Developing a morpholgical disambiguator for Lithuanian based on Constraint Grammar

Francis Jagiella
Department of Linguistics
Indiana University
Bloomington, IN, USA
fjagiell@indiana.edu

Abstract

This paper presents preliminary work on a morphological disambiguator for Lithuanian based on Constraint Grammar (Karlsson, 1990). Lithuanian is a Baltic language with rich morphology. The pipeline consists of a morphological analyser of all possible interpretations for the wordforms in the corpus as well as a constraint grammar. In the test corpus, the constraint grammar has a precision of .9025, a recall of .9752, and an F1 score of .9374.

1 Introduction

This paper presents a preliminary constraint grammar for Lithuanian*. A constraint grammar is a rule-based disambiguator which can serve as input for other natural language processing tasks. The main objective in developing this constraint grammar was precision. The corpus used to develop this constraint grammar with the Lithuanian ALKSNIS treebank from the Universal Dependencies Project (Bielinskiene et al., 2016). The train corpus of the treebank was used to write the rules, the dev corpus was used to further develop the rules, and the test corpus was used for computing the reported precision, recall, and F1 score.

The paper is organized as follows: section 2 contains a brief review of literature, section 3 reviews ambiguities in Lithuanian, section 4 describes the analysis pipeline, section 5 describes the development process, section 6 evaluates the results, and section 7 presents the conclusion and future work related to the project.

2 Review of literature

There has not yet been a constraint grammar developed for the Lithuanian language. There has, however, been a fair amount of linguistic research on the language. For many years this research was largely in historical linguistics, where Lithuanian received much more attention than fellow Baltic language Latvian. Since the fall of the Soviet Union, there has been greater study into other areas of the Lithuanian language. Little of this work, however, has been translated into other languages (Usoniene et al., 2012). Nevertheless, there are English-language books and translations on Lithuanian grammar as well as Lithuanian dictionaries readily available, which were consulted in the development of this constraint grammar (Mathiassen, 1997; Ramoniene and Pribusauskaite, 2008; Piesarkas, 2006; Piesarkas and Svecevicius, 1995). Additionally, there have been instances of computational work done on the language, including a morphological analyser (Kapočiūtė-Dzikienė et al., 2017).

In terms of previous work done with constraint grammar, there are many examples of it being used in various languages and for different purposes. In addition to functioning on its own as a disambiguator, constraint grammar can also be integrated into parsers based on other formalisms, such as LFG (Dione, 2014). Constraint grammars have also been integrated with finite-state transducers (Trosterud, 2009). Although a high precision was the goal in developing the Lithuanian constraint grammar, others have focused on a high recall (Reynolds and Tyers, 2015).

3 Ambiguity in Lithuanian

In the corpus there are various examples of intrapardigmatic, morphosyntactically incongruent, and morphosyntactically congruent ambiguity.

In intraparadigmatic ambiguity, homographic wordforms are part of the same lexeme but contain different features. One such wordform is 'yra' of the lemma būti-'to be':

- *yra* ‘be-AUX, SING, 3’
- *yra* ‘be-AUX, PLUR, 3’
- *yra* ‘be-VERB, SING, 3’
- *yra* ‘be-VERB, PLUR, 3’

Another intrapardigmatic ambiguity in Lithuanian is that of ‘kitas’-‘other’ (Piesarkas and Svecevicius, 1995):

- *kitas* ‘other-PRON, ACC, FEM, PLUR’
- *kitas* ‘other-PRON, NOM, MASC, SING’

Morphosyntactically incongruent ambiguity is when two homographs are part of separate lexemes and their morphosyntactic values are different. The wordform ‘tai’ serves as such an example in Lithuanian (Piesarkas, 2006):

- *tai* ‘that/those-DET NEUT DEM’
- *tai* ‘so-PART’

In contrast, morphosyntactically congruent ambiguity is when two homographs are part of separate lexemes but their morphosyntactic values are the same. The wordform ‘pirmyn’ demonstrates this phenomenon:

- *pirmyn* pirma ‘first-ADV, POS’
- *pirmyn* pirmyn ‘forward-ADV, POS’

4 Analysis pipeline

4.1 Morphological analyser

To create a list of wordforms and their possible interpretations, I used Python program which creates a morphological analyser by treating the train corpus of the treebank as a fullform list*.

Examples from the morphological analyser are as follows:

- nėrinys Case=Nom Gender=Masc Number=Plur ‘nėriniai’ ‘lace’
- komitetas Case=Gen Gender=Masc Number=Plur ‘komitetu’ ‘committee’

*https://github.com/ftyers/ud-scripts/blob/master/conllu-analyser.py

4.2 Rule writing

The constraint grammar, lt.cg3, is composed of 79 rules, 26 of which are remove and 53 of which are select. Remove will eliminate all readings containing a specific feature, such as in the following example:

- REMOVE:r21 CCONJ IF (1C PUNCT) ;

This rule will discard all coordinating conjunction (CCONJ) readings if the following token is punctuation (PUNCT). Select, on the otherhand, will retain all readings containing a specific feature, discaring whatever else remains.

- SELECT:s19 SCONJ IF (-1 PUNCT) (1 NOMINAL) ;

This rule will retain all subordinating conjunction (SCONJ) readings while discarding all others if the preceding token has a punctuation reading and the following token has a nominal reading.

The rules were developed by running a sentence of the train CoNLL-U file through the morphological analyser to find a list of its outputs. Based on the ambiguities of the output and the correct lemma in the corpus, rules were written to make the constraint grammar pick the correct interpretation. The sentence would then be re-run through the morphological analyser and the constraint grammar to confirm accuracy of the rule. Once a small set of about five rules was developed, all sentences were run through the morphological analyser and the constraint grammar rather than just the morphological analyser to see what wordforms were being disambiguated and which were not. If there were additional forms to be disambiguated, more rules would be written and the sentence would be re-run through the morphological analyser and constraint grammar.

In cases that were deemed to difficult to disambiguate, the wordforms were either left alone or a rule was written to partially disambiguate. The lemma būti-‘to be’, with its many interparadigmatically ambiguous forms, did not have rules written for it. Instead, however, existing rules would often correctly disambiguate the wordform entirely, as was the case with the sentence *Dabar ji yra iskilusi - žioji maždaug 5 cm plyšys* ‘It is now cracked, with a gap of approximately 5 cm.’ The following rules were used to disambiguate the wordform *yra*:

- SELECT:s8 $$PLURALITY IF (-1C $$PLURALITY) (1C $$PLURALITY) ;

- REMOVE:r15 VERB IF (1C VERB) ;

Some additional rules were written in consultation with Lithuanian grammar books. For example, in Lithuanian, when a transitive verb is used with a negative, the object is marked with genitive case rather than accusative case (Ramoniene and Pribusauskaite, 2008). This was reflected in the following rule:

- SELECT:s14 NOMINAL + GEN IF (*-1 VERB + NEG) ;

The sentence *Ganyklon karvašūdžiu žiūrėti tegu eina tos, kurios neturi televizoriu* 'The pasture cowboy let those who do not have a television go' demonstrates this case. The wordform *neturi* is a negative verb and consequently, the wordform *televizoriu* is correctly selected for a genitive reading over an accusative reading.

At this stage in the process, accuracy of rules was not of concern. If a rule made an incorrect reading of a wordform in a new sentence. The inaccuracy was disregarded and only remaining ambiguities were considered for new rules. In the development process, rules were evaluated for accuracy and modified or eliminated accordingly.

5 Development process

To test the rules in the constraint grammar and further develop it, a script was run. It took the dev CoNLL-U file of the Lithuanian ALKSNIS treebank and ran it through the analyser and lt.cg3 files and then compared the output of this process to the annotation in the dev file. This script outputs the true and false positives for each rule; the number of input, output, and reference analyses; input, output, and reference ambiguity; total true and false positives and negatives; and precision, recall, and F-score.

From the rule by rule output of the script, poorly performing rules could be eliminated or modified accordingly. Rules which were eliminated were simply commented out of the constraint grammar file as a means of keeping track of what rules have been discarded and to avoid rewriting a bad rule.

For final analysis of the performance of the constraint grammar, a similar script was run, using the test corpus.

6 Evaluation

6.1 Corpus analysis

The dev corpus of the Lithuanian ALKSNIS treebank on which the constraint grammar was tested consisted of 10,826 tokens. The test corpus contained 10,118 tokens. When running them through the morphological analyser and constraint grammar, the following results were computed:

Table 1: Ambiguity in the test corpus

	Input	Reference	Output
Analyses	12629	10118	10933
Ambiguity	1.25	1.0	1.08

The table above summarizes the ambiguity left in the test corp after being run through the constraint grammar. It can be seen that while the corpus had a noticeable reduction in ambiguity, there still remains plenty of ambiguity within the corpus.

The following tables demonstrate the performance of the constraint grammar through precision and recall.

Table 2: Precision and Recall with Disambiguation

	dev	test
Precision	.91	.90
Recall	.98	.98
F1 Score	.94	.94

Table 3: Precision and Recall without Disambiguation

	dev	test
Precision	.80	.80
Recall	1.0	1.0
F1 Score	.89	.89

As can be seen in the tables above, both corpora saw an increase in precision of about 10%, effectively reducing around half of ambiguity. Recall remained high after being run through the grammar at about 98% in each, and the F1 score had a similar improvement to that of the precision. Precision is defined as the number of true positives over all positives. Recall is defined as the number of true positives over the sum of true positives and false negatives. The F1 score is calculated by taking 2 times the product of precision and recall over the sum of precision and recall. True positives are correct readings that are retained by the grammar. True negatives are not found in the output of the analyser. False positives are readings the grammar retains but are not in the corpus. False negatives are correct readings the grammar removes.

In the following tables the number of true and false positives and negatives for both the dev and test corpora are presented:

Positives and Negatives in the dev corpus

	Positives	Negatives
True	10573	1699
False	1076	253

Positives and Negatives in the test corpus

	Positives	Negatives
True	9867	1445
False	1066	251

6.2 Correcting underperfoming rules

Upon the initial run of the run-grammar.sh program, it was evident that certain rules were underperforming, perhaps even incorrectly disambiguating more often than they were correctly disambiguating. One such rule was as follows:

- REMOVE:r3 CCONJ IF (-1 PUNCT) ;

This rule was intended to disambiguate words which could be coordinating conjunctions or particles. While in the first few sentences, the rule appeared to be working well, I did notice its frequent failure in later sentences while in the initial rule writing stage and working with the train corpus. Upon running the evaluation on the dev corpus, I discovered the rule was only applying correctly 24 out of 65 times, a mere 36% of the time. As a result, I simply deleted the rule without attempting to modify it in any way.

Another rule which had a poor performance was one which was made to select a singular reading over an underspecified reading if no possible reading was plural.

- SELECT:s6 SING IF (NOT 0 PLUR) ;

This rule performed just better than 50%, correctly disambiguating 41 of 80 tokens in the dev corpus.

In other cases, rules needed to be made more specific to properly capture the goal of the rule and increase its precision. The initial version of the rule below simply had the verb select the plurality of the noun following it. I discovered this frequently led to incorrect decisions when the following noun was not nominative. The rule had correctly disambiguated 35 out of 48, or about 72.9% of the time, in the dev corpus. As a result, I added the nominative requirement, on the assumption that the verb was conjugating for the subject, which typically will have a nominative case.

- SELECT:s25 VERBAL + $$PLURALITY IF (1 NOMINATIVE + $$PLURALITY);

The updated rule was very successful, correctly disambiguating 71 out of 76 tokens, or about 93.4% of the time, in the dev corpus and correctly disambiguating all 64 tokens it was used on in the test corpus.

7 Conclusion and future work

Thus far, the constraint grammar has been working relatively successfully. A lot of progress has been made without any prior knowledge of Lithuanian or constraint grammar. With precision sitting at about 90%, rules will still need to be added and fine-tuned to cover as much of the remaining gap as possible.

Only 170 of 1056 sentences in the train corpus have been run through the morphological analyser to develop the current set of 79 rules. With the remaining sentences to be run, the constraint grammar has the possibility of getting more fine-tuned rules that will disambiguate more precisely.

References

Agne Bielinskiene, Loic Boizou, Jolanta Kovalevskaite, and Erika Rimkute. 2016. Lithuanian Dependency Treebank ALKSNIS. *I. Skadiņa and R. Rozis (Eds.): Human Language Technologies – The Baltic Perspective*.

Cheikh M. Bamba Dione. 2014. Pruning the search space of the wolof lfg grammar using a probabilistic and a constraint grammar parser. *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

Jurgita Kapočiūtė-Dzikienė, Erika Rimkutė, and Loic Boizou. 2017. A comparison of lithuanian morphological analyzers. *Text, Speech, and Dialogue. TSD 2017. Lecture Notes in Computer Science, vol 10415*.

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

Terje Mathiassen. 1997. *Short Grammar of Lithuanian*. Slavica Publishers, Inc., Columbus, OH.

Bronius Piesarkas. 2006. *Didysis Lietuviu-Anglu Kalb Zodynas*. Zodynas Publishers, Vilnius, Lithuania.

Bronius Piesarkas and Bronius Svecevicius. 1995. *Lithuanian Dictionary English-Lithuanian Lithuanian-English*. Zodynas Publishers, Vilnius, Lithuania.

Meilute Ramoniene and Joana Pribusauskaite. 2008. *Practical Grammar of Lithuanian*. Baltos Lankos, Lithuania.

Robert Reynolds and Francis M. Tyers. 2015. A preliminary constraint grammar for russian. *Proceedings of the Constraint Grammar workshop at NODALIDA, the 20th Nordic Conference of Computational Linguistics*.

Trond Trosterud. 2009. A constraint grammar for Faroese. In *Proceedings of the NODALIDA 2009 workshop Constraint Grammar and robust parsing*.

Aurelija Usoniene, Nicole Nau, and Ineta Dabasinskiene. 2012. *Multiple Perspectives in Linguistic Research on Baltic Languages*. Cambridge Scholars Publishing, Newcastle upon Tyne, UK.