No more fumbling in the dark – Quality assurance of high-level NLP tools in a multi-lingual infrastructure

Linda Wiechetek linda.wiechetek@uit.no
Flammie A Pirinen tommi.pirinen@uit.no
Børre Gaup boerre.gaup@uit.no
Thomas Omma thomas.omma@uit.no

Divvun, UiT Norgga árktalaš universitehta

Abstract

We argue that regression testing is necessary to ensure reliability in the continuous development of NLP tools, especially higher level applications like grammar checkers. Our approach is rule-based, building on successful work for a number of low-resourced languages over the last 20 years. Instead of working with a black box, we choose a method that allows us to pinpoint the exact reasons for failures in the system. We present a tool for regression testing for *GramDivvun*, the rule-based open source North Sámi grammar checker. The regression tool is available for any of the 135 languages in the Giella-LT infrastructure and can be applied when respective tools are built. An evaluation of the system shows how the precision of the regression tests improves with almost 20% over a time span of 1.5 years. We also illustrate that the regression tool can detect undesired effects of rule changes that affect the performance of the grammar checker.

Abstrákta

Mii ákkastallat ahte regrešuvdnaiskosat leat dárbbašlaččat jus galgá sáhttit ráhkadit luohtehahtti NLP-reaidduid, erenoamážit reaidduid nugo grammatihkkadárkkisteddjiid, mat sorjástit máŋga eará prográmmaide. Min bargu lea njuolggadusvuođđuduvvon, huksejuvvon barggu ala mii lea dahkkon smávva-resursagielaiguin maŋemuš 20 jagi. Dan sajis go bargat "čáhppes bovssain", mii válljet vuogi man bokte mii dalán oaidnit gokko vuogádagas meattáhus čuožžila. Mii čájehit reaiddu mii iská leatgo *GramDivvumis* regrešuvnnat. *GramDivvun* lea njuolggadusvuođđuduvvon davvisámi rabas gáldokoda grammatihkkadárkkisteaddji. Regrešuvdnaiskanreaidu lea olámuttus visot 135 gillii mat leat GiellaLT-infrastruktuvrras ja dan sáhttá vuodjit go gullevaš reaiddut leat huksejuvvon. Vuogádatevalueren čájeha ahte regrešuvdnaiskosiid bohtosat leat buorránan measta 20 %:in beannot jagis. Mii maid čájehit ahte regrešuvdnaiskanreaidu gávdná meattáhusaid maŋŋá rievdadusaid mat váikkuhit grammatihkkadárkkisteaddji bohtosiidda.

Tiivistelmä

Tässä artikellissa esitämme että regressiotestaus on välttämätöntä kielitekonologiatyökalujen, eritoten korkeampitasoisten sovellusten kuten kieliopiontarkistinten, jatkuvassa kehityksessä. Meidän lähestymislähtökohtamme on sääntöpohjainen, ja rakentuu aiemmalle vähäresurssisten kielten työlle viimeisen 20 vuoden ajalta. Musta laatikko -lähestymistavan sijaan käytämme menetelmiä joiden avulla voimme suoraan paikantaa ongelmakohdat järjestelmässä. Esittelemme työkaluja joilla regressiotestataan *GramDivvunia*, sääntöpohjaista pohjoissaamen kieliopintarkistinta. Regressiotestaus on valmiina käytettävissä 135 kielelle, joita kehitetään GiellaLT-infrastruktuurissa ja sitä voi hyödyntää vastaavissa työkaluissa. Järjestelmää evaluoimalla huomaamme että tarkkuus kasvaa 20 % 1,5 vuoden seurantajakson aikana. Sen lisäksi tuomme esille kuinka regressiotesteillä voi havaita säännöstömuutosten vaikutuksia kieliopintarkistimen suorituskykyyn.

1 Introduction

This paper illustrates an efficient way to quality check high level rule-based NLP applications for low resource languages with complex morphology like North Sámi. In particular, we develop a powerful regression testing tool for the rule-based open source North Sámi grammar checker *GramDivvun* (Wiechetek et al., 2019a) that provides statistics of precision and recall specific to each error type[1] and a detailed analysis of each sentence including one or more (nested) errors[2], together with an advanced system of error mark-up that allows us to properly identify each error type module that is successful enough to be included in the grammar checker released to the public.

GramDivvun has been released by Divvun as a free plugin for Microsoft Office and Google Docs[3]. A grammar checker, as opposed to a spellchecker, is a tool that verifies and corrects errors in writing that are not mere mistyped non-words, but real words where the error is dependent on the whole sentence-context and its grammatical features.

North Sámi is a minority language in a bilingual language community, which faces challenges as regards writing proficiency. In this context, a reliable grammar checker can therefore also serve as a tool to improve writing skills. However, it is a difficult task to make a precise tool that meets users needs. If it underlines too many or even any correct sentences, the user will easily be frustrated and switch off the grammar checking. Regression testing resolves this problem in a robust and uniform way and ensures high quality of the tools.

North Sámi is a Uralic language spoken in Norway, Sweden and Finland by approximately 25,700 speakers (Simons and Fennig, 2018). It is a synthetic language, where the open parts of speech (PoS) – e.g. nouns, adjectives – inflect for case, person, number and more. The grammatical categories are expressed by a combination of suffixes and stem-internal processes affecting root vowels and consonants alike, making it perhaps the most fusional of all Uralic languages. In addition to compounding, inflection and derivation are common morphological processes in North Sámi. Due to its morphological complexity and, in addition, a large amount of homonymous forms or similar forms that can be confused in writing, there are many different grammatical error types. Similarly to other low-resource languages, there is little to no error marked-up data available for it, and the available data is seldom quality checked with regard to spelling and grammar. This poses a challenge to automatic grammar checking and testing.

Regression testing within software programming practice is defined as testing that ensures that recent code changes do not have any negative effects on existing features.[4] While regression testing is not a new idea and has been applied for some decades, to our knowledge, there are no in-detail publications of the challenges and practical solutions for it in grammar checking. However, Butt and Holloway King (2003) describe different testing strategies and their necessity for syntactic parsing. Since 2003, complexity of Natural Language Processing (NLP) tools has increased, which also requires adapting appropriate testing routines.

The rule-based model enables us to be very precise in locating the shortcomings of our grammar checker, and the regression tests ensure that the grammar checker keeps improving as new rules and tests to check them are added. The novelty in our approach to building grammar checkers lies in the workflows of simultaneously building the grammar checker rules, the error corpus and the regression testing suite. This workflow is an efficient approach to both building regression data and constructing our tools. The features of our tool are powerful enough to handle these multi-modular applications as well as an advanced mark-up system for a real world corpus that includes some spelling, morphological, syntactic, punctuation, space, real-word errors as well as nested errors per sentence. Also, the regression tool provides a detailed error analysis and not just overall regression statistics. It outputs error-specific statistics, including error subtypes, and enables efficient debugging of the system. The regression tools come with a database of tests, including several thousand sentences marked-up manually per error type.

2 Background

2.1 Framework

We are using a NLP development infrastructure called *GiellaLT* (Moshagen et al., 2014), which is

[1]More information on the different error types covered in *GramDivvun* can be found in (Wiechetek, 2017) and (Wiechetek et al., 2019b)

[2]Nested errors are errors within errors (typically with different scopes), for example a typo within an agreement error.

[3]`https://divvun.no/korrektur/gramcheck.html`

[4]`https://www.guru99.com/regression-testing.html` (Accessed 2021-03-23)

at present used by 135 languages. It consists of systems capable of building, testing and deploying a large range of NLP applications – including spelling and grammar checkers among others – based on finite-state morphology (Beesley and Karttunen, 2003) and Constraint Grammar (Karlsson, 1990). We apply a rule-based approach, which has a long tradition for the previously mentioned 135 languages, but is not as wide-spread as neural network approaches these days. Neural networks have shown to provide good results for many higher level NLP applications. However, they are also known to require large amounts of high quality or marked-up data, which for North Sámi would mean a manual quality check or mark-up as this data is not available. Our current error marked-up corpus (for all error types including nested errors) contains 120,459 words—a typical amount for training a neural network is at least several millions, and for a morphologically complexer language possibly more. Considering the amount of different types of errors there are and that not all of the sentences contain an error at all, this is very little data to train any kind of model.

Our work strategy consists in minimizing the workload by a combination of developing rule-based tools that reliably annotate and quality check our data and searching for and annotating example sentences from the corpus that give us further insight in the grammatical issue we are dealing with.

There is current work on neural network error detection/correction for specific 'simpler' grammatical errors (i.e. compound errors) in North Sámi that do not involve changing morphological forms or restructuring of a whole sentence (Wiechetek et al., 2021). However, rule-based tools were used, both to prepare the data and to access PoS information. Furthermore, its insertion of non-sense words restricts its usability for a community of real users. A full-fledged neural network grammar checker - that is not based on the rule-based grammar checker - is not to be realized in the near future.

Rule-based methods have the advantage of formalizing concise rules about the grammatical structure of a language. This gives us detailed insights in the language - as opposed to the black box of a neural network. This knowledge is necessary for defining errors in the first place, especially in cases where normative descriptions do not exist. It is also a prerequisite for debugging errors in our system. As we are able to translate language insights into formal grammar rules, we can pinpoint the exact causes of errors in our system. In other words, we can write a grammar that is both machine-, and to some extent, human-readable, which means that our knowledge can be used in other contexts outside of the grammar checker.

In the context of grammar checking tasks, specifically for morphologically complex and/or low-resourced languages, we would like to discuss two relevant tasks for neural network approaches, i.e. the systems for Latvian (Deksne, 2019) and Russian (Rozovskaya and Roth, 2019). The evaluation of Latvian neural network grammar checker shows a good performance with precisions between 78% and 98.5% (evaluated on a corpus of 115,000 sentences) depending on the error type. However, judging from their regular expressions to insert artificial errors, most of their error types seem to be fairly local errors that can be resolved based on shorter n-grams. The Russian system, on the other hand, focuses on more advanced error types, including case and agreement. However, precision (evaluated on a 206,258 token learners' corpus) is significantly lower — between 22% and 56%, only gender agreement reaches 68%. The corpus is rather small with regard to the task of correcting a large variety of errors. None of these two approaches deal with the advanced syntactic constructions we resolve in our approach, requiring an analysis of the whole sentence, valencies, semantic cues, etc.

The testing approach described here, while used in conjunction with a rule-based system, is agnostic of underlying technology, and could well be applied in the context of a neural system as well, should there be one that allows for correcting the errors the system makes.

2.2 Continuous integration and deployment

In order to provide a consistent grammar checking experience but also automatic updates and improvement, we apply stringent testing and combine that with a *continuous integration / deployment* (CI/CD) environment. To our knowledge, there are no publications on how to apply CI / CD to NLP product pipelines such as grammar checking, so in this article we lay out some guidelines and good practices. However, in the text books for the development of NLP applications we find some recommendations on the use of regression tests to compare different versions of the same application. (Grove, 2009, p.222) There have also been some workshops on regression testing in NLP, e.g. (Farrow and Dzikovska, 2009), however, these ideas have not found popular use, yet. One of the scientific contributions of our work is not only that we can provide the end users with products that work as expected, but also we can maintain scientific integrity of the systems in terms of *reproducibility*. We can apply the CI methods to ensure that systems can reproduce comparable results at all times. This is especially attractive for our case, since we apply mainly rule-based methods for grammar checking and correction, the results should stay relatively stable for the same versions of the system. In the recent years, the reproducibility has been brought to focus of the NLP research, with famous works like Pedersen (2008).

Typically, continuous development of rule-based NLP applications involves unexpected breakage. With regression tests for each error type in the grammar checker, regressions are caught quickly. This means that refactoring or larger changes to the code can be done without decreasing the overall quality of the grammar checker.

The main motivation behind introducing regression testing came from the need of automatizing the grammar checker evaluation. Manual evaluation to calculate precision and recall got rather cumbersome. This led to the development of a more powerful tool for testing grammar checking automatically (Wiechetek et al., 2019b), and there was parallel work and methodological in-depth study on corpus mark-up. Based on this work, we did not have to make a big leap to get regression testing. We reused the evaluation tool and turned it into a proper tester, with detailed statistics of the performance of the tool and sentence-by-sentence analysis that provides a basis for debugging.

2.3 The North Sámi grammar checker

The grammar checker for North Sámi (*GramDivvun*) performs both spell- and grammar checking – i.e. requiring full sentence analysis to identify local and global syntactic errors – in addition to punctuation and format checking. It includes a version of the open-source spelling checker that has been freely distributed since 2007[5], cf. also Gaup et al. (2006). It uses the HFST-based spelling mechanism described in Pirinen and Lindén (2014) for a number of modules, and in addition includes six Constraint Grammar modules, cf. Figure 1. These are:

- Two valency grammars applied before and after spellchecking (*valency.cg3* and *valency-postspell.cg3*)
- A tokenizer (*mwe-dis.cg3*)
- Two morpho-syntactic disambiguators applied before and after spellchecking (*grc-disambiguator.cg3* and *after-speller-disambiguator.cg3*)
- A module for more advanced grammar checking (*grammarchecker-release.cg3*)

The current version of the grammar checker module in *GramDivvun*[6] includes 313 error detection rules, 4 purely morpho-syntactic rule types, 17 morpho-syntactic rule types that are caused by general real-word rule types, 17 idiosyncratic real word error rule types, 14 punctuation or space error rule types and one spelling error rule type. A real word error is typically a misspelling, but unlike regular typos it results in (similar) real word rather than a non-word. Therefore, an analysis of the sentence is necessary to identify the error. In English language, *dessert* can be a real word error of *desert* and vice versa.

As in English, there are numerous idiosyncratic real word error types in North Sámi, made by native speakers for various reasons (i.e. dialectal phonetic differences that do not coincide with the written norm, vowel and consonant errors based on confusion of different forms, etc.) But some of these errors are more systematic, such as the confusion of case-marked (locative case) vs. attributive adjective forms. This is the case in ex. (1)[7], where the locative form *álkis* should be an attributive one, i.e. *álkes*, and the only distinction between these forms is the vowel - *e* vs. *i*.

(1)

Snoranuohtti	lea	gehppes	ja	**álkis**	veahkkeneavvu.
Danish seine	be.3SG	light	and	simple.LOC	tool

'Danish seine is a light and simple tool.'

Instead of resulting in a simple non-word, in North Sámi vowel confusion can have grammatical

[5]http://divvun.no/korrektur/korrektur.html

[6]https://github.com/giellalt/lang-sme/releases/tag/naacl-2021-4

[7]All examples are original examples or fragments from *SIKOR* and are most likely native speaker texts or translations.

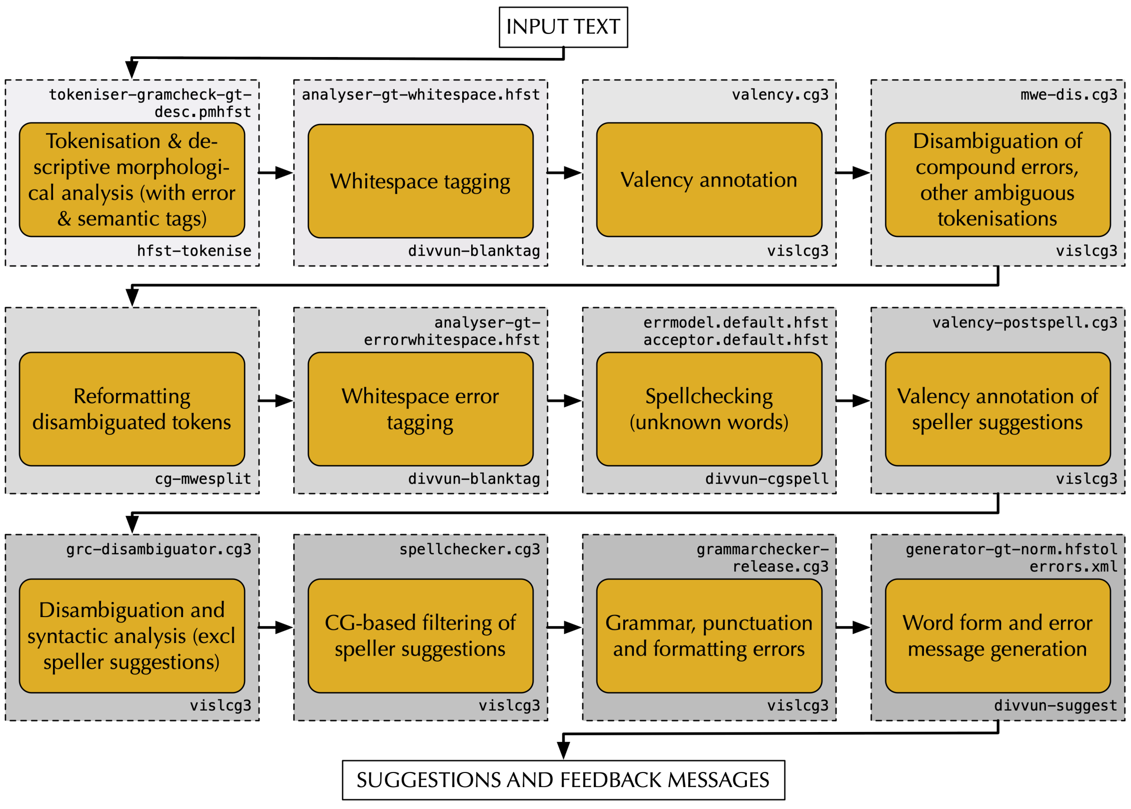

INPUT TEXT
tokeniser-gramcheck-gt-desc.pmhfst
Tokenisation & descriptive morphological analysis (with error & semantic tags)
hfst-tokenise
analyser-gt-whitespace.hfst
Whitespace tagging
divvun-blanktag
valency.cg3
Valency annotation
vislcg3
mwe-dis.cg3
Disambiguation of compound errors, other ambiguous tokenisations
vislcg3
Reformatting disambiguated tokens
cg-mwesplit
analyser-gt-errorwhitespace.hfst
Whitespace error tagging
divvun-blanktag
errmodel.default.hfst
acceptor.default.hfst
Spellchecking (unknown words)
divvun-cgspell
valency-postspell.cg3
Valency annotation of speller suggestions
vislcg3
grc-disambiguator.cg3
Disambiguation and syntactic analysis (excl speller suggestions)
vislcg3
spellchecker.cg3
CG-based filtering of speller suggestions
vislcg3
grammarchecker-release.cg3
Grammar, punctuation and formatting errors
vislcg3
generator-gt-norm.hfstol
errors.xml
Word form and error message generation
divvun-suggest
SUGGESTIONS AND FEEDBACK MESSAGES

Figure 1: System architecture of *GramDivvun*

consequences. That means that a certain grammatical form can be confused with another grammatical form of the same lemma. Since both forms regard the same lemma, these errors can be detected and corrected systematically. Apart from that, other (morpho-phonetic) criteria decide which forms are eligible for this error type. These are lemma endings (e.g. *-it*, *-at*, or *-ut*), number of syllables (even vs. uneven), and consonant gradation class membership.[8] Table 1 illustrates one of the consonant gradation classes with examples.

Nominal derivations of certain types of verbs (i.e. with a particular ending and a specific consonant gradation pattern In ex. (2), the vowel confusion (*u*/*o*) regards derived nouns (that should be past participle forms) from consonant gradation class 4D (cf. Table 1). Here, the (derived) noun *vákšun* '(the act of) observing' is confused with the past participle *vákšon* 'observed'.

(2)	Politiijat	leat	otne	**vákšun**	johtolaga
	police	be.3PL	today	observing.NOM	traffic

Consonant center		Example		Translation
kc	vcc	ci**kc**ut	ci**vcc**ui	'(to) pinch – s/he pinched'
kč	včč	go**kč**at	go**včč**at	'(to) cover – you cover'
ks	vss	oa**ks**i	oa**vss**it	'branch – branches'
kst	vstt	tea**kst**a	tea**vstt**at	'text – texts'
kš	všš	di**kš**ut	di**všš**un	'(to) take care – I take care'
kt	vtt	[…]		

Table 1: Consonant gradation group 4D according to Nickel (1994, p. 30)

Guovdageainnus.
Guovdageaidnu.LOC
'The police has conducted a traffic control in Guovdageaidnu today.'

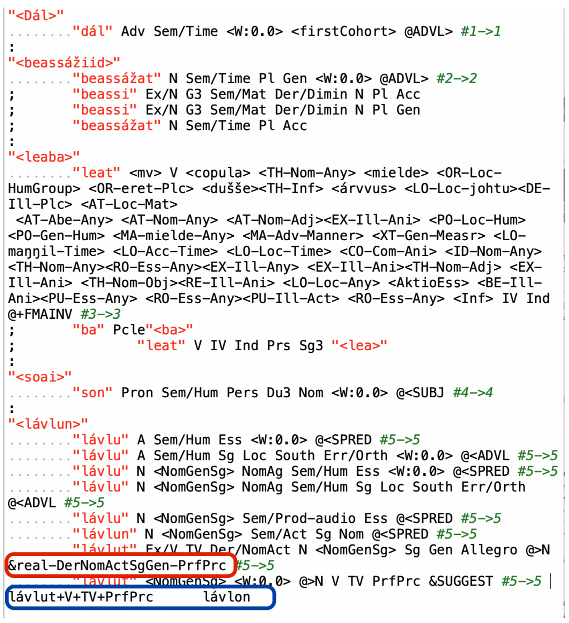
The complex structure of the grammar checker shows that there are modifications in many different modules that can be responsible for possible mishaps, since changes in one module can affect the input to subsequent modules.

The input for the grammar checker are unmarked sentences. The input for the regression tests are sentences with an error mark-up like in ex. (3).

[8]A number of Finno-Ugric languages use stem-internal morpho-phonological changes in addition to suffixes to mark case and other morphological processes. In North Sámi there are 123 consonant gradation patterns (Nickel, 1994, p.23-30)

(3) Dál beassážiid leaba soai
now Easter have they.DU
{lávlun}¢{lávlon} dáid sálmmaid
singer sing.PASTP these psalm.ACC.PL
girkuin Guovdageainnus,
church.LOC.PL Guovdageaidnu.LOC,
Kárášjogas ja Mázes.
Kárášjohka.LOC and Máze.LOC
'This Easter they have sung these psalms in the churches of Guovdageaidnu, Kárášjohka and Máze.'

Figure 2 shows the output for the grammar checker including error detection (red rectangle) and error correction (blue rectangle). The sentence is tokenized and reads from the top to the bottom. Word forms are in angle brackets, indented lines are homonymous analyses of each form, including lemmata, morphological, semantic and syntactic tags followed by numerical dependencies.



```
"<Dál>"
........"dál" Adv Sem/Time <W:0.0> <firstCohort> @ADVL> #1->1
:
"<beassážiid>"
........"beassážat" N Sem/Time Pl Gen <W:0.0> @ADVL> #2->2
;       "beassi" Ex/N G3 Sem/Mat Der/Dimin N Pl Acc
;       "beassi" Ex/N G3 Sem/Mat Der/Dimin N Pl Gen
;       "beassážat" N Sem/Time Pl Acc
:
"<leaba>"
........"leat" <mv> V <copula> <TH-Nom-Any> <mielde> <OR-Loc-
HumGroup> <OR-eret-Plc> <dušše><TH-Inf> <árvvus> <LO-Loc-johtu><DE-
Ill-Plc> <AT-Loc-Mat>
 <AT-Abe-Any> <AT-Nom-Any> <AT-Nom-Adj><EX-Ill-Ani> <PO-Loc-Hum>
<PO-Gen-Hum> <MA-mielde-Any> <MA-Adv-Manner> <XT-Gen-Measr> <LO-
maŋŋil-Time> <LO-Acc-Time> <LO-Loc-Time> <CO-Com-Ani> <ID-Nom-Any>
<TH-Nom-Any><RO-Ess-Any><EX-Ill-Any> <EX-Ill-Ani><TH-Nom-Adj> <EX-
Ill-Ani> <TH-Nom-Obj><RE-Ill-Ani> <LO-Loc-Any> <AktioEss> <BE-Ill-
Ani><PU-Ess-Any> <RO-Ess-Any><PU-Ill-Act> <RO-Ess-Any> <Inf> IV Ind
@+FMAINV #3->3
;       "ba" Pcle"<ba>"
;               "leat" V IV Ind Prs Sg3 "<lea>"
:
"<soai>"
........"son" Pron Sem/Hum Pers Du3 Nom <W:0.0> @<SUBJ #4->4
:
"<lávlun>"
........"lávlu" A Sem/Hum Ess <W:0.0> @<SPRED #5->5
........"lávlu" A Sem/Hum Sg Loc South Err/Orth <W:0.0> @<ADVL #5->5
........"lávlu" N <NomGenSg> NomAg Sem/Hum Ess <W:0.0> @<SPRED #5->5
........"lávlu" N <NomGenSg> NomAg Sem/Hum Sg Loc South Err/Orth
@<ADVL #5->5
........"lávlu" N <NomGenSg> Sem/Prod-audio Ess @<SPRED #5->5
........"lávlun" N <NomGenSg> Sem/Act Sg Nom @<SPRED #5->5
        "lávlut" Ex/V TV Der/NomAct N <NomGenSg> Sg Gen Allegro @>N
&real-DerNomActSgGen-PrfPrc #5->5
        "lávlut" <NomGenSg> <W:0.0> @>N V TV PrfPrc &SUGGEST #5->5
lávlut+V+TV+PrfPrc      lávlon
```

Figure 2: Output of *GramDivvun* in the command line

3 Regression testing for grammar checking

Regression testing for grammar checking is based on an error marked-up corpus. We have collected an error corpus of representative errors in *Yaml*-formatted[9] files specific to each error type. At the current date in august 2021, these include 17,800 sentences. Typically, each regression file contains several hundred sentences, some up to 4,300 sentences. There should be a balance of correct and erroneous sentences covering the same phenomena so that one can test for false positives and false negatives. Test sentences should cover a variety of syntactic contexts and pay attention to long-distance relationships between syntactic functions. They should include coordination, (inserted) subclauses, complex noun phrases, multiple adverbials, idiomatic constructions, multiple errors, punctuation, and other phenomena that can alter the status of the error/correct form. The collected errors are designed to cover a maximally large amount of real-world errors that people make when writing texts, in order to keep the grammar checker usable for people. The file naming is now error-specific,[10] but as they come from an authentic corpus, they can contain multiple errors per sentence including other types of errors and nested errors.

Yaml is a mark-up language with a simple syntax that makes writings of the tests convenient and co-operation with programmers and linguists easier. We chose to use the Yaml format for grammar testing because of positive experiences with the use of the same format for spell checker testing.[11] The original test framework for morphology testing initiated by Brendan Molloy can be found on GitHub.[12]

The regression test script measures both error detection and error correction and whether they match the manual error mark-up. False negatives of the type fn_1 are correctly detected errors that do not receive any corrections by the grammar checker. False negatives of the type fn_2 are undetected errors. The same goes for false positives, where: fp_1 are correctly detected errors with a wrong correction, and fp_2 are error detections that are not manually marked up. True positives (tp), on the other hand, are detected and corrected errors that match with the manual mark-up. In our final evaluation, we will not distinguish between these and only take into account successful vs. unsuccessful error correction in terms of false negatives and true/false positives. The tester script is implemented in Python and can be downloaded from GitHub[13].

[9]https://yaml.org/spec/1.2/spec.html

[10]current examples: https://github.com/giellalt/lang-sme/tree/main/tools/grammarcheckers/tests

[11]https://giellalt.uit.no/infra/infraremake/AddingMorphologicalTestData.html#Yaml+tests

[12]https://github.com/apertium/apertium-tgl-ceb/blob/master/dev/verbs/HfstTester.py

[13]https://github.com/giellalt/giella-core/blob/master/scripts/gramcheck-test.py

The grammar checker makes a list of each error that consists of the erroneous word, the position of the error (start and end), a list of suggestions and error type. The error mark-up is then converted to the same structure so that manual and grammar checker mark-up can be compared. For each of these test sentences, three things are collected: the erroneous version of the error marked-up sentence, the error marked-up version of the errors in the sentence and the errors detected by the sending the erroneous sentence through the grammar checker. The tester prints the outcome of each of the tests in a detailed manner, sentence by sentence and with references to the particular error types involved. The final report contains the number of total passes, fails, true and false positives/negatives, precision, recall and F_1-score. On exit, the script returns 0 or 1, 0 meaning all tests succeeded, 1 otherwise.

The test script is fast and light-weight enough to be part of a CI/CD system, even with processor time and RAM limitation, e.g. testing 300 sentences on the developers' machines takes about 30 seconds.

The error mark-up formalism has earlier been used to automatize spellchecking for Greenlandic, Icelandic, North, Lule and South Sami.

The error mark-up follows a number of guidelines[14] based on earlier corpus mark-up (Moshagen, 2014) and applies eight different general error types, each of them marked by a different sign: orthographic, real word, morpho-syntactic, syntactic, lexical, formatting, foreign language, and unclassified errors. The error is enclosed in curly brackets, followed by its correction in another set of curly brackets. The second curly bracket may or may not include a part of speech, morpho-syntactic criteria and a subclassification of the error type.

Orthographic errors (marked by $) include non-words only. They are traditional misspellings confined to single (error) strings, and the traditional speller should detect them. *Real word errors* (marked by ¢) are misspellings that cannot be detected by a traditional speller, they are an analysis of the surrounding words. *Morpho-syntactic errors* (marked by £) are case, agreement, tense, mode errors. They require an analysis of (parts of) the sentence or surrounding words to be detected. *Syntactic errors* (marked by ¥) require a partial or full analysis of (parts of) the sentence or surrounding words. They include word order errors, compound errors, missing words, and redundant words. *Lexical errors* (marked by €) include wrong derivations. *Foreign language* (marked by ∞) includes words in other languages that do not require a correction. *Formatting errors* (marked by ‰) include spacing errors in combination with punctuation. Unclassified errors are marked with §.

In ex. (4), the tokens involved in the error are nouns, the syntactic error is a missing word and the correction is adding the subjunction *ahte* 'that'.

(4) Illá {jáhkken}¥{missing|jáhkken ahte}
hardly think.PAST.1SG
lei duohta.
be.PAST.3SG true
'I hardly thought that it was true.'

Regarding the span of an error, we typically mark as little as possible, even if larger parts of the sentence are responsible for the identification of the error. This is done to facilitate matching error markup with grammar checker marking of the error, and it has direct effect on automatic evaluation. Most of the frameworks we use to process language material in context, e.g. Constraint Grammar takes a token-based approach to language processing, and therefore marking several words can get cumbersome and should be avoided if possible.

Ex. (5) shows the mark-up of nested errors. There is both a morpho-syntactic error, the case of *linjá* 'line' should be accusative instead of nominative, and a compound error, *njuolggo* and *linjjá* should be written as one word.

(5) Sárggo **{njuolggo**
draw.IMPRT.2SG straight
{linjá}£{noun,obj,accsg,nomsg,casellinjjá}}
line
¥{noun,cmp|njuolggolinjjá} dán guovtti
(straightline) these two
čuoggá gaskka.
points between.
'Draw a straight line between these two points.'

4 Evaluation

We performed two measurements of the system quality: firstly we have the well-curated and targeted regression test suite that is summarized in Table 2. Secondly, we measure an overview of how the system fares for texts in the whole corpora in the wild in Table 3. The first test suite verifies our system's quality in the regression test sense, and the second test ensures that the system works for open text case.

[14] https://giellalt.uit.no/proof/spelling/testdoc/error-markup.html

	naacl-1	**naacl-2 baseline**	**naacl-4**
Precision	70.9%	68.9%	88.8%
Recall	66.9%	84.0%	91.0%
F_1**-score**	68.8	75.7	89.9

Table 2: Evaluation results from the regression tests.

4.1 Quantitative evaluation

In Table 2 we show the results of the regression tests at the same three stages of the development. We measure the success percentage in terms of the number of the tests passed from the overall tests. The regression test corpus we use is a set of tests selected to have a representative coverage of the various error types and contexts. With the carefully selected grammar tests we can control the quality of the overall system, the overall aim for these grammar tests is to keep the correctness at 100 %. The correctness measure C here is $C = \frac{\text{tp}}{\text{CS}}$ where CS is the corpus size.

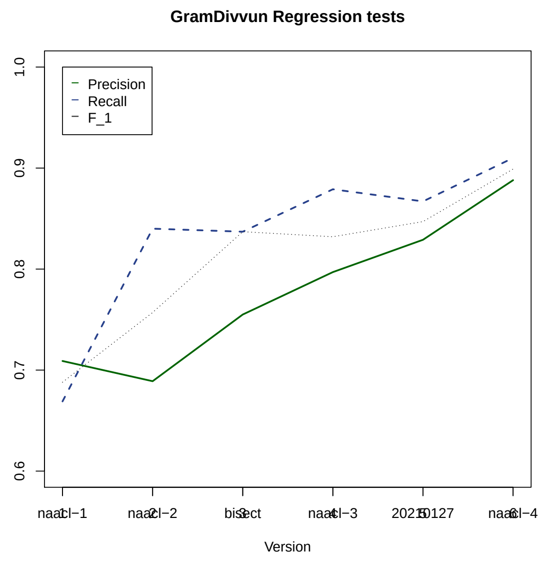


Figure 3: Development of *GramDivvun* precision and recall in the regression tests

In Table 3, we show the overall performance of *GramDivvun* at three stages over the course of approximately one and a half years of continuous development. This means that all grammatical errors are included, also the ones that the grammar checker does not have any module for yet. The tests are done on an error marked-up evaluation-corpus of approx. 26,000 words. The first test is made with the North Sámi grammar checker from 2019-11-21[15] before the introduction of the Yaml-tests (naacl-1). The second test uses the version from 2020-11-20[16] (naacl-2 - Yaml baseline) from when we had first introduced the regression tests. The third test uses the North Sámi grammar checker from 2021-03-20[17] (naacl-4) where we have taken into account results from the regression tests in the form of general rule changes.

The results show that the overall performance of the grammar checker on a small error marked-up corpus improves only slightly. This is due to the frequency of the errors we worked on. The corpus to test these error types in particular needs to be substantially bigger to show a change in performance. However, especially recall has improved by 6% showing an increased coverage of the error types covered in the grammar checker.

Figure 3 shows a number of stages of the performance of the grammar checker after developing regression tests. There was a significant drop in precision (naacl-2) and a number of drops in recall (bisect).[18] These coincided with the addition of test sentences (the regression tests grew from a couple of sentences to larger corpora of several thousand sentences), introducing new contexts that required stricter rules. Stricter rules typically lower recall to ensure stable precision. New, more specific rules need to be introduced to get recall up again. This explains the ups and downs in the graph. After the introduction of Yaml tests, however, we can see that precision has steadily been going up, and by that proves the main objective of regression tests right.

4.2 Qualitative evaluation

One can generally see, that rule types that have been prioritized in the grammar checker improved after involving regression testing.

Precision got better in ex. (6), where the nominalization *dovdan* 'feeling' is confused with the first-person singular form *dovddan* 'I know', forms that are distinguished by a change in the consonant centre only.

(6)	Buohkat,	geaid	**dovdan**,	oaivvildit
	All,	who.ACC.PL	feeling,	think.3PL

[15] https://github.com/giellalt/lang-sme/releases/tag/naacl-2021-1

[16] https://github.com/giellalt/lang-sme/releases/tag/naacl-2021-2

[17] https://github.com/giellalt/lang-sme/releases/tag/naacl-2021-4

[18] https://github.com/giellalt/lang-sme/commit/216d00d37bff6ebbd34a1529eb822b61b50a3f08

	naacl-1	**naacl-2 baseline**	**naacl-4**
Precision	80.0%	75.3%	82.3%
Recall	59.8%	65.9%	65.1%
F_1**-score**	68.4	70.2	72.7
TP	391	439	430
FP	98	144	92
FN	263	227	231

Table 3: Performance of *GramDivvun* over the span of a year, before and after introducing regression tests

seamma:
same

'Everybody I know thinks the same'

In ex. (7), *GramDivvun* finds the locative adjective form *oktageardánis*, which by analogy is confused with the nominative form *oktageardán*.

(7)

Skuvllas	berreše	maiddái	leat
school.LOC	should.COND.3PL	also	be
oktageardánis			
simple			

'The school should also have simple'

A number of error type rules are causing false positives in certain contexts such as ex. (8), where the infinitive *oastit* 'buy' is a correct form. However, it is homonymous with a second-person plural imperative reading of the same verb, and is falsely corrected to the third-person plural reading *ostet*.

(8)

Golut	**oastit**	dárbbašlaš	girjjiid
expenditure.PL	buy	necessary	book.ACC.PL
čađahit	prošeavtta.		
carry.through	project.ACC		

'Expenditure to buy necessary books to carry through the project.'

Some errors that are dealt with in the grammar checker are not recognized in certain syntactic contexts, such as the compound error *guovddáš doaimmat* that should be written as one word in ex. (9).

(9)

Movttiidahttin	ja	bagadeapmi	leat
motivation	and	instruction	be.3PL
SOR:a	prošeakta-jođiheaddji	deatalaš	ja
SOR.GEN	project-leader	important	and
guovddáš	**doaimmat**.		
central	task.PL		

'Motivation and instruction are important and central tasks for SOR's project leader'

In addition, there are error types that the grammar checker does not deal with at all, which is why they are not recognized, and the result are false negatives. This is the case of the syntactic error ex. (10), where the subjunction *vai* 'so that' before the finite verb *beassaba* 'get to' is missing.

(10)

Máŋgii	vahkkui	viežžá	son	vierrobeatnaga
often	week.ILL	fetch.3SG	s/he	foreign.dog
beassaba	vázzit.			
get.to3DU	walk			

'Many times a week she fetches the foreign dog so that they get to walk.'

5 Discussion and future outlook

In this paper we have shown that regression testing is necessary to provide reliable results (i.e. in particular a stable precision) for the users of higher level NLP applications like grammar checkers. A rule-based approach is successful for applications like grammar checking which require a high level of systematicity and reliable results. For low-resourced languages, where availability of resources such as expert-curated error-correction corpora are scarce, the development of rule-based tools is the most efficient approach. We showed that by using comprehensive regression testings we can keep developing the grammar checking and correction on a day-to-day basis and provide the end users with the newest updates without worrying about their quality. In the future we would like to see if it is possible to gather enough resources for a neural network based grammar checking and correction. Regression testing of the kind we described is applicable for neural network approaches as well. However, neural network systems do not allow for specific adjustments within the error types, which is rather a weakness of the system itself. It is therefore natural to apply these regression tests for neural network models as well, and we expect that the system will work in conjunction to neural network without any major changes.

We have started with neural network-approaches (forthcoming) for the correction of certain error types from our rule-based grammar checker. These require a preparation of the data by means of our existing rule-based tools, both for part-of-speech tagging and marking up error data.

One of the interesting features of a rule-based system, that has been brought to focus on the NLP community recently, is the energy-footprint of the used models. In case of our models, the rules can be compiled into finite-state automata on an average consumer desktop within minutes, and the actual models can be run on low-end mobile devices, so the energy footprint is trivially multiple orders of magnitude lower than that of any neural language models.

Acknowledgements

We thank Ritva Nystad for marking up part of the evaluation corpus, testing *GramDivvun* and contributing to critical discussions about grammatical errors. We also thank Sjur Nørstebø Moshagen and Brendan Molloy for the initial test setup.

References

Kenneth R Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI publications.

Miriam Butt and Tracy Holloway King. 2003. *Grammar Writing, Testing, and Evaluation*, pages 129–179. CSLI Publications, Stanford.

Daiga Deksne. 2019. Bidirectional lstm tagger for latvian grammatical error detection. In *Ekštein K. (eds) Text, Speech, and Dialogue. TSD 2019. Lecture Notes in Computer Science, vol 11697. Springer*.

Elaine Farrow and Myroslava O. Dzikovska. 2009. Context-dependent regression testing for natural language processing. In *Proceedings of the Workshop on Software Engineering, Testing, and Quality Assurance for Natural Language Processing (SETQA-NLP 2009)*, pages 5–13, Boulder, Colorado. Association for Computational Linguistics.

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.

Ralph F Grove. 2009. *Web Based Application Development*. Jones & Bartlett Publishers.

Fred Karlsson. 1990. 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.

Sjur Moshagen. 2014. Test data and testing of spelling checkers. Presentation at the NorWEST2014 workshop.

Sjur Moshagen, Jack Rueter, Tommi Pirinen, Trond Trosterud, and Francis M Tyers. 2014. Open-source infrastructures for collaborative work on under-resourced languages. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC*, pages 71–77.

Klaus Peter Nickel. 1994. *Samisk grammatikk*, second edition. Davvi Girji, Kárášjohka.

Ted Pedersen. 2008. Empiricism is not a matter of faith. *Computational Linguistics*, 34:465–470.

Tommi A. Pirinen and Krister Lindén. 2014. State-of-the-art in weighted finite-state spell-checking. In *Proceedings of the 15th International Conference on Computational Linguistics and Intelligent Text Processing - Volume 8404*, CICLing 2014, pages 519–532, Berlin, Heidelberg. Springer-Verlag.

Alla Rozovskaya and Dan Roth. 2019. Grammar error correction in morphologically rich languages: The case of russian. In *Transactions of the Association for Computational Linguistics, vol. 7, pp. 1–17, 2019*.

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

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.

Linda Wiechetek, Sjur Nørstebø Moshagen, Børre Gaup, and Thomas Omma. 2019a. Many shades of grammar checking – launching a constraint grammar tool for north sámi. In *Proceedings of the NoDaLiDa 2019 Workshop on Constraint Grammar - Methods, Tools and Applications*, NEALT Proceedings Series 33:8, pages 35–44.

Linda Wiechetek, Sjur Nørstebø Moshagen, Børre Gaup, and Thomas Omma. 2019b. Many shades of grammar checking - Launching a Constraint Grammar tool for North Sámi. In *Proceedings of the NoDaLiDa 2019 Workshop on Constraint Grammar - Methods, Tools and Applications (NoDaLiDa 2019)*, pages 35–44.

Linda Wiechetek, Tommi A Pirinen, Mika Hämäläinen, and Chiara Argese. 2021. Rules ruling neural networks - how can rule-based and neural models benefit from each other when building a grammar checker? In *forthcoming*.