Statistical syntactic parsing for Latvian

Lauma Pretkalniņa, Laura Rituma

Institute of Mathematics and Computer Science, University of Latvia
Raiņa 29, Rīga, LV–1459, Latvia

lauma@ailab.lv, laura@ailab.lv

ABSTRACT

Syntactic parsing is an important technique in the natural language processing, yet Latvian is still lacking an efficient general coverage syntax parser. This paper reports on the first experiments on statistical syntactic parsing for Latvian — a highly inflective Indo-European language with a relatively free word order. We have induced a statistical parser from a small, non-balanced Latvian Treebank using the MaltParser toolkit and measured the unlabeled attachment score (UAS). As MaltParser is based on the dependency grammar approach, we have also developed a convertor from the hybrid dependency-based annotation model used in the Latvian Treebank to the pure dependency annotation model. We have obtained a promising 74.63% UAS in 10-fold cross-validation using only ~2500 sentences. The results revealed that best results can be achieved using non-projective stack parsing algorithm with lazy arc adding strategy, but comparably good results can be achieved using projective parsing algorithms combined with appropriate projectiviziation preprocessing.

KEYWORDS: Latvian, treebank, dependency parsing, statistical parsing, MaltParser.

1 Introduction and related work

Latvian is a highly inflective Indo-European language with a relatively free word order. Even though several experiments on automatic syntax parsing for Latvian have been carried out, it is still an open problem as no efficient general coverage parser is currently available. The closest one is an experimental rule-based partial parser based on a hybrid grammar model (Bārzdiņš, et al., 2007), however it uses hand-crafted rules that cover only simple extended sentences and is already highly ambiguous. Others address specific use-cases e.g. grammar checking (Deksne and Skadiņš, 2011) or the development of multilingual controlled languages (Paikens and Grūzītis, 2011). The aim of our research is to produce a general coverage statistical parser induced from currently available syntactically annotated data.

We use the Latvian Treebank (Pretkalniņa et al., 2011b) as the data source. The treebank is under active development, and it is still relatively small (approx. 2500 sentences) and by no means balanced, therefore acquiring good results for a language with a free word order is rather challenging. For current experiments we have chosen MaltParser as the parser induction framework mainly for two reasons. First, we consider that the hybrid grammar used in the Latvian Treebank is more in lines with the dependency formalism than with the phrase structure approach. Second, MaltParser is distributed in a handy, well documented implementation and can be trained in reasonable time. We plan to continue our experiments, comparing these results with what can be obtained with other data-driven approaches, e.g. (Koo and Collins, 2010).

The paper first gives an overview of the Latvian Treebank annotation model and transformations to make the data compliant with MaltParser (Chapter 2). In Chapter 3 we describe our experimental setup for MaltParser. In Chapter 4 we evaluate the results in terms of unlabeled attachment score.

2 Latvian Treebank

The Latvian Treebank is being developed since 2010 (Pretkalniņa et al., 2011a). It features multilayer annotations, containing both syntactical and morphological information. The approach used in the Latvian Treebank is similar to Prague Dependency Treebank's (Hajič et al., 2000) multilayered approach. The main difference is that Latvian Treebank has no tectogrammatical annotation layer yet, but some features of it are treated in the syntax layer. TrEd toolkit (Hajič et al., 2001) is used for annotating the Latvian Treebank. The annotation process is mainly done manually, and all the treebank data is human-verified. The analytical or syntactic layer is annotated using the SemTi-Kamols grammar model (Nešpore et al, 2010), further described in Chapter 2.1. The morphological layer is annotated by adding a positional tag to every token. Apart from morphological features, these tags represent also some lexical features like declension group. The tagset is derived from the annotation principles used for other languages in the MULTEXT-East project (Erjavec, 2010). Currently the Treebank contains 2519 sentences representing various genres (e.g. news and fiction), however, the genres are by no means balanced. There is an ongoing work on the corpus expansion and genre diversification. Up to our knowledge, it is the biggest syntactically annotated corpus for Latvian.

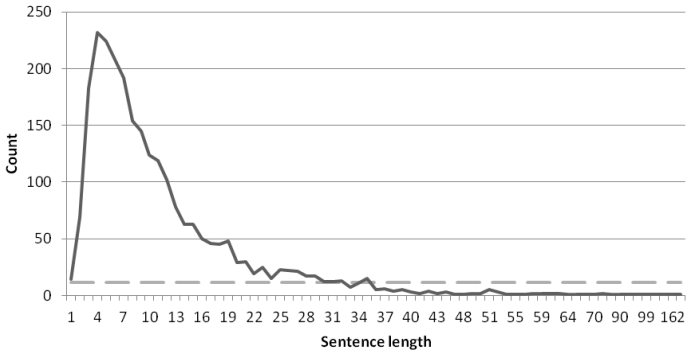

FIGURE 1: Latvian Treebank centence counts per length; average length shown dashed

Latvian Treebank contains 29 398 tokens. Sentence length varies from 1 to 200 tokens with average length on 11.67 tokens. 13.29% of all sentences are non-projective (measurement done after converting Treebank to pure dependency annotation, see section 2.2). For sentence length distribution see Figure 1.

2.1 Annotation model

The Latvian Treebank uses the hybrid dependency-based SemTi-Kamols annotation model (Pretkalniņa et al., 2011b). In essence, SemTi-Kamols annotations can be described as dependency trees that are augmented with phrase structure nodes (an example is given in Figure 2). Each phrase node can act as a dependency head or as a dependant. Each constituent in a phrase node can also act as a dependency head, but not as a dependant. Each constituent in a phrase node can be either a word or a phrase node.

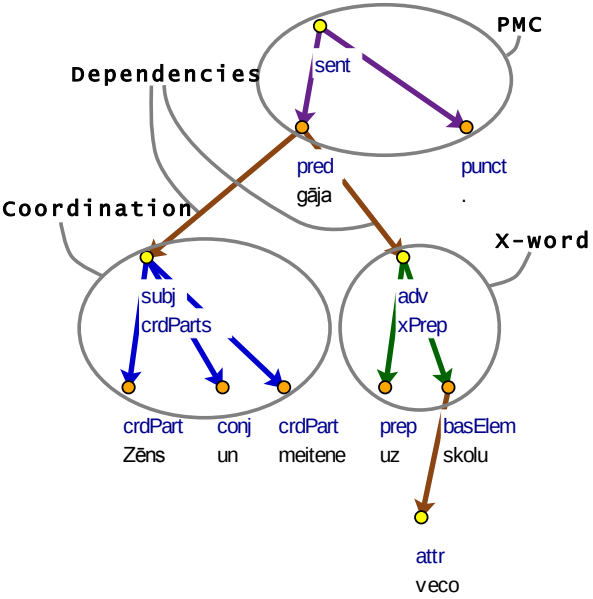

FIGURE 2: Latvian Treebank annotation example — *Zēns un meitene gāja uz veco skolu.* 'A boy and a girl went to the old school.'

Dependency links are used to depict subordination relations in a sentence. Apart from dependency relations, the Latvian Treebank has three types of phrase nodes: coordination, punctuation mark constructs (PMC), and the so-called x-words. Coordination nodes consist of coordinated parts of sentences, conjunctions, and punctuation marks that are used to bind together coordinated parts of sentences. Coordinated clauses are annotated in the same way. PMCs are introduced to annotate how punctuation marks in Latvian are used to show the syntactical structure. A PMC consists of one base element, a word or a phrase, and the punctuation marks whose usage are invoked by the base element, e.g. participle clauses and subordinated clauses. The x-words are the closest units to the classical understanding of phrases in the phrase structure grammars: they consist of several ordered words and are used to annotate constructions where the word order is important, e.g. analytical verb forms, multi-word numerals, named entities etc. For each type of a phrase node, several subtypes are distinguished depending on different features that the particular type of syntactic construction can represent. E.g. x-words are further divided in prepositional constructions, complex predicates, etc., but coordination is further divided in coordinated parts of sentence and coordinated clauses.

2.2 Transformation to pure dependency trees

In order to use the Latvian Treebank data for parser induction with MaltParser, we had to develop procedures for converting the hybrid SemTi-Kamols annotations to pure dependency annotations. Consulting with linguists, we have developed precise rules detailing how each type and subtype of phrase nodes is converted to a fragment of a dependency tree. Linking between constituents of phrase node is based on linguistic considerations taking into account the intention to develop these rules so that the obtained labelled dependency tree would contain as much from original information as possible, and that the trees eventually returned by the parser could be converted back as close to original model as possible. Currently, the SemTi-Kamols annotation scheme allows vast amount of all kinds of nested phrase structures, e.g. coordination of coordination (of coordination, etc.). As SemTi-Kamols also makes distinction between dependents of phrase (as whole) and dependents of its constituents, it is more expressive than the pure dependency model. In future we plan to use this information as additional features for parser training and to explore other parsing algorithms to obtain parser for full annotation model. These rules are implemented in a treebank conversion script.

According to the conversion rules (a transformation example is given in Figure 3), each phrase is converted to a dependency tree fragment which is a valid dependency tree itself. The constituents of a phrase structure node are each transformed to a single dependency tree node, and all dependents of these constituents (in the original tree) remain as dependents of the corresponding nodes in the dependency tree. Nodes connected to a phrase node as a whole become connected to the root node[1] of the subtree representing the phrase node after the transformation. The constituent that becomes the root node of the subtree is determined by the type, subtype and structure of the phrase. Whenever it is possible, we choose a constituent that cannot have dependents itself, so that dependents of a phrase node as a whole and dependents of the head constituent would not mix. In a coordination phrase, the root node is a conjunction (or punctuation, if there is no

[1] The head of the phrase structure.

conjunction). In punctuation mark constructs, the root node is a punctuation mark (the first, if multiple occurs in the phrase). For x-words the root node differs depending on the subtype of the x-word, e.g. the preposition in a prepositional construction, the last numeral in a multi-word numeral, etc. All other constituents are either directly linked to the chosen root, or chained in the order they occur in the sentence, and then linked to the chosen root. The way in which other constituents are added to the root depends on the phrase type, e.g. multi-word numerals are chained, but PMC constituents are linked directly to the root. If no other considerations are available, all other constituents are added as dependents of the last constituent.

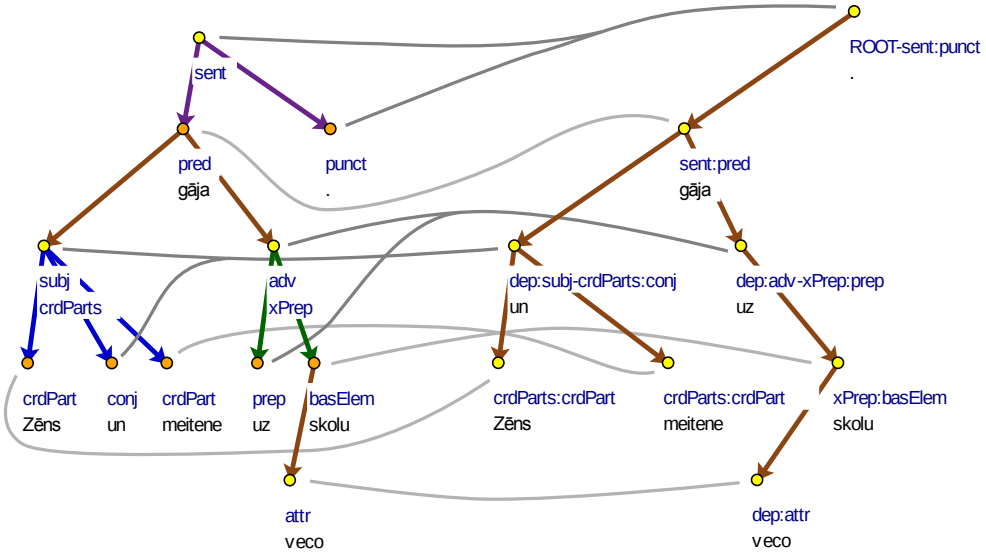

FIGURE 3: Example of transforming a hybrid tree (on the left) to a pure dependency tree (on the right). Corresponding showed with gray lines.

During the transformation, the syntactic roles of the obtained dependency tree can be augmented to reveal the full information needed to convert the trees back to the hybrid grammar representation, but this can lead to potentially infinite set of roles: when the chosen root element for some phrase is a phrase itself, we need to add additional information to the role labels, and, theoretically, phrases can stack infinitely. In practice, at least three levels deep stacking has been observed in the treebank. Thus, we still need to explore the linguistic background and the actual data to find the best way to define the set of syntactic roles for the obtained dependency trees. As the original treebank data contain annotations with sentence-scope ellipsis, appropriate roles for displaying this information in dependency trees are also needed.

3 Experimental setup

Currently, state of the art parsers (Bohnet and Nivre, 2012) achieve up to 93% unlabeled attachment score for English — a highly analytic language — and up to 88.8% for Czech which is grammatically more similar to Latvian. Acquiring comparably good parsing results for Latvian is highly challenging due to the very limited size of the available data set and

due to the rich annotation model used in the treebank. The data set consists of 2519 manually annotated sentences (29496 tokens). To obtain a better understanding of what parsing accuracy limitations are imposed by such corpus size, we use the treebank's original morphological tagging (manually verified) for accuracy testing. However, to get small insight about errors caused by possible mistakes in statistical morphological tagging we perform single experiment with POS tagged data, using statistical tagger, described by Paikens et al. (2013).

For parser induction we use MaltParser, version 1.7.1[2]. Following the advice given by Nivre and Hall (2010) we use LIBSVM (Chang and Lin, 2011) as the learning agent to avoid necessity of manual feature combination and cross-validation to evaluate our parsers, again — due to the limited size of the data set. We use 10-fold cross-validation in terms of sentence count and report the minimal, maximal and average score for each parsing experiment. To obtain a more even distribution, we form the validation sets by selecting approx. 10% of sentences of each document in treebank.

We have performed several experiments with various parsing algorithms and various representations of morphological features. As the Latvian Treebank contains non-projective annotations (e.g. parenthesis, direct speech), for the baseline we have performed a series of experiments with the non-projective stack algorithm (Nivre, 2009; Nivre et al. 2009) with MaltParser's default feature configuration (tags and wordforms). First, we trained a parser with the full morphological tag. To reduce the sparseness of the morphological tags (the full tagset has approx. 500 different tags), we continued with an experiment with a dataset where the lexical features are removed from the tags. Finally, we split the positional tags into sets of multiple features representing each morphological feature separately to increase the parser's ability to learn regularities like "an adjective depending on a noun has to agree on the number, case and gender".

In the next step, we performed a series of experiments using various parsing algorithms provided by the MaltParser toolkit and combining their settings to see how the algorithm choice influences the parsing accuracy. We used the non-projective stack algorithm, both eager and lazy versions, and several projective algorithms combined with different pseudo-projective tree processing. We run both experiment series with no pseudo-projective transformations and series where we used arc-eager, arc-standard (Nivre, 2003; Nivre, 2004) and the stack projective algorithms (Nivre, 2009; Nivre et al., 2009). We used head, path and head+path marking strategies for pseudo-projective transformations (Nivre and Nilsson, 2005). We also performed experiments with planar and 2-planar algorithms (Gómez-Rodríguez and Nivre, 2010).

In the last step, we added additional features containing lemmas to the dataset and then run best-performing algorithms from the previous experiment sessions.

For evaluating the parser performance, we used the unlabeled attachment score (UAS), calculating it using the script provided in the CoNLL-07 shared task toolbox[3].

[2] Available at http://www.maltparser.org/ [last visited on 26/01/2013]
[3] Available at http://nextens.uvt.nl/depparse-wiki/SoftwarePage [last visited on 26/01/2013]

4 Evaluation

The first series of experiments — with the non-projective stack parser using forms and morphological tags as features — gave the following results: with no additional features, the data set with smaller tag sparseness performs better. By adding additional features — decoding each morphological tag into separate positions, the results get better, and the difference between full or pruned tags usage becomes insignificant as LIBSVM does feature combination itself. On these data, the stack-lazy algorithm tends to achieve slightly better results. During these experiments we have achieved 74% UAS. The full results of the first experiment series are shown in Table 1. Thus, the further experiments were done with a dataset containing full morphological tags and decoded features.

Algorithm	Stack eager				Stack lazy			
Feature decoding	No		Yes		No		Yes	
Tag type	Full	Short	Full	Short	Full	Short	Full	Short
AVG %	69.24	71.95	73.78	73.47	69.77	72.55	**74.15**	**74.04**
MIN %	65.59	69.75	71.51	71.02	67.22	69.57	71.41	71.17
MAX %	72.53	73.79	77.72	77.9	73.79	75.91	77.77	78.68

TABLE 1: Non-projective stack parser training: 10-fold cross-validation results.

Algorithm	Nivre's arc-eager				Nivre's arc-standard				Stack projective			
Projecti-vization	No	Head	Path	Head, path	No	Head	Path	Head, path	No	Head	Path	Head, path
AVG %	71.96	73.66	73.78	73.78	70.77	73.71	73.47	73.47	71.89	**74.03**	73.74	73.74
MIN %	69.36	71.2	70.8	70.8	68.54	70.5	70.6	70.6	69.47	71.97	70.86	70.86
MAX %	76.34	77.38	77.85	77.85	75.56	77.12	76.86	76.86	76.3	76.38	76.95	76.95

TABLE 2: Pseudo-projective parser training: 10-fold cross-validation results. Full morphological tags are used.

The next series of experiments compare different projective algorithms and different pseudo-projective transformations. These results show that the pseudo-projective approach gives equivalently good results compared to the non-projective stack algorithm. The chosen algorithm and projectivization strategy does not matter much — less than 0.5% points. Using the projectivization improves the UAS for 1–2% points over not using the projectivization at all. The full results are shown in Table 2.

Algorithm	Planar	2-planar	Stack lazy	Stack projective with head coding
AVG %	72.38	71.96	**74.15**	**74.03**
MIN %	69.57	70.26	71.41	71.97
MAX %	76.9	76.25	77.77	76.38

TABLE 3: Summary on parser training algorithms: 10-fold cross-validation results.

In the last step, to compare the different parsing algorithms, we have used the planar and 2-planar algorithms on the same data, but they fail to repeat success of the stack parsing algorithm. The comparison between these two algorithms, the best performing projective algorithm and the best performing non-projective stack algorithm is given in Table 3.

Algorithm	Stack eager	Stack lazy	Stack projective with head coding	Nivre's arc-eager with path encoding	Nivre's arc-standard with head encoding
AVG %	74.21	**74.63**	74.57	74.47	74.49
MIN %	71.22	71.51	71.86	71.44	72.16
MAX %	78.63	79.24	77.98	79.11	77.9

TABLE 4: Summary on the parser training with lemmas: 10-fold cross-validation results.

Finally, we have retrained five best performing parsers on data augmented with lemmas. This gives UAS increase of approx. 1.5% point using stack-lazy algorithm, thus, giving the best UAS achieved — 74.63%. This is achieved by non-projective stack algorithm with the lazy swap strategy. The full results are shown in Table 4.

Algorithm	Stack lazy on morpho-tagged data
AVG %	72.2
MIN %	69.39
MAX %	76.95

TABLE 5: Parser training with on automatically tagged data: 10-fold cross-validation results.

To get some insight about impact of the morphological tagger inaccuracies to the parsing accuracy we redo experiment with best performing parser configuration (stack-lazy algorithm, lemmas) with Treebank retagged with statistical morphological tagger (Paikens et al., 2013). Morphological tagger also adds some lexical features not available in manual tagging, e.g. source lemma for verbs derived with prefixes and deverbal nouns. Result is given in Table 3. This gives UAS decrease by approx. 2.4%, but this should be viewed as rather preliminary result.

5 Conclusion and future tasks

We have presented first general coverage statistical (data-driven) parser for Latvian. The parser is induced from a small Latvian Treebank with the help of the MaltParser toolkit. The parser achieves 74.63% unlabeled attachment score with gold standard morphological tags and 72.2% UAS — with automatically tagged morphology, which we consider a good result taking into account, that only 2500 annotated sentences were available for the parser development and that Latvian has rich morphology and, thus, a relatively free word order. We have also briefly described the current state of the Latvian Treebank and have laid out our methodology for transforming the treebank from a hybrid annotation model to the pure dependency trees.

Our experiments show that the best results can be obtained using the non-projective stack algorithm with the lazy arc adding strategy. Almost as good results can be achieved combining the projective stack algorithm with the head marking strategy for the projectivization transformation, or Nivre's arc-eager algorithm with path marking, or Nivre's arc-standard algorithm with head marking. By decoding morphological tags into separate features and providing lemmas as additional features, it gave us significant increase of UAS — 2% points total.

Our future work will include the use of dependency labels adapted to allow the transformationing back to original hybrid annotation format and to do labelled parsing

experiments. We also plan to compare the current results with what can be achieved with a parser proposed by Koo and Collins (2010). The further work will include the estimation of accuracy decrease when a statistical morphological tagger is used instead of human-verified tags. We also plan to compare these results with the approach proposed by Bohnet and Nivre (2012) to find out if it can improve the accuracy when a comparatively smaller treebank is available. In the future we hope to expand our research beyond the pure dependency parsers to use to the full extent the hybrid annotations that the Latvian Treebank has. We also plan to work with lexicalization of our parsers, but this will require more work on resource development.

6 Acknowledgements

This work is supported by ERDF project „Informācijas un komunikāciju tehnoloģiju kompetences centrs" nr. KC/2.1.2.1.1/10/01/001, agreement L-KC-11-0003 research 2.7 "Teksta automātiskas datorlingvistikas analīzes pētījums jauna informācijas arhīva produkta izstrādē". We thank the University of Latvia for the financial support in preparing and presenting this paper and the anonymous reviewers for their improvement recommendations.

References

Bārzdiņš, G., Grūzītis, N., Nešpore, G. and Saulīte, B. (2007). Dependency-Based Hybrid Model of Syntactic Analysis for the Languages with a Rather Free Word Order. In: *Proceedings of the 16th Nordic Conference of Computational Linguistics*, pages 13–20, Tartu.

Bohnet, B. and Nivre, J. (2012) A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM : a library for support vector machines. In *ACM Transactions on Intelligent Systems and Technology*, 27(2), pages 1–27.

Deksne, D. and Skadiņš, R. (2011). CFG Based Grammar Checker for Latvian. In *Proceedings of the 18th Nordic Conference of Computational Linguistics*, pages 275–278 Riga.

Erjavec, T. (2010). MULTEXT-East Version 4: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC'2010)*, pages 19–21, Malta.

Gómez-Rodríguez, C. and Nivre, J. (2010). A Transition-Based Parser for 2-Planar Dependency Structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501

Hajič, J., Böhmová, A., Hajičová, E. and Vidová Hladká, B. (2000). The Prague Dependency Treebank: A Three-Level Annotation Scenario. A. Abeillé (ed.): *Treebanks: Building and Using Parsed Corpora*, pages 103–127, Amsterdam, Kluwer.

Hajič, J., Vidová Hladká, B. and Pajas, P. (2001). The Prague Dependency Treebank: Annotation Structure and Support. In *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 105–114, Philadelphia.

Koo, T. and Collins, M. (2010). Efficient Third-order Dependency Parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Association for Computational Linguistics.

Nešpore G., Saulīte B., Bārzdiņš G. and Grūzītis N. (2010). Comparison of the SemTi-Kamols and Tesnière's Dependency Grammars. In *Proceedings of the 4th International Conference on Human Language Technologies — the Baltic Perspective*. Frontiers in Artificial Intelligence and Applications, Vol. 219, pages. 233–240, IOS Press.

Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pages 149–160, Nancy.

Nivre, J. (2004). Incrementality in Deterministic Dependency Parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together*. *Workshop at ACL-2004*, Barcelona.

Nivre, J. (2009). Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th*

International Joint Conference on Natural Language Processing of the AFNLP, pages 351–359.

Nivre, J. and Nilsson, J. (2005). Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Nivre, J., Kuhlmann, M. and Hall, J. (2009). An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09), pages 73–76*.

Nivre J. and Hall J. (2010). A Quick Guide to MaltParser Optimization. http://maltparser.org/guides/opt/quick-opt.pdf [last visited on 16/01/2013].

Paikens P., Grūzītis N. (2012). An implementation of a Latvian resource grammar in Grammatical Framework. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC)*, pages 1680–1685, Istanbul.

Paikens P., Rituma L., and Pretkalniņa L. (2013). Morphological analysis with limited resources: Latvian example. In *Proceedings of 19th Nordic Conference of Computational Linguistics*, to be published, Oslo.

Pretkalniņa L., Nešpore G., Levāne-Petrova K., and Saulīte B. (2011a). A Prague Markup Language Profile for the SemTi-Kamols Grammar Model. In *Proceedings of the 18th Nordic Conference of Computational Linguistics*, pages 303–306, Riga.

Pretkalniņa L., Nešpore G., Levāne-Petrova K., and Saulīte B. (2011b). Towards a Latvian Treebank. In *Actas del 3 Congreso Internacional de Lingüística de Corpus. Tecnologias de la Información y las Comunicaciones: Presente y Futuro en el Análisis de Corpus*, pages 119–127, Valence.