numeral phrases

Numeral phrases are complex in Lule Sámi and they can be long. They typically consist of at minimum a numeral and a noun with possible modifiers (attributive adjectives or nouns in genitive case to mark the possessor). However, they can also contain coordinated numerals, demonstratives in front of the numeral and coordinated nouns.

(7) *Moadda tjijnnujn, girkkojn
some.ACC.SG cinema.INE.PL, church.INE.PL
ja almulasj bájkijn li dákkár
and common place.INE.PL are these
teleslivŋŋasa.
hearing.loop

(8) Moatten tjijnnun, girkkon
some.INE.SG cinema.INE.SG, church.INE.SG
ja almulasj bájken li dákkár
and common.ATTR place.INE.SG are these
teleslivŋŋasa.
hearing.loop
'Such audio induction loops are found in many cinemas, churches and public places'

We would like it to be corrected as displayed in Figure 6.

Ex. (8) consists of the following components:

There can be errors in only the demonstrative, only the numeral, only the noun, or in all three parts. That also means that sometimes we have several suggestions for corrections, depending on if we want to adapt the numeral to the noun, or

[6]In some rules you will see the error tag with a prefix **co&**, like `co&syn-abs-wordorder`; this is equivalent to `&syn-abs-wordorder`, except it makes it explicit that this cohort is not the "main" word of the error, so for example it should not be `$1` in explanations.

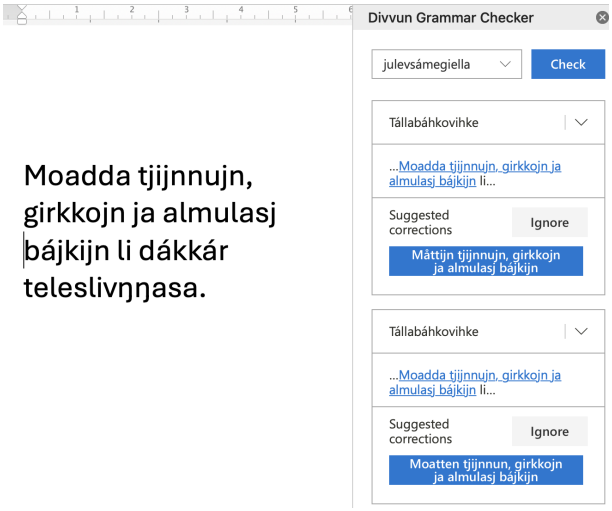

Figure 6: Divvun Grammar checking for Lule Sámi

the noun to the numeral. Technically that requires that we link all parts together in a certain way and also the corrections. In this case, we have an alternative correction which must be kept separate – all rules for the example (8) correction use the tag `&msyn-numphrase-sgine` while those for example (9) use the tag `&msyn-numphrase-sgcom` in order to avoid interference.

(9) Måttijn tjijnnujn,
some.COM.SG ccomma.COM.SG,
girkkojn ja almulasj
church.COM.SG and common.ATTR
bájkijn
place.COM.SG

The main error here is the first noun after the numeral, *tjijnnujn*. The numeral needs the co-errortag, and so do all the coordinated nouns. That means we need `LEFT` and `RIGHT` relations; `RIGHT` relations to the coordinated things to the right of the main error, and `LEFT` relations to the left of the main error, i.e. the numeral and/or the demonstrative.

1. ADD-rules for nouns
   (a) main error
   (b) coordinated nouns (co-error to the right)
   (c) coordinated nominal modifiers
2. COPY rules for nouns

3. ADD-rules for simple and coordinated numerals (co-errors)
4. ADDRELATION-rules from the main error to its co-errors
   (a) main noun to first coordinated noun (RIGHT)
   (b) main noun to second coordinated nouns (RIGHT)
   (c) noun to numeral (LEFT)
5. COPY rules for numerals
6. ADD-rule for demonstratives
7. ADDRELATION rules for demonstratives (LEFT)
8. COPY rule for demonstratives

In addition to the RIGHT and LEFT relation we add $2–$N to each of the co-errors (the main error being by default associated with $1), so we can refer to the word forms in the explanation shown in the MS Word plug-in etc.

8 Changing word order in Greenlandic

Word order rules are often relevant for languages that have a different standard word order than the languages they compete with, such as minority languages that are spoken in a majority language context. Both Greenlandic and Lule Sámi have SOV (subject object verb) as their standard word orders, whereas their competitors Danish, Swedish and Norwegian have a SVO standard word order, which may interfere and lead to word order errors. Danish and Greenlandic also differ in terms of noun phrase internal structure.

Moving around parts of the sentences requires that we know where the part has been before and at the same time know where we want to move it to.

The following set of rules for Greenlandic applies to possessive nominal phrases that are marked for their possessor, which is in relative case. The rules add an error-tag to a noun in absolutive case (the head of a possessum) preceded by an adjectival absolutive and a noun in relative case (the possessor). This is an error since in Greenlandic, possessor noun phrases place the adjective after and not before noun. We also add the DELETE tag since it is to be removed in the suggestion. We then copy the respective cohort (that belongs to the possessive adjectival noun phrase) and place it in the desired position (without removing the one in the erroneous position) and mark it with the tag &ADDED to show that this is not the original but the corrected position. Lastly, we establish a relation from the erroneous one so that we get the correct underline.

```
WITH (N Abs) + $$NUMERUS - ADJEKTIVISK IF
    (*-1 (3SgPoss Abs) + $$NUMERUS + ADJEKTIVISK
        BARRIER (*) - Abs
        LINK *-1 BOS LINK 1 (N Rel Sg))    # _C1_
    (NEGATE *0 &msyn-obj-marking-abs-3Pl0) # _C2_
    (NEGATE 0  &msyn-obj-marking-abs-3Pl0) # _C3_
{
  ADD (&syn-abs-wordorder DELETE) (*);
  COPYCOHORT
    (&ADDED co&syn-abs-wordorder)      # new tags
    EXCEPT (DELETE &syn-abs-wordorder) # remove these
    TARGET    (*)       # Copy from main WITH target
    TO BEFORE (jC1 (*)) # Put it before _C1_ context
    ;
  ADDRELATION ($3 LEFT) (*)
     TO (jC1 (*) LINK -1 (&ADDED));
};
```

This rule-set uses the new WITH (Swanson et al., 2023) feature in CG – the rules inside the braces will only apply when the outer context conditions match. Their target is the first context, and they may refer to the following outer contexts with C1, C2 and so on. This feature lets us avoid some redundancy in rule contexts.

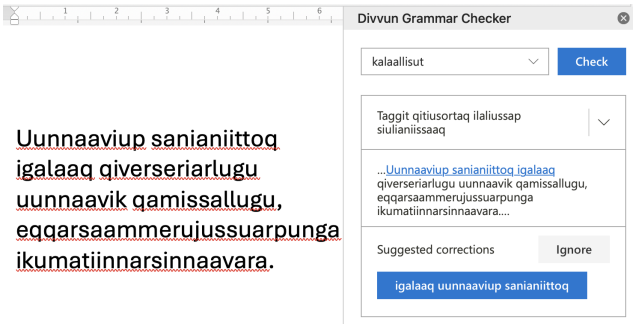

Figure 7: Divvun Grammar checking for Greenlandic (with MS Word default English spellchecking)

Example (10) has the wrong word order; the correct order is in example (11).[7]

(10) **Uunnaaviup sanianiittoq**
kettle.REL.SG next.to.3SGPOSS.ABS.SG
igalaaq qiverseriarlugu
window.ABS.SG upon.opening.1SG>3SG

[7]For further analysis check https://nutserut.gl/gloss.

uunnaavik
kettle.ABS.SG
qamissallugu,
when.intend.to.turn.off.1SG>3SG
eqqarsaammerujussuarpunga
think.suddenly.1SG
ikumatiinnarsinnaavara.
can.leave.on.1SG>3SG
'After/upon opening the window (which is) next to the kettle, when I went to turn off the kettle I suddenly thought I can leave it on.'

(11) **Igalaaq uunnaaviup sanianiittoq** qiverseriarlugu uunnaavik qamissallugu, eqqarsaammerujussuarpunga ikumatiinnarsinnaavara.

The word error correction is shown in Figure 7.

9 Performance

The Divvun grammar checker system is used for many languages, some with more extensive support than others. Mikkelsen and Wiechetek (2023) contains an evaluation of the state of the first version of the Lule Sámi grammar checker. That analysis does not include the new developments for multiword support. An extensive evaluation is out of scope of this paper, but we plan on evaluating, both the updated Lule Sámi system as well as other languages involved in the future.

Preliminary results for Lule Sámi numeral phrases are promising, cf. Table 1. We tested on the 39,891 word Lule Sámi corpus for evaluation purposes (SIKOR), which has been marked-up and then run through *GramDivvun*.

Metric	Count
False Positives	12
False Negatives	17
True Positives	42
F_1	74.3 %

Table 1: Evaluation results for numeral phrases

A third of the false negatives in our test are numeral phrases including the word *åvtå/avta*. We decided against performing grammar checking on this word due to its polysemy (in addition to the numeral *one* it also means *same*) which would lead to a lot of false positives.

Some of the false positives regard other grammatical constructions like *20 jahkásasj* meaning *20-year-old* and are mistaken to be part of a larger numeral phrase by the grammar checker.

10 Summing up

As this paper shows, the combination of finite state morphologies and Constraint Grammar lets us detect and correct a wide range of complex grammatical errors in morphologically rich, low-resource languages. The rule formalism and the surrounding tools enable proofing for languages that do not have official support in word processors like Microsoft Word and Google Docs – and in principle any text editing tool with some support for spelling and grammar plug-ins. Bridging the gap from the command-line to mainstream user interfaces both increases the usefulness of these tools, and boosts development progress – the user interface lets the linguist see the feedback from new angles and highlights shortcomings one might otherwise miss. When working on different languages, we discover new challenges in error correction. Looking forward, we plan to explore additional languages, cover more error types and streamline the rule writing process. The approach described in this paper demonstrates that rule-based GEC is an effective solution for correcting complex errors in languages that receive little-to-no official support from mainstream software vendors, enabling a user-friendly interface which gives speakers and learners of low-resource languages access to high quality proofing tools.

Acknowledgments

We thank Inga Lill Sigga Mikkelsen for her invaluable work on Lule Sámi, her help with the evaluation and suggestions for how the Constraint Grammar formalism should practically resolve grammar checking corrections. We thank Judithe Denbæk for her work on Greenlandic grammar checking. We thank Seanán Ó Coistín for his initiative to start up Irish grammar checking.

References

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: Xerox tools and techniques. *CSLI, Stanford*, pages 359–375.

Eckhard Bick. 2000. *The Parsing System 'Palavras': Automatic Grammatical Analysis of Portuguese in a Constraint Grammar Framework (higher doctoral dissertation)*. Aarhus University Press, Aarhus.

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.

Tino Didriksen. 2010. *Constraint Grammar Manual: 3rd version of the CG formalism variant*. GrammarSoft ApS, Denmark.

Vsevolod Dyomkin. 2015. Running lisp in production.

Fred Karlsson. 1990a. 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. 1990b. Constraint grammar as a framework for parsing unrestricted text. In *Proceedings of the 13th International Conference of Computational Linguistics*, volume 3, pages 168–173, Helsinki.

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

Krister Lindén, Erik Axelson, Senka Drobac, Sam Hardwick, Juha Kuokkala, Jyrki Niemi, Tommi A Pirinen, and Miikka Silfverberg. 2013. Hfst—a system for creating nlp tools. In *International workshop on systems and frameworks for computational morphology*, pages 53–71. Springer.

Inga Lill Sigga Mikkelsen and Linda Wiechetek. 2023. Supporting language users-releasing a fullfledged lule sámi grammar checker. In *Proceedings of the NoDaLiDa 2023 Workshop on Constraint Grammar-Methods, Tools and Applications*, pages 37–45.

SIKOR. UiT The Arctic University of Norway and the Norwegian Saami Parliament's Saami text collection, Version 06.11.2018. `http://gtweb.uit.no/korp`. Accessed: 2018-11-06.

Daniel Swanson, Tino Didriksen, and Francis Tyers. 2023. With context: Adding rule-grouping to visl cg-3. In *Proceedings of the NoDaLiDa 2023 Workshop on Constraint Grammar-Methods, Tools and Applications*, pages 10–14.

Linda Wiechetek, Katri Hiovain-Asikainen, Inga Lill Sigga Mikkelsen, Sjur Moshagen, Flammie Pirinen, Trond Trosterud, and Børre Gaup. 2022. Unmasking the myth of effortless big data - making an open source multi-lingual infrastructure and building language resources from scratch. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 1167–1177, Marseille, France. European Language Resources Association.

Linda Wiechetek, Sjur Nørstebø Moshagen, and Kevin Brubeck Unhammer. 2019. Seeing more than whitespace — tokenisation and disambiguation in a North Sámi grammar checker. In *Proceedings of the 3rd Workshop on the Use of Computational Methods in the Study of Endangered Languages Volume 1 (Papers)*, pages 46–55, Honolulu. Association for Computational Linguistics.