Automated Deep Lexical Acquisition for Robust Open Texts Processing

Yi Zhang, Valia Kordoni

Department of Computational Linguistics
Saarland University
D-66041 Saarbrücken, Germany
{yzhang, kordoni}@coli.uni-sb.de

Abstract

In this paper, we report on methods to detect and repair lexical errors for deep grammars. The lack of coverage has for long been the major problem for deep processing. The existence of various errors in the hand-crafted large grammars prevents their usage in real applications. The manual detection and repair of errors requires a significant amount of human effort. An experiment with the British National Corpus shows about 70% of the sentences contain unknown word(s) for the English Resource Grammar (ERG; (Copestake and Flickinger, 2000)). With the help of error mining methods, many lexical errors are discovered, which cause a large part of the parsing failures. Moreover, with a lexical type predictor based on a maximum entropy model, new lexical entries are automatically generated. The contribution of various features for the model are evaluated. With the disambiguated full parsing results, the precision of the predictor is enhanced significantly.

1. Background

Deep linguistic processing delivers fine-grained syntactic and semantic analyses which are difficult to achieve with shallow methods. The core part of deep processing is a complex rule system, called the *deep grammar*. Linguistic data is processed by recursively applying the grammar rules. Although there are still doubts on how such analyses can be used, we do see a strong call for deep analyses from advanced NLP applications like question answering, machine translation, etc.

Traditionally, deep processing is considered to be resource consuming. This is true given that with detailed linguistic features the search space is much larger. However, with various efficient parsing algorithms (Callmeier, 2000) and advanced modern hardware, *efficiency* is no longer the number one practical issue.

The main problem with deep processing is that the number of outputs from deep processing is hard to control. The *specificity* problem arises when there are more analyses generated than expected. The analyses might be linguistically sound, but are practically uninteresting for real applications. On the other hand, the *robustness* problem arises when fewer or no results are delivered, probably due to the insufficient coverage of the grammar.

From the *grammar engineering* point of view, *specificity* and *robustness* are a pair of dual problems. Gain on the one side means potential loss on the other. A balance point should be achieved in order to maximize the amount of linguistic accuracy retained by the grammar. But for specific applications, preference of one over the other can be achieved by extra mechanism.

In this paper, we tackle the robustness problem for wide coverage open texts processing with deep grammars. Previous evaluation (Baldwin et al., 2004) suggested that the majority of parsing failures with large grammars are caused by lexicon missing. The coverage of construction is normally satisfying (except for some idiosyncratic constructions). We will focus on exploring how to automatically detect and create missing lexical entries. By assuming a complete set of atomic lexical types defined by the grammar, our methods work as external modules. No direct modification to the grammar is necessary, so that the linguistic elegance of the grammar is preserved and minimal attention from the grammar developer is required. By incorporating the techniques like error mining, parse disambiguation, together with the maximum entropy models, the robustness of deep processing is significantly enhanced. Also the methods are largely language and formalism independent.

The remainder of the paper is organized as follow. Section 2. reviews previous proposals on grammar error detection and lexical acquisition. Section 3. describes the techniques used for detecting missing lexical entries and related experiments on an English HPSG grammar (ERG; (Copestake and Flickinger, 2000)) with BNC. Section 4. reports on a statistical approach towards automated lexical type prediction. Finally, section 5. concludes the paper.

2. Previous Work

Hand-crafted large grammars are error-prone. An error can be roughly classified as *under-generating* (if it prevents a grammatical sentence to be generated/parsed) or *over-generating* (if it allows an ungrammatical sentence to be generated/parsed). In the context of wide-coverage parsing, we focus on the *under-generating* errors which normally lead to parsing failure.

Traditionally, the errors of the grammar are detected manually by the grammar developers. This is usually done by running the grammar over a carefully designed test suite and inspecting the outputs. This procedure becomes less reliable as the grammar gets larger, and is especially difficult when the grammar is developed in a distributed manner.

In recent years, some approaches have been developed to (semi)automatically detect and/or repair the errors in linguistic grammars. Such analyses mainly take two directions: the symbolic approach or the statistical approach.

(Cussens and Pulman, 2000) describe a symbolic approach towards grammar extension with *inductive logic programming*. The basic idea is that, after a failed parse, abduction is used to find the *needed edges*, which, if existing, would allow a complete parse of the sentence. Various heuristics

are applied to constrain the generation of new rules from these *needed edges* to filter out the linguistically implausible cases. Further, generalization operators are used in order to cope with the problem that generated rules are too specific. This approach focuses on the missing construction rules. However, the generated new rules might still be either too specific or too general. The experiments are carried out on a very small grammar and a few sentences that require the missing rules. The extendability is doubtful for large grammars.

(van Noord, 2004) follows a statistical approach towards semi-automated error detection using the parsability metric for word sequences. The techniques are simple and of particular interest to large grammars. The details are described in Section 3.1.

Detecting the error is only half of the story. The automatic repair of the errors is even more difficult. As we mentioned before (and will show in details in Section 3.2.), the majority of errors found in state-of-the-art deep grammars are the incompleteness of the lexicon. To automatically extend the lexicon of a grammar, various approaches have been reported.

(Erbach, 1990; Barg and Walther, 1998; Fouvry, 2003) have followed a unification-based approach towards unknown word processing for constraint-based grammars. The basic idea is to use the underspecified lexical entries, namely the entries with less constraints, to parse the whole sentences. And the lexical entries are generated afterwards by the collected information from the full parse. However, lexical entries generated in this way might be both too general and too specific. And underspecified lexical entries with fewer constraints allow more grammar rules to be applied while parsing. Fully underspecified entries are computationally intractable. It gets even worse when two unknown words occur next to each other, which might allow almost any constituent to be constructed.

(Baldwin, 2005) takes a statistical approach towards automated lexical acquisition for deep grammars. Focused on generalizing the method of deriving deep lexical acquisition models on various secondary language resources, (Baldwin, 2005) uses a large set of binary classifiers to predict whether a given unknown word is of a particular lexical type. This data-driven approach is grammar independent and can be scaled up for large grammars. A potential problem is that the binary classifiers are not related to each other. If all the classifiers give negative outputs for a given word, no lexical entry will be generated. In the context of robust processing, an *inaccurate* result will be preferable over *NO* result.

Aiming at robust processing of open texts, we need to automate both the error detection, as well as the lexical acquisition processes.

3. Error Detecting

3.1. Error Mining

(van Noord, 2004) reports on a simple yet practical way of identifying grammar errors. The method is particularly useful for discovering systematic problems in a large grammar with reasonable coverage. The idea behind it is that each (under-generating) error in the grammar leads to the parsing failure of some specific grammatical sentences. By running the grammar over a large corpus, the corpus can be split into two subsets: the set of sentences covered by the grammar and the set of sentences failed to parse. The errors can be identified by comparing the *statistical difference* between these two sets of sentences.

By *statistical difference*, any kind of uneven distribution of linguistic phenomena is meant. In (van Noord, 2004), the word sequences are used. This is mainly because the cost to compute and count the word sequences is minimum. Specifically, the parsability of a sequence $w_i \dots w_j$ is defined as:

$$R(w_i \dots w_j) = \frac{C(w_i \dots w_j, OK)}{C(w_i \dots w_j)} \quad (1)$$

where $C(w_i \dots w_j)$ is the number of sentences in which the sequence $w_i \dots w_j$ occurs, and $C(w_i \dots w_j, OK)$ is the number of sentences with a successful parse which contain the sequence.

A frequency cut is used to eliminate the infrequent sequences. With suffix arrays and perfect hashing automata, the parsability of all word sequences (with arbitrary length) can be computed efficiently. The word sequences are then sorted according to their parsabilities. Those sequences with the lowest parsabilities are taken as direct indication of the grammar errors.

3.2. Experiment with ERG on BNC

In (van Noord, 2004), various errors have been discovered for the Dutch Alpino Grammar (Bouma et al., 2001) using the Twente Nieuws Corpus. The errors are mainly introduced by the tokenizer, the error/incomplete lexical descriptions, the frozen expressions with idiosyncratic syntax, or the incomplete grammatical descriptions. However, no concrete data is given about the distribution of different types of errors discovered.

For the purpose of robust open texts processing, we are interested in seeing what the major type of error for a typical large scale deep grammar is. Based on this reason, we have run the error mining experiment with the English Resource Grammar (ERG; (Copestake and Flickinger, 2000))[1] and the British National Corpus 2.0 (BNC; (Lou Burnard, 2000)).

We used a subset of the BNC written component. The sentences in this collection contain no more than 20 words and only ASCII character. That is about 1.8M distinct sentences.

These sentences are then fed into an efficient HPSG parser (PET; (Callmeier, 2000)) with ERG loaded. The parser has been configured with a maximum edge number limit of 100K and it is running in the *best-only* mode so that it does not exhaustively find all the possible parses. The result of each sentence is marked as one of the following four cases:

- P means at least one parse is found for the sentence;
- L means the parser halted after the morphological analysis and was not able to construct any lexical item for the input token;

[1] ERG is a large-scale HPSG grammar for English. In this paper, we have used the June 2004 release of the grammar.

- N means the parser exhausted the searching and not able to parse the sentence;
- E means the parser reached the maximum edge number limit and was still not able to find a parse.

Running over the entire collection of sentences has only taken less than 2 days with a 64bit machine with 2GHz CPU. The results are shown in Table 1.

Result	# Sentences	Percentage
P	301,503	16.74%
L	1,260,404	69.97%
N	239,272	13.28%
E	96	0.01%

Table 1: Distribution of Parsing Results

From the results shown in Table 1, one can see that ERG has full lexical span for only a small portion (about 30%) of the sentences. For these sentences, about 56% are successfully parsed. These numbers are very similar to the results reported in (Baldwin et al., 2004).
Obviously, L indicates the unknown words in the input sentence. But for N and E, it is not clear where and what kind of error has occurred. In order to pinpoint the errors, we used the error mining techniques on the grammar and corpus. We have taken the sentences marked as N or E (because the errors in L sentences are already determined) and calculate the word sequence parsabilities against the sentences marked as P. The frequency cut is set to be 5. The whole process has taken no more than 20 minutes, resulting in parsability scores for 74,500 n-grams (word sequences). The distribution of n-grams in length with parsability below 0.1 is shown in Table 2.

	Number	Percentage
uni-gram	2,336	10.52%
bi-gram	15,183	68.36%
tri-gram	4,349	19.58%

Table 2: Distribution of N-gram in Length in Error Mining Results ($R(x) < 0.1$)

Although pinpointing the problematic n-grams still does not tell us what the exact errors are, it does shed some light on the cause. From Table 2 we see quite a lot of uni-grams with low parsabilities. Table 3 gives some examples of the word sequences. By intuition, we make the bold assumption that the low parsability of uni-grams is cause by the missing of appropriate lexical entries for the corresponding word.[2] Another heuristic we have used in order to detect missing lexical entries is based on the bi-grams starting with a determiner. For example, *"the poor"* received low parsability because the nominal reading of *"poor"* is missing in the grammar's lexicon. There are also other cases of errors in the lexicon. For instance, the verb *"peer"* should take an appropriate lexical type (e.g., *"v_empty_prep_intrans_le"* in the ERG) which will allow its combination with the preposition *"at"*; the *"World Cup"* should be treated as a multi-word expression, etc.

N-gram	Count
weed	59
the poor	49
a fight	113
in connection	85
as always	84
peered at	28
the World Cup	57

Table 3: Some Examples of the N-grams in Error Mining Results

[2]It has been later confirmed with the grammar developer that almost all of the errors detected by these low parsability uni-grams can be fixed by adding correct lexical entries.

4. Automated Deep Lexical Acquisition

In the previous section, we have seen that about 70% of the sentences contain one or more unknown words. And about half of the other parsing failures are also due to lexicon missing. In this section, we propose a statistical approach towards lexical type prediction for unknown words.

4.1. Atomic Lexical Types

Lexicalist grammars are normally composed of a limited number of rules and a lexicon with rich linguistic features attached to each entry. Some grammar formalisms have a type inheriting system to encode various constraints, and a flat structure of the lexicon with each entry mapped onto one type in the inheritance hierarchy. The following discussion is based on the *Head-driven Phrase Structure Grammar (HPSG)* (Pollard and Sag, 1994), but should be easily adapted to other formalisms.
The formalism of HPSG is based on the *typed feature structure (TFS)* (Carpenter, 1992). In HPSG, all the linguistic objects are modeled by TFSs. Formally, a TFS is a *directed acyclic graph (DAG)*. Each node in DAG is labelled with a *sort symbol* (or *type*) corresponding to the category of the linguistic object. All the *sort symbols* are organized into an inheritance system, namely the *type hierarchy*. Two types are compatible, if they share at least one common subtype in the hierarchy.
The lexicon of HPSG consists of a list of well-formed TFSs, which convey the constraints on specific words by two ways: the type compatibility, and the feature-value consistency. Although it is possible to use both features and types to convey the constraints on lexical entries, large grammars prefer the use of types in the lexicon because the inheritance system prevents the redundant definition of feature-values. And the feature-value constraints in the lexicon can be avoided by extending the types. Say we have n lexical entries $L_i : \underset{t}{\left[\mathrm{F} \quad a_1\right]} \ldots \; L_n : \underset{t}{\left[\mathrm{F} \quad a_n\right]}$. They share the same lexical type t but take different values for the feature F. If $a_1, \ldots, a_n$ are the only possible values for F in the context of type t, we can extend the type t with subtypes $t_{a1} : \underset{t}{\left[\mathrm{F} \quad a_1\right]} \ldots \; t_{an} : \underset{t}{\left[\mathrm{F} \quad a_n\right]}$ and modify the lexical entries to use these new types, respectively. Based on the fact that large grammars normally have a very restricted

number of feature-values constraints for each lexical type, the increase of the types is acceptable. Also, it is typical that the types assigned to lexical entries are maximum on the type hierarchy, which means that they have no further subtypes. We will call the maximum lexical types after extension the *atomic lexical types*. Then the lexicon will be an one-to-one mapping from the word stems to the atomic lexical types.

4.2. Statistical Lexical Type Predictor

Given that the lexicon of deep grammars can be modelled by a mapping from word stems to atomic lexical types, we now go on designing the statistical methods that can automatically "guess" such mappings for unknown words.

Similar to (Baldwin, 2005), we also treat the problem as a classification task. But there is an important difference. While (Baldwin, 2005) makes predictions for each unknown word, we create a new lexical entry for each occurrence of the unknown word. The assumption behind is that there should be exactly one lexical entry that corresponds to the occurrence of the word in the given context[3].

We use a single classifier to predict the atomic lexical type. There are normally hundreds of atomic lexical types for a large grammar. So the classification model should be able to handle a large number of output classes. We choose the Maximum Entropy based model because it can easily handle thousands of features and a large number of possible outputs. It also has the advantages of general feature representation and no independence assumption between features. With the efficient parameter estimation algorithms discussed in (Malouf, 2002), the training of the model is now very fast.

For our prediction model, the probability of a lexical type t given an unknown word and its context c is:

$$p(t|c) = \frac{exp(\sum_i \theta_i f_i(t,c))}{\sum_{t' \in T} exp(\sum_i \theta_i f_i(t',c))} \qquad (2)$$

where feature $f_i(t,c)$ may encode arbitrary characteristics of the context. The parameters $< \theta_1, \theta_2, \ldots >$ can be evaluated by maximizing the pseudo-likelihood on a training corpus (Malouf, 2002).

4.3. Feature Selection

The basic feature templates used in our ME-based model include the prefix and suffix of the unknown word, the context words within a window size of 5 and their corresponding lexical types.

Considering that deep lexical types normally encode complicated constraints that only make sense when they work together with the grammar rules, it is presumable that the syntactic features of the sentence will have a good contribution to the prediction model. However, with the unknown words in place, full analysis of the sentence cannot be generated. So we modify our strategy by inserting a partial parsing stage before the lexical type predictor, if there are unknown words on the input sequence.

[3]Lexical ambiguity is not considered here for the unknowns. In principle, this constraint can be relaxed by allowing the classifier to return more than one results by, for example, setting a confidence threshold.

The partial parse needs some clarification. A full parse can be represented by a set of edges as shown in Figure 1(a). Each edge is derived from a rule application. There is no more than one edge between each pair of positions. And there is always exactly one full span edge in a full parse.

A partial parse of an input sequence is a set of edges which comprises the shortest path from the beginning to the end of the sequence[4]. There might be more than one partial parse for a given input sequence. As shown in Figure 1(b), when the word between position 2 and 3 is unknown, a dummy edge c is created. This dummy edge will prevent further rule application. Both $a - c - d$ and $b - c - d$ are partial parses.

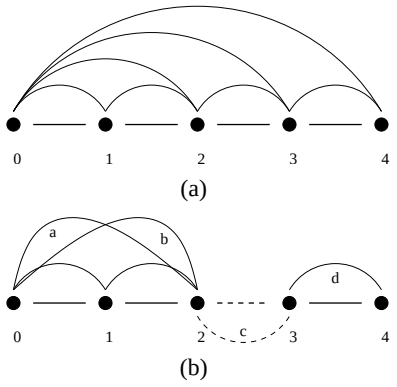

Figure 1: Parsing edges: (a) edges in a full parse; (b) edges in partial parses.

From the partial parses, we collect all edges that are adjacent to the left/right of the unknown word, respectively. Then the rules that generate these edges are counted according to their application (once per edge). The most frequently used rules to create left/right adjacent edges are added as two features conveying syntactic information into the ME-based model. A complete list of features templates used in our lexical type predictor is shown in Table 4.

In general, our lexical type predictor is a tagger that assigns each occurrence of unknown an atomic lexical type. The differences to the widely used POS taggers are:

- A POS tagger normally assigns tags from a small POS tagset (typically less than 100 different tags), while our predictor assigns from a much larger set of atomic lexical types (for ERG, the number is around 800).
- A POS tagger normally assigns a sequence of tags to the input token sequence, while our predictor only tags the unknown word.
- A POS tagger normally works as a preprocessing module, so it only uses the surface features. But our predictor also takes the outputs from other processing modules (lexical types of context words, syntactic contexts, etc.) into consideration.

[4]Note that the edges on the full parse of the sentence are not necessary in the corresponding partial parses, if a word is assumed to be unknown. However, partial parses do reduce the number of candidate edges for consideration.

Features
X is prefix of w_i, $\|X\| \leq 4$
X is suffix of w_i, $\|X\| \leq 4$
$t_{i-1} = X, t_{i-2}t_{i-1} = XY, t_{i+1} = X, t_{i+1}t_{i+2} = XY$
$w_{i-2} = X, w_{i-1} = X, w_{i+1} = X, w_{i+2} = X$
LP is the left adjacent most frequent edge of w_i
RP is the right adjacent most frequent edge of w_i

Table 4: Feature templates used in ME-based prediction model for word w_i (t_j is the lexical type of w_j)

To evaluate the contribution of various features and the overall precision of the unknown word prediction model, we have done a 10-fold cross validation on the *Redwoods Treebank* (Oepen et al., 2002) [5]. For each fold, words that do not occur in the training partition are assumed to be unknown and are temporarily removed from the lexicon.

For comparison, we have also built a baseline system that always assigns a majority type to each unknown according to the POS tag. Specifically, we tag the input sentence with a small Penn Treebank-like POS tagset. It is then mapped to a most popular lexical type for that POS.[6] Table 5 lists part of the mappings.

POS	Majority Lexical Type
noun	n_intr_le
verb	v_np_trans_le
adj.	adj_intrans_le
adv.	adv_int_vp_le

Table 5: Part of the POS tags to lexical types mapping

Again for comparison, we have built another two simple prediction models with two popular general-purpose POS taggers, *TnT*(Brants, 2000) and *MXPOST*(Ratnaparkhi, 1996). *TnT* is a HMM-based trigram tagger while *MXPOST* is a maximum entropy based one. We have trained the tagging models with all the leaf lexical types as the tagset. The taggers tag the whole sentence. But only the output tags for the unknowns are taken to generate the lexical entries.

The maximum entropy based model is tested both with and without using partial parsing results as features. The precision of different prediction models are shown in Table 6.

The baseline model achieves precision around 30%. This means that the task of unknown word type prediction for deep grammars is non-trivial. The general-purpose POS taggers based models perform quite well, outperforming the baseline by 10%. As a confirmation to (Elworthy, 1995)'s claim, a larger tagset does not necessarily imply that tagging will be more difficult[7]. Our ME-based model

Model	Precision
Baseline	30.7%
TnT	40.4%
MXPOST	40.2%
ME(-pp)	50.0%
ME(+pp)	50.5%

Table 6: Precision of Unknown Word Type Predictors (+/-pp means w or w/o partial parsing result features)

significantly outperforms the POS tagger-based models by another 10%, as foreseen by the differences between our prediction model and POS taggers.

By incorporating simple syntactic information into the ME-based model, we get extra precision gain of less than 1%. Also, by applying partial parsing, the computation complexity increases significantly in comparison to our basic ME-based model.

4.4. Enhancing the Performance

The experiment result shows that the incorporation of partial parsing results does not enhance the precision much. This is mainly because the partial parses introduce many extra edges that are not on the full parse. Especially, when the unknown word occurs on the head path of the sentence, the entire derivation tree falls apart completely.

An alternative way to use the syntactic information is to help the parser to generate full parses in the first place and let the parsing result tell which lexical entry is good.

In order to help the parser generate a full parse of the sentence, we feed the newly generated lexical entries directly into the parser. Instead of generating only one entry for each occurrence of unknown, we pass on top n most likely lexical entries. With these new entries, the sentence will receive one or more parses (assuming the sentence is grammatical and covered by the grammar). From the parsing results, a best parse is selected with the disambiguation model and the corresponding lexical entry is taken as the final result of lexical extension. Within this processing model the incorrect types will be ruled out, if they are not compatible with the syntactic context. Also the infrequent readings of the unknown will be dispreferred by the disambiguation model.

Still using the same evaluation methods, we generate and pass on the top 3 most likely lexical entries for each oc-

[5] *Redwoods* is a HPSG treebank that records the full analyses of the sentences with *ERG*. The genre of texts includes email correspondence, travel planning dialogs, etc. The 5th growth of *Redwoods* we have used contains about 16.5K sentences and 122K token (not including sentences without a full analysis).

[6] This is similar to the built-in unknown word handling mechanism of the *PET* system.

[7] The learning curves of the taggers are analyzed with respect to the size of the tagsets and the training data. The taggers with large tagsets get lower precision at the beginning. But with the growth of training data, they catch up very soon.

currence of unknown to the parser. The results are shown in Table 7 (with some numbers from Table 6 repeated for comparison).

Model	Precision
ME(-pp)	50.0%
ME(-pp)+ disambi. result	61.3%

Table 7: Precision of Unknown Word Type Predictors w/wo Disambiguated Parsing Results

By incorporating the disambiguation results, the precision of the model boosts up for another 10%. The computational overhead is proportional to the number of candidate entries added for each unknown word. However, in most cases, introducing lexical entries with incorrect types will end up to parsing failure and can be efficiently detected by quick checking. In such cases the slowdown is acceptable.

5. Conclusion

In this paper, we have tackled the robustness problem of deep processing from two aspects. The error mining techniques have been used to semi-automatically detect errors in deep grammars. And a statistical lexical type predictor has been designed in order to automatically repair errors in the lexicon. The error analysis clearly indicates that the lexical coverage is the major barrier for wide-coverage open texts processing. With some heuristics, many missing lexical entries can be automatically detected. With the maximum entropy model based lexical type predictor, the new lexical entries can be generated on the fly. Experiments on the ERG with the Redwoods Treebank shows, by incorporating parse disambiguation results, the unknown word type predictor achieves precision over 60%.

Although the experiments are carried out with the ERG, the underlying model is general enough to be easily applied to other constraint-based lexicalist grammars, provided the lexical categories can be abstracted by a set of atomic types.

6. References

Timothy Baldwin, Emily M. Bender, Dan Flickinger, Ara Kim, and Stephan Oepen. 2004. Road-testing the English Resource Grammar over the British National Corpus. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal.

Timothy Baldwin. 2005. Bootstrapping deep lexical resources: Resources for courses. In *Proceedings of the ACL-SIGLEX Workshop on Deep Lexical Acquisition*, pages 67–76, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Petra Barg and Markus Walther. 1998. Processing unkonwn words in HPSG. In *Proceedings of the 36th Conference of the ACL and the 17th International Conference on Computational Linguistics*, Montreal, Quebec, Canada.

Gosse Bouma, Gertjan van Noord, and Robert Malouf. 2001. Alpino: Wide-coverage computational analysis of dutch. In *Computational Linguistics in The Netherlands 2000*.

Thorsten Brants. 2000. TnT - a statistical part-of-speech tagger. In *Proceedings of the Sixth Confrence on Applied Natural Language Processing ANLP-2000*, Seattle, WA, USA.

Ulrich Callmeier. 2000. PET – a platform for experimentation with efficient HPSG processing techniques. *Journal of Natural Language Engineering*, 6(1):99–108.

Bob Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, England.

Ann Copestake and Dan Flickinger. 2000. An open-source grammar development environment and broad-coverage english grammar using hpsg. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000)*, Athens, Greece.

James Cussens and Stephen Pulman. 2000. Incorporating Linguistics Constraints into Inductive Logic Programming. In *Fourth Conference on Computational Natural Language Learning and of the Second Learning Language in Logic Workshop*.

David Elworthy. 1995. Tagset design and inflected languages. In *EACL SIGDAT workshop "From Texts to Tags: Issues in Multilingual Language Analysis"*, pages 1–10, Dublin, Ireland, April.

Gregor Erbach. 1990. Syntactic processing of unknown words. IWBS Report 131, IBM, Stuttgart.

Frederik Fouvry. 2003. Lexicon acquisition with a large-coverage unification-based grammar. In *Companion to the 10th of EACL*, pages 87–90, ACL, Budapest, Hungary.

Lou Burnard. 2000. User Reference Guide for the British National Corpus. Technical report, Oxford University Computing Services.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conferencde on Natural Language Learning (CoNLL-2002)*, pages 49–55.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods treebank: Motivation and preliminary applications. In *Proceedings of COLING 2002: The 17th International Conference on Computational Linguistics: Project Notes*, Taipei.

Carl J. Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, Illinois.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, Somerset, New Jersey.

Gertjan van Noord. 2004. Error mining for wide-coverage grammar engineering. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 446–453, Barcelona, Spain, July.