LFG without C-structures

Özlem Çetinoğlu[1], Jennifer Foster[1], Joakim Nivre[2],
Deirdre Hogan[1], Aoife Cahill[3], Josef van Genabith[1]

[1]Dublin City University, School of Computing
[2]Uppsala University, Department of Linguistics and Philology
[3]University of Stuttgart, IMS
E-mail: [1]{ocetinoglu,jfoster,dhogan,josef}@computing.dcu.ie
[2]joakim.nivre@lingfil.uu.se [3]aoife.cahill@ims.uni-stuttgart.de

Abstract

We explore the use of two dependency parsers, Malt and MST, in a Lexical Functional Grammar parsing pipeline. We compare this to the traditional LFG parsing pipeline which uses constituency parsers. We train the dependency parsers not on classical LFG f-structures but rather on modified dependency-tree versions of these in which all words in the input sentence are represented and multiple heads are removed. For the purposes of comparison, we also modify the existing CFG-based LFG parsing pipeline so that these "LFG-inspired" dependency trees are produced. We find that the differences in parsing accuracy over the various parsing architectures is small.

1 Introduction

Phrase structure parsing has come a long way in the last decade. Techniques such as iterative chart pruning (Charniak et al., 2006) and cell-closing (Roark and Hollingshead, 2008) have been used to speed up parsing, and discriminative reranking (Charniak and Johnson, 2005) and latent variable grammar induction (Matsuzaki et al., 2005; Petrov et al., 2006) have pushed Parseval f-scores on the standard Wall Street Journal test set over the 90% mark. In the same time period, dependency parsing has also flourished (Yamada and Matsumoto, 2003; Nivre and Nilsson, 2005; McDonald et al., 2005; Nivre et al., 2007). Dependency trees are often viewed as a more intuitive and less anglocentric way of representing syntactic phenomena, and a multilingual dependency parsing system such as Malt allows such structures to be produced in linear or at worst quadratic time (Nivre et al., 2006; Nivre, 2009). In the context of these two types of linguistic representation, Lexical-Functional Grammar (LFG) (Bresnan, 2001) is an interesting linguistic theory since it has a foot in both camps, encoding constituent structure

(c-structure) using context-free constituency trees and grammatical dependencies between the main words in a sentence using directed acyclic graphs (f-structures).

In recent years there have been two main approaches within the LFG community to parsing text into LFG c-structures and f-structures: one in which hand-crafted LFG grammars are used in conjunction with unification-based parsing (Kaplan et al., 2004; Maxwell and Kaplan, 1996) and another (Cahill et al., 2004, 2008) which uses treebank-based resources and an LFG f-structure Annotation Algorithm (AA) which decorates nodes in treebank or parser-output CFG trees with LFG functional equations which are then passed to a constraint solver to produce f-structures. The advantages of the treebank-based approach to LFG-parsing include substantially reduced grammar development time, high-quality, wide-coverage and robust output, and the fact that the approach can be applied to languages for which no hand-crafted wide-coverage LFG grammar is available but for which a CFG treebank exists. LFG parsing pipelines (both those based on hand-crafted and treebank resources) are CFG-based: a constituency parser produces trees and these trees carry functional annotations from which a constraint solver can produce an f-structure. Strong advances in dependency parsing in terms of both speed and accuracy, and the fact that, for some languages (e.g., Turkish (Oflazer et al., 2003)), only a dependency bank is available, raise interesting research questions: *is it possible to directly parse strings into LFG f-structures, obviating the CFG-parsing step in traditional LFG parsing architectures?*[1] *Do current dependency parsing technologies require changes in the f-structure level of representation and what are the accuracy results for direct dependency-parsing-based LFG models as compared to CFG-based pipelines?*

We attempt to answer these questions by experimenting with the Malt and MST parsing systems in treebank-based LFG parsing and comparing the results we obtain to those obtained using the Brown and Berkeley constituency parsers. In the direct dependency LFG parsing pipeline, the dependency parsers are trained on LFG-inspired dependency trees, obtained by modifying the f-structures produced by the AA from the original PTB constituency trees. In the CFG-based LFG parsing pipeline, constituency parsers are trained on PTB trees and the AA is applied to the parser output, yielding f-structures. For evaluation, these are then converted to dependency trees using the same procedure which was used to produce the training material for the dependency parsers. We document and discuss the implications of all design decisions which were applied in order to convert the AA output so that it can be used with Malt and MST.

The paper is organised as follows: in Section 2, we describe LFG in more detail and we show how the AA is applied to the output of constituency parsers to obtain LFG f-structures; Section 3 details the changes that were made to the AA-based LFG f-structure DAGs so that they could be used with parsers which directly produce dependency trees; the parsing experiments are described in Section 4; related work is discussed in Section 5; finally, Section 6 summarises the main points of the

[1] Guo et al. (2008) explore a similar question for LFG-based generation.

paper and suggests pointers for future work.

2 LFG and Treebank-based LFG Acquisition

Lexical-Functional Grammar (Bresnan, 2001) is a constraint-based theory of grammar with minimally two levels of representation: c(onstituent)- structure and f(unctional)-structure. C-structure (CFG trees) captures language specific surface configurations such as word order and the hierachical grouping of words into phrases, while f-structure represents more abstract, language independent grammatical relations (essentially bilexical labelled dependencies with some morphological and semantic information, approximating to basic predicate-argument structures) in the form of attribute-value structures or directed acyclic graphs (DAGs). F-structures are defined in terms of equations annotated to nodes in c-structure (grammar rules).

Treebank-based LFG acquisition was originally developed for English (Cahill et al., 2004, 2008) and is based on an f-structure annotation algorithm pipeline. The pipeline has four main components: a constituency parser creates PTB style trees. The function labeller FunTag (Chrupała et al., 2007) enriches the parser output trees with the PTB function labels. The function labelling component of the pipeline is optional. The core annotation algorithm annotates trees with f-structure equations, which are read off the tree and passed on to a constraint solver producing a proto f-structure for the given sentence. Finally, the long-distance dependency component resolves non-local dependencies using automatically acquired subcategorisation frames and finite approximations of functional-uncertainty equations (Cahill et al., 2004).

3 LFG-Inspired Dependencies

The LFG AA takes PTB style trees and generates LFG f-structures. In order to use the output of the AA to train the dependency parser, we convert the LFG f-structure DAGs to dependency trees in the CoNLL format. LFG f-structures are recursive attribute-value structures close to but not exactly the same as the bilexical dependencies assumed in CoNLL format: LFG f-structures are somewhat more abstract and, unlike for CoNLL-style dependencies, not every token in a string is represented as a node (i.e. as the value of a PRED attribute) in the f-structure (e.g. auxiliary sequences in languages with analytic tense are represented in terms of abstract tense/aspect features; punctuation is not represented at f-structure etc.). Words are represented as lemmas rather than surface forms and properties of strings (tense, aspect, mood, statement-type, number, person etc.) are encoded in terms of features. Non-local dependencies are represented in terms of coindexation (reentracies in the graph) and dependencies can be multi-headed.

In order to use the output of the AA to directly train Malt and MST, we convert the LFG f-structures to dependency trees by carrying out the following modifications:*i)* representing each token in the f-structure *ii)* removing dependencies that

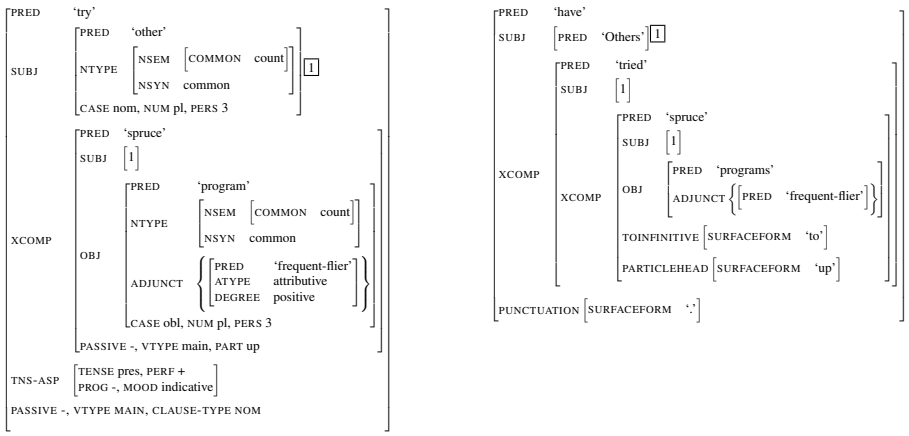

Figure 1: LFG f-structure for *Others have tried to spruce up frequent-flier programs.* in the original form (left) and as bilexical dependencies (right)

result in multiple heads *iii)* avoiding multiple roots.

Representing each token in the f-structure: in LFG some words map to atomic-valued features in the f-structures, rather than semantic forms. Moreover, punctuation is not explicitly represented. In the CoNLL format, every token of a sentence and its dependency relation has to be explicitly stated. In order to obtain a bilexical dependency for every token in the input, we modify the AA to turn f-structures into full bilexical dependencies, making sure that every token in the sentence is explicitly represented as a PRED (or SURFACEFORM)-valued node. The original AA produces the f-structure on the left in Figure 1 for the sentence *Others have tried to spruce up frequent-flier programs.* The auxiliary *have* is not explicitly represented in the f-structure but contributes to the values of the TENSE, PERF and MOOD features. Similarly, the particle *up* does not have a PRED but is represented as a feature of the verb *spruce*. The modified f-structure for the same sentence is given on the right. The converted f-structure has an explicit representation for the auxiliary *have*, the verbal particle *up*, the infinitival marker *to* as well as for the punctuation marker. Atomic-valued features like CASE, NUM, TENSE are removed. The PRED features now represent the surface form of the tokens rather than word lemmas.

Removing dependencies that result in multiple heads: For many non-local dependencies such as wh-elements in relative clauses, topicalisation, and subject/object control, LFG (and the LFG AA) assigns multiple heads to a word. In the example on the right in Figure 1 *others* is the subject of both *have*, *tried* and *spruce up*. Multiple heads are not supported in the dependency tree representations used by, for example, Malt.

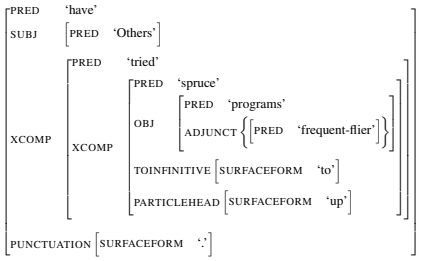

Figure 2: F-structure as bilexical dependencies with multiple heads removed

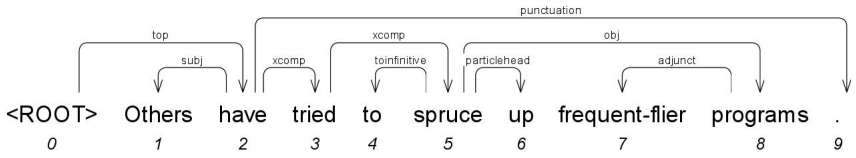

Figure 3: Dependency representation of the example sentence

In order to avoid multiple heads we follow two simple strategies. If the multi-headed construction involves a discourse function (TOPIC, TOPICREL, FOCUS), we remove this dependency, but (crucially) keep the dependency to the local head to capture the non-local dependency. Otherwise we keep the dependency relation with the head at the outermost level of the f-structure and remove the other dependencies, as e.g. for cases of control. Our intention is to keep more informative dependencies and ones that cause less non-projectivity.

The f-structure on the right in Figure 1 contains a multi-headed dependency of the latter type. We keep the relation between *others* and *have*, and remove the other dependencies. The resulting simplified f-structure is given in Figure 2. The corresponding dependency representation for this f-structure is given in Figure 3.

Avoiding multiple roots: In the current version of the LFG-converted training data, there are tokens without heads, arising from inadequacies in the LFG AA or the f-structure-to-dependency-tree conversion procedure (detailed above). This leads to f-structure fragments and corresponding multiple roots in the dependency trees for a particular string. We automatically make heads of such fragments dependent on a dummy root node ROOT with a dummy label `dep`.[2]

In the LFG-inspired dependency conversion, we exclude PTB trees with FRAG-(ment) constituents from the training data, as the LFG AA is not designed to deal with such structures. For the phrases marked as X (unknown, uncertain, or unbracketable) in the PTB, we use the dummy dependency relation `dep`. In some

[2]Currently this happens in 1.03% of the converted f-structures.

	Multi-head	**Single-head**
# sentences	39132	39163
# dependency types	25	25
multi-headed dependencies	7%	0%
non-projective dependencies	4.16%	0.15%
non-projective sentences	63.76%	2.52%
head left of modifier	51%	53%

Table 1: WSJ sections 02-21 conversion statistics for LFG bilexical f-structures with multi-headed and single-headed dependencies

adjunct	adjunct	poss	possesive
app	apposition	∗possmarker	possesive marker *'s*
comp	complement	∗prepositionhead	used for *so that*, *so as to*, *as if*,... these are MWEs in LFG
coord	coordination item	∗punctuation	punctuation
∗dep	dependency (dummy)	quant	quantifier
det	determiner	relmod	relative modifier
focus	focus	subj	subject
obj	object	∗toinfinitive	to infinitive
obj2	2nd object (obj-th in LFG)	∗top	top (root of a dependency tree)
obl	oblique object	topic	topic
∗obl2	2nd oblique object	topicrel	relative topic
obl-ag	oblique agent	xcomp	open complement
∗particlehead	head of a particle		

Table 2: LFG-inspired conversion tagset

cases, the AA produces f-structures for only a small fragment of the tree and there are no explicit dependency relations for the remaining words in the sentence. Since we cannot obtain dependency relations for all tokens, we omit these sentences. At this stage, the number of trees/f-structures in Sections 02-21 of the PTB correctly converted to dependency trees is 39,163, i.e. 99.51% of the standard training set. The work on the LFG dependency conversion is ongoing with the objective to eventually cover the whole training set.

The training set has different characteristics when multi-headed dependencies are allowed and when multi-heads are removed to obtain single-headed dependencies. The differences can be observed in Table 1. The percentage of multi-headed dependencies is 7% and when they are removed, the non-projective dependencies decrease to 0.15% from 4.16%. This has a drastic effect on the number of sentences with at least one non-projective dependency, which drops down to 2.52% from 63.76%. The LFG-inspired conversion tagset consists of 25 dependencies. Table 2 lists those dependencies with their descriptions. The tagset is based on core LFG grammatical functions present in the original LFG DAGs. The labels marked with an asterisks indicate the additional dependencies that are not originally present in the LFG theory. The additional labels are used when the more abstract f-structure DAGs are modified to represent more surfacy bilexical dependencies in dependency trees that cover all tokens in the input strings.

4 Parsing Experiments

In this section, we describe our LFG parsing experiments. The four parsers we use are described in Section 4.1, the experimental procedure is detailed in Section 4.2 and the results are presented in Section 4.3.

4.1 Parsers

Berkeley: The Berkeley parser (Petrov et al., 2006) is a generative constituency parser which parses using an unlexicalised yet fine-grained smoothed PCFG. This PCFG is obtained in an iterative process of splitting the treebank non-terminals into subcategories, estimating the parameters of the resulting grammar using Expectation Maximisation and merging the less useful category splits. For efficient parsing, multi-stage coarse-to-fine pruning using the intermediate grammars obtained during training is carried out (Petrov and Klein, 2007). We train a grammar using 6 split-merge cycles and we run the parser in *accurate* mode, meaning that the pruning thresholds are tuned for accuracy at the expense of speed.

Brown: The Brown parser (Charniak, 2000) is a generative constituency parser which uses a head-lexicalised smoothed PCFG which is conditioned on the parse history and which combines five probability models fine-tuned for English. In our experiments, we use both this parser and the reranking version in which the n-best list returned by the generative parser is re-ordered using a discriminative reranker trained on features which are unavailable to the original parser (Charniak and Johnson, 2005). We employ these parsers in their *out-of-the-box* settings.

MaltParser: MaltParser is a flexible multi-lingual dependency parsing system (Nivre et al., 2006). During training a classifier is induced to predict a parsing action at a particular parsing configuration using information from the parse history and the remaining input string. During parsing, the classifier is used to drive the deterministic construction of a dependency tree. MaltParser can be used with several parsing algorithms including variants of shift-reduce parsing. We use the *stacklazy* algorithm, which employs a swap operation so that non-projective structures can be handled (Nivre, 2009). Following Attardi and Ciaramita (2007) we train a linear classifier where the feature interactions are modelled explicitly.

MSTParser: While MaltParser learns to predict parsing actions, MSTParser (McDonald et al., 2005), learns to predict entire dependency trees. The parser finds the maximum spanning tree in a multi-digraph using one of several algorithms described in McDonald (2006). For our experiments, we use the second-order approximate non-projective parsing model introduced in McDonald and Pereira (2006), which parameterizes dependency trees by pairs of adjacent sibling arcs and uses hill-climbing to find the highest scoring (possibly non-projective) tree, starting from the highest-scoring projective tree derived by dynamic programming (Eisner, 1996). MSTParser can be run in one or two stages. In the two-stage model, an unlabelled tree is predicted and the labeling of dependency arcs is carried out during the second stage. We employ the one-stage parser which directly predicts

labels.

4.2 Procedure

In the **LFG constituency parsing pipeline**, *i)* The constituency parsers are trained in the usual way on the PTB (function tags and traces are excluded from the training trees). *ii)* Input sentences are parsed with the parsers. *iii)* The parse trees are passed through the FunTag labeller (Chrupala et al., 2007) which assigns Penn-II treebank function tags to raw CFG parser output trees. We also carry out the experiment with this step omitted. *iv)* The LFG AA is applied to the constituency trees producing LFG f-structures. *v)* Parser output f-structures are converted to dependency trees. *vi)* The output is evaluated against the LFG-style dependency trees for WSJ Sections 00 and 23.

In the **LFG dependency parsing pipeline**, *i)* The AA is applied to the PTB training data producing f-structures. *ii)* The f-structures are converted into dependency trees. *iii)* The dependency parsers are trained on the LFG-inspired dependency trees. *iv)* Input sentences are parsed with the dependency parsers. *v)* The output is evaluated against the LFG-style dependency trees for Sections 00 and 23.

Our training data consists of Sections 02-21 of the WSJ section of the PTB. We use Section 00 as our development set to tune the MaltParser feature model and to perform error analysis, and present final results on Section 23. We experiment with the use of gold POS tags, POS tags obtained using a POS tagger (Giménez and Màrquez, 2004) and, for Brown and Berkeley, POS tags produced by the parsers themselves.[3] We use the CoNLL evaluation metrics of labelled attachment score (LAS) and unlabelled attachment score (UAS).[4]

4.3 Results

Evaluation results on Section 00 and on Section 23 are given in Table 3. We observe that using FunTag leads to a 3% increase for constituency parsers. The Brown+Reranker+FT architecture has the highest scores, outperforming even the gold-POS-tagged systems. Constituency parsers with FunTag rank higher than dependency parsers, but differences between the systems are small. The trends for Section 00 carry over to Section 23. The main difference is that the two dependency parsers, Malt and MST, suffer a greater drop in accuracy when using predicted rather than gold POS tags.

4.4 Discussion

We examine Section 00 accuracy scores broken down by dependency type for all parsing systems by picking the best non-gold-POS-tagged architecture for each

[3]Note that the Brown parsers always perform their own POS tagging.

[4]For replicability, we provide all experimental settings at http://www.nclt.dcu.ie/gramlab/experiments.html

POS	Parser	Section 00		Section 23	
		LAS	**UAS**	**LAS**	**UAS**
Gold	Berkeley	87.38	91.71	87.63	91.48
	Berkeley+FT	**90.51**	**92.10**	**90.81**	**92.27**
	Malt	89.02	90.64	89.01	90.59
	MST	89.79	91.68	89.97	91.73
OwnTag	Berkeley	86.82	91.38	87.12	91.19
	Berkeley+FT	89.97	91.8	90.29	91.95
	Brown	86.30	90.93	86.56	90.81
	Brown+FT	89.54	91.3	89.75	91.48
	Brown+Reranker	87.55	92.24	87.72	92.04
	Brown+Reranker+FT	**90.81**[1]	**92.59**[3]	**91.13***	**92.81***
Predicted	Berkeley	86.97	91.42	86.51	90.60
	Berkeley+FT	90.11[2]	91.83[4]	89.61*	91.33*
	Malt	88.66*	90.41*	87.57*	89.47*
	MST	89.43*	91.47*	88.51*	90.59*

Table 3: Accuracy scores on Sections 00 and 23 for the LFG-inspired conversion (*p-value (1)&(2)=0.004; p-value (3)&(4)=0.002; For all other comparisons marked with *, p-value $\ll$ 0.001*)

parser. All four systems have over 95% scores for subjects and objects. The performance on adjuncts is almost the same ($\sim$ 88%) in all systems. MST outperforms all other systems in identifying thematic objects (`obj2`), followed by Malt. Constituency parsers suffer on `obj2` with scores lower than 50%. Coordination f-score for constituency parsers is 85% while dependency parsers perform slightly worse with a 80% f-score. The accuracy of all parsers drop down to the 70%-80% range for relative modifiers. The dummy `dep` relation is rarely used by the LFG-inspired conversion and only identified by Brown+Reranker+FT albeit with very low scores.

The breakdown shows that thematic objects, relative clauses and coordination are harder to recover regardless of the parsing system. Since the `dep` relation does not represent a specific linguistic phenomenon, it is not possible to identify it either for constituency parsers or dependency parsers. The small difference between the constituency and dependency parsing results is promising for applications that can sacrifice some accuracy for speed[5]. Another important issue is the granularity of the information represented. The LFG-inspired dependencies omit almost half of the features of the original LFG f-structures during the conversion. Some of those features, such as morphological information can easily be incorporated into CoNLL trees when needed. But information lost on functional relations such as non-local dependencies are potentially harder to recover from dependency trees. In future work, we will explore an approach that allows us to recover such functions after parsing.

[5]The Berkeley parser (run in *accurate* mode) takes 10m25s to parse Section 23 compared to 3m17s for MaltParser on the same machine.

5 Related Work

Recent research which attempts to combine ideas from Lexical Functional Grammar and data-driven dependency parsing is described in Schluter and van Genabith (2009) as well as Øvrelid et al. (2009). Schluter and van Genabith (2009) are concerned with French parsing: they convert LFG f-structures into pseudo-projective dependencies and train both MSTParser and MaltParser on these. They conclude that the use of a dependency parser is a reasonable alternative to a c-structure parser in an LFG parsing pipeline. The approach of Øvrelid et al. (2009) is quite different: in contrast to our approach they do not produce LFG-inspired dependency trees but adapt the feature set used by MaltParser's parser-action classifier so that it includes features obtained from the hand-crafted ParGram English and German LFGs to improve parser output in the format of CoNLL-style dependency trees.

6 Concluding Remarks

LFG f-structures are close to but not the same as the bilexical dependencies used in dependency parsers and we have shown how f-structures can be systematically converted into bilexical dependencies for use in direct parsing into f-structure, obviating the c-structure component in classical LFG parsing. This makes theoretically motivated abstract LFG dependency representations available to the dependency parsing community and fast dependency parsing technology available to the LFG community. This does not come without a price however. A number of pragmatic decisions needed to be made in order to allow existing dependency parsers to be trained, and these decisions include deciding what information can be lost in moving from graphs to trees. How important the lost information is to downstream applications remains an open question. We plan to investigate this, to compare our LFG-inspired dependency scheme to other dependency schemes such as the Stanford dependencies (de Marneffe and Manning, 2008), the Pennconverter dependencies (Johansson and Nugues, 2007) and Tésniére-inspired dependencies (Sangati and Mazza, 2009), and to experiment with dependency parsers which can handle multiple-headed constructions (Sagae and Tsujii, 2008).

Acknowledgements

This research is partly supported by the Science Foundation Ireland (Grant 07/CE/I1142) as part of the Centre for Next Generation Localisation (www.cngl.ie) at Dublin City University, School of Computing.

References

Attardi, Giuseppi and Massimiliano Ciaramita (2007). Tree Revision Learning for Dependency Parsing. In *Proceedings of HLT-NAACL-07*. Rochester, NY.

Bresnan, Joan (2001). *Lexical-Functional Syntax*. Oxford: Blackwell Publishers.

Cahill, Aoife, Michael Burke, Ruth O'Donovan, Stefan Riezler, Josef van Genabith and Andy Way (2008). Wide-Coverage Deep Statistical Parsing using Automatic Dependency Structure Annotation. *Computational Linguistics* 34(1).

Cahill, Aoife, Michael Burke, Ruth O'Donovan, Josef van Genabith and Andy Way (2004). Long-Distance Dependency Resolution in Automatically Acquired Wide-Coverage PCFG-Based LFG Approximations. In *Proceedings of ACL-04*.

Charniak, Eugene (2000). A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL-00*. Seattle.

Charniak, Eugene and Mark Johnson (2005). Course-to-fine n-best-parsing and MaxEnt discriminative reranking. In *Proceedings of ACL-05*. Ann Arbor.

Charniak, Eugene, Mark Johnson et al. (2006). Multilevel Coarse-to-fine PCFG Parsing. In *Proceedings of HLT-NAACL-06*. New York.

Chrupała, Grzegorz, Nicolas Stroppa, Josef van Genabith and Georgiana Dinu (2007). Better training for function labeling. In *RANLP 2007*. Bulgaria.

de Marneffe, Marie-Catherine and Christopher D. Manning (2008). The Stanford typed dependencies representation. In *Proceedings of the COLING Workshop on Cross-Framework and Cross-Domain Parser Evaluation*. Manchester.

Eisner, Jason (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING-96*. Denmark.

Giménez, Jesús and Lluís Màrquez (2004). SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of LREC-04*. Lisbon.

Guo, Yuqing, Haifeng Wang and Josef van Genabith (2008). Accurate and robust LFG-based generation for Chinese. In *Proceedings of INLG-08*. Salt Fork.

Johansson, Richard and Pierre Nugues (2007). Extended Constituent-to-Dependency Conversion for English. In *Proceedings of NODALIDA-07*.

Kaplan, Ron, Stefan Riezler, Tracy King, John Maxwell, Alexander Vasserman and Richard Crouch (2004). Speed and Accuracy in Shallow and Deep Stochastic Parsing. In *Proceedings HLT-NAACL-04*. Boston, MA.

Matsuzaki, Takuya, Yusuke Miyao and Jun'ichi Tsujii (2005). Probabilistic CFG with latent annotations. In *Proceedings of ACL-05*. Ann Arbor.

Maxwell, John and Ron Kaplan (1996). An Efficient Parser for LFG. In *Proceedings of LFG-96*. Grenoble.

McDonald, Ryan (2006). Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing. Ph.D. thesis, University of Pennsylvania.

McDonald, Ryan, Koby Crammer and Fernando Pereira (2005). Online Large-Margin Training of Dependency Parsers. In *Proceedings of ACL-05*. Ann Arbor.

McDonald, Ryan and Fernando Pereira (2006). Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of EACL-06*.

Nivre, Joakim (2009). Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of ACL-AFNLP-09*. Singapore.

Nivre, Joakim, Johan Hall, Sandra Kübler, Ryan Mac Donald, Jens Nilsson, Sebastian Riedel and Deniz Yuret (2007). The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of EMNLP-CoNLL-07*. Prague.

Nivre, Joakim, Johan Hall and Jens Nilsson (2006). MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In *Proceedings LREC-06*.

Nivre, Joakim and Jens Nilsson (2005). Pseudo-Projective Dependency Parsing. In *Proceedings of ACL-05*. Ann Arbor.

Oflazer, Kemal, Bilge Say, Zeynep Hakkani-Tür and Gökhan Tür (2003). Building a Turkish treebank. In A. Abeillé (ed.), *Treebanks: Building and Using Parsed Corpora*, Amsterdam:Kluwer.

Øvrelid, Lilja, Jonas Kuhn and Kathrin Spreyer (2009). Improving data-driven dependency parsing using large-scale LFG grammars. In *Proceedings of ACL-AFNLP-09*. Singapore.

Petrov, Slav, Leon Barrett, Romain Thibaux and Dan Klein (2006). Learning Accurate, Compact and Interpretable Tree Annotation. In *Proceedings of ACL-06*.

Petrov, Slav and Dan Klein (2007). Improved Inference for Unlexicalized Parsing. In *Proceedings of HLT-NAACL-07*. Rochester, NY.

Roark, Brian and Kristy Hollingshead (2008). Classifying chart cells for quadratic complexity context-free inference. In *Proceedings of COLING-08*. Manchester.

Sagae, Kenji and Jun'ichi Tsujii (2008). Shift-Reduce Dependency DAG Parsing. In *Proceedings of COLING-08*. Manchester.

Sangati, Federico and Chiara Mazza (2009). An English Dependency Treebank á la Tesniére. In *Proceedings of TLT8*. Milan.

Schluter, Natalie and Josef van Genabith (2009). Dependency Parsing Resources for French: Converting Acquired Lexical Functional Grammar F-Structure Annotations and Parsing F-Structures Directly. In *Proceedings of NODALIDA 2009*. Odense.

Yamada, Hiroyasu and Yuji Matsumoto (2003). Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of IWPT-03*. France.