An On-line Learning Method to Speed-up Natural Language Processing

Günter Neumann
DFKI GmbH
Stuhlsatzenhausweg 3
66123 Saarbrücken, Germany
neumann@dfki.uni-sb.de

Abstract

We present an *on-line* method of automatic subgrammar extraction based on Explanation-based Learning, in which the source grammar processor and the subgrammar processor are strictly interleaved allowing for sub-sentential interactions. Informally, the new method can be considered as an intelligent storage unit of example-based generalized parts of the grammatical search space determined through training by the source grammar processor. Processing of similar new input is then reduced to simple but efficient lookup and matching operations, which circumvent re-computation of this already known search space. The method is *uniform* in the sense that it can be used for both, parsing and generation. It has fully been implemented and tested with broad-coverage grammars for English, German, and Japanese.

1 Introduction

Recently, one has been able to observe divergent research activities with respect to the representation and processing of natural language (NL) grammars. On the one hand, advanced grammatical frameworks have been developed which allow the representation of grammatical knowledge in a very compact and principled manner making use of type and constraint information in a systematic way. In particular, the sign-based approach employed by Head-driven Phrase Structure Grammar (HPSG, [Pollard and Sag, 1994]) has made important contributions not only linguistically, but also for the development of deep NL-systems. On the other hand, it has been experienced that parsing and generation with such high-level grammars is extremely complex, in particular for real-world applications, e.g., speech processing or message extraction systems. Here, statistical corpus-based methods and finite-state grammars as a means for shallow NL-processing have been demonstrated to be quite more successful in achieving fast and robust behavior. However, with the increasing popularity of such shallow strategies issues like easy portability, re-usability and multi-linguality have recently become highly relevant [Grishman and Sundheim, 1996].

A promising direction towards greater flexibility and a means for combining deep and shallow strategies is the automatic extraction of domain-specific subgrammars from a given source grammar, which can be processed very efficiently by a specialized parser, e.g., [Samuelsson and Rayner, 1991; Briscoe and Carroll, 1993; Neumann, 1994; Samuelsson, 1994; Srinivas and Joshi, 1995; Rayner and Carter, 1996].

A major disadvantage of current approaches is that they are *off-line*, i.e., they require that the whole training corpus has already been parsed by some general parser using the source grammar before the subgrammar extraction process can start. Because the training phase can be extremely time-consuming—especially when the source grammar is processed in a constraint-based grammar formalism like HPSG—there is a need to speed-up the training phase, too. Otherwise, it will be very hard to take advantage of the important new results obtained in the areas of grammar formalism and theory for real-world applications. Furthermore, speeding-up parsing through on-line learning also improves the development cycle of the source grammar, since linguists are able to test their grammar coverage on some test corpus more effectively.

As a move towards faster training, we present a novel *on-line* method of automatic subgrammar extraction, in which the general parser and the subgrammar parser are strictly interleaved allowing for sub-sentential interactions. The new on-line method is based on Explanation-Based Learning (EBL)[1], and informally performs as follows: for a parsed sentence, the derivation tree and all its relevant subtrees are extracted, generalized, and efficiently indexed using the generalized terminal yield

[1] See [Minton *et al.*, 1989] for an overview and introduction to EBL.

of the derivation tree. Then, during parsing of a new sentence the subgrammar parser's job is to find and instantiate previously extracted generalized derivation trees (called templates) and to pass them to the general parser. Consequently, the general parser can avoid costly re-computation of already observed (partial) solutions. The main properties of the new on-line method are:

- during parsing of the training corpus, the subgrammar is incrementally extracted;
- the processing of the subgrammar is vastly sped up using simple but efficient mechanisms;
- type-driven generalization and indexing allows for a very compact representation of the subgrammar;
- the general and the subgrammar parser interact on arbitrary phrasal level;
- the new method is *uniform*, in the sense that in can be used for both, parsing and generation.

This means that the whole power of grammatical processing is only needed for yet completely unobserved structures, and avoids costly re-computation of already known generalized constructions.

The new approach has been fully implemented and tested on broad-coverage HPSG-based grammars for English, German, and Japanese using an advanced typed constraint-based grammar formalism. Since generalization and indexing of parse trees is strictly type-driven, adaption to new grammars (not necessarily HPSG-based) is realized through a simple type-based interface. In fact, the on-line learning method has a high-degree of modularity which is comparable to that of standard parsers. Additionally, a dynamic memory management strategy is embedded for controlling the size of a subgrammar (by performing a kind of garbage collection of un-used templates, see section 6 for more details).

2 Prerequisites

In this paper, the focus is on grammatical processing, i.e., the mapping of strings to structures (usually representing semantic information) in the case of parsing, and vice versa in the case of generation. Furthermore, the rest of the paper will explain the new approach from the parsing perspective. This is no essential restriction, since parsing and generation share the same learning strategy. In section 6 we shortly outline how the same method is used during natural language generation.

Informally, EBL can be considered as an intelligent storage unit of example-based generalized parts of the grammatical search space determined through training by the grammatical processor. Processing of similar new input is then reduced to simple lookup and matching operations, which *circumvent* re-computation of this already known search space.

We concentrate on constraint-based grammar formalisms following a sign-based approach considering linguistic objects (i.e., words and phrases) as utterance-meaning associations [Pollard and Sag, 1994]. Thus viewed, a grammar is a formal statement of the relation between utterances of a natural language and representations of their meanings in some logical or other artificial language, where such representations are usually called *logical forms*. The basic task of the parser or generator is then to unfold this relationship for a given input (a string or logical form). The result is a feature structure (or a set of such structures in case of alternatives) containing among others the string, the logical form, and a representation of the derivation.

In our current implementation we are using TDL, a typed feature-based language and inference system for constraint-based grammars [Krieger and Schäfer, 1994]. TDL allows the user to define hierarchically-ordered types consisting of type and feature constraints. As shown later, a systematic use of type information leads to a very compact representation of the extracted data and supports an elegant and efficient generalization step.

3 Overview of the new on-line learning method

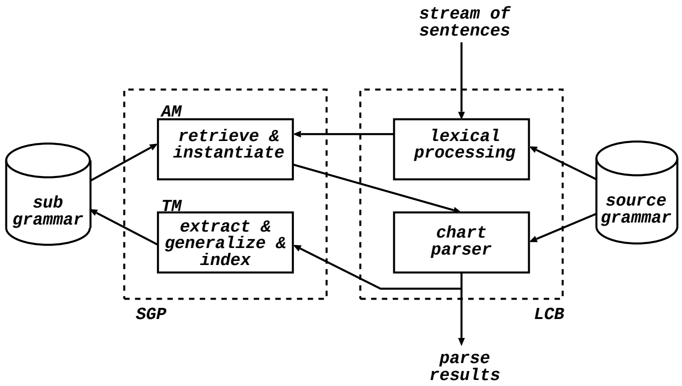


Figure 1: The overall architecture of the on-line learning method.

Figure 1 displays the overall architecture of the on-line learning method. The right part of the graphic shows the linguistic competence base (LCB) and the left one the subgrammar processing component (SGP). LCB is divided into a lexical component and a chart parser. Both have access to the source grammar. SGP is divided into a training and an application module, called TM and AM, respectively). TM performs the subgrammar extraction task, where a subgrammar consists of a set of *templates* (i.e., generalized derivation trees computed by the chart parser). Each template is stored in a *decision tree* using the (unambiguous) chain of terminal types of the generalized derivation tree. AM performs subgrammar parsing by retrieval and instantiation of templates on the basis of lexical information, and transmits them to the parser's chart.

A sentence α is processed using an AM-parse-TM strategy as follows: First, α is lexically analyzed yielding a (possibly ambiguous) lexical chain α_l. Then, AM uses the corresponding lexical type chain for traversing the decision tree using a recursive tree traversal strategy. As we will explain later in more detail, the traversal strategy allows a partial retrieval and guarantees that also overlapping templates can be accessed. Furthermore, since generally the lexical types of α_l are more specific than the type information in the decision tree (because of the generalization step), traversal is directed by type subsumption. Each successfully retrieved template dt is instantiated in two steps: first dt is used for replaying the derivation by *deterministically* applying the grammar rules in the order as indicated by dt. This yields a grammatically expanded derivation tree dt^e. Note, that α needs only to be *type-compatible* to examples observed earlier. Next, the terminal yield of dt^e is matched against that subchain of α_l which corresponds to the lexical type path used for indexing dt.

The instantiated templates together with α_l are then used for the initialization of the chart parser's agenda[2], which sorts them according to longest span. This way, the parser will prefer templates and will also be able to "backtrack" into templates by just providing next smaller ones. Since we also transmitted α_l to the agenda, it is possible to "backtrack" to lexical items. This way, completeness is not lost. However, it depends on the used priority function, what strategy the parser should follow. For example, during grammar development one might want to use an all-solution strategy, and for a real application a best-first strategy. In the latter case, it is also possible to initialize the agenda only with longest matching templates, and with elements in α_l which are not covered by a template (and even this can be ignored).

Each found parsing result α_r is then passed to the training module TM. In a first step it recursively extracts all relevant derivation subtrees where "relevance" is determined through predicates or *filters* (see below). Then, each extracted subtree dt_{α_r} is generalized using abstraction operation applied on the existing type hierarchy. Finally, the generalized tree $dt^e_{\alpha_r}$ will be inserted into the decision tree using the terminal yield of $dt^e_{\alpha_r}$ as index. However, if there already exists a tree dt_i, such that $dt_i = dt^e_{\alpha_r}$ then insertion of $dt^e_{\alpha_r}$ is blocked, and the recursive extraction process stops at the branch.

[2] Of course, if no matching template can be found, only α_l is transmitted.

4 Details illustrated by an example

The following example is used to illustrate the strategy in more detail. It is assumed that the training set consists of the two strings "Who does Kim think Sandy needed to bother?" (α) , and "Every book Kim wanted to try was very big." (β), which are processed in that order. We will use the first example to illustrate the training phase, and the second one is used to illustrate the application phase.

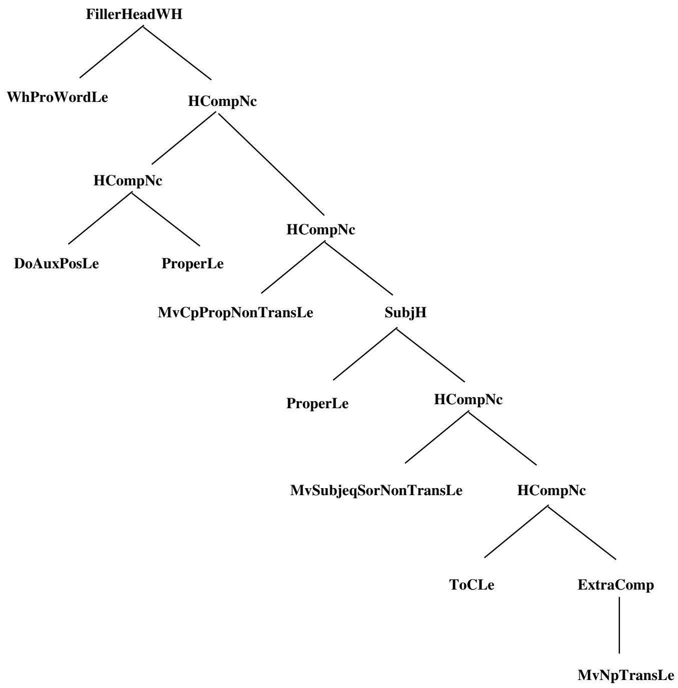


Figure 2: The generalized derivation tree dt of the parsed sentence "Who does Kim think Sandy needed to bother?"

4.1 Training phase: some more details

First, the training example α is lexically analyzed. The type chain of the resulting sequence of (possibly ambiguous) lexical elements is used by AM for retrieval in the decision tree. Since we assume that no templates have been extracted in the past, AM will find none. Therefore, the agenda of the general parser is initialized only with the found lexical elements. After grammatical processing, the resulting feature structure fs_α is passed to TM.

Generalization TM extracts and generalizes the derivation tree of fs_α. A generalized derivation tree is a tree in which non-terminal nodes denote rule names and terminal nodes denote the type of the corresponding terminal elements. Figure 2 displays the generalized derivation tree dt of our current example α.

It is possible to perform a further generalization step by selecting a *super-type* of each terminal type element instead of the given specialized type. In our current system, the super-type is obtained by just going one level higher in the type hierarchy. (Although this shows to be practical enough, the system can be parameterized wrt. more complex type abstraction mechanisms.)

An extracted and generalized derivation tree is also called a *template*. It is applicable to each new string, whose lexical type chain is *type compatible* wrt. subsumption, and which has the same derivational structure.

Phrasal templates In the next step of the training phase, we extract recursively all possible subtrees also called *phrasal templates*. Then, each phrasal template is inserted in the decision tree using its terminal yield as index. It is possible to direct the subtree extraction process through the application of *filters*, which are applied in each recursive step to the whole remaining subtree. Using these filters it is possible to restrict the range of structural properties of candidate phrasal templates (e.g., extract only saturated NPs, or subtrees having at least two daughters, or subtrees which have no immediate recursive structures). These filters serve the same means as the "chunking criteria" described in [Rayner and Carter, 1996], and are used to control the quality of the extracted subgrammar.

For each extracted subtree dt_s it is established during the extraction phase whether the decision tree already contains a path pointing to a previously extracted subtree $dt_s^{'}$, such that $dt_s = dt_s^{'}$. In that case, dt_s is not inserted and the recursion stops at that branch.

After all relevant subtrees have been indexed, the training phase stops for α.

4.2 Application phase: some more details

The next training example β is processed using the same control flow. However, since it can potentially benefit from the structures obtained for the first training example, we will describe the application phase in more detail.

Remember that the core steps during AM are retrieval and instantiation of found templates. As already mentioned, retrieval of the decision tree is performed on the basis of the type chain of the lexically analyzed input. For β this yields the type chain:

$$\langle DetSgLe, \{IntrNounWordLe, MvParticpleNp, \\ MvNpTransLe\}, ProperLe, \\ \{MvNpTransLe, MvSubjeqSorNonTransLe\}, \\ \{HourPrepWordLe, ToCLe\}, MvNpTransLe, \\ \{BeIdWas, \ldots\}, DegreeSpecLe, IntransAdjLe\rangle$$

Retrieval Retrieval is realized through traversal of the decision tree using the type chain as path description (also called input path), whereby traversal

- is directed by *type subsumption*, and
- supports *partial indexing*.

The first property means that during each step of traversal it is checked whether the current type element of the input path is subsumed by the type information of the current node in the decision tree. Only if this is the case traversal proceeds. Indexing simultaneously performs a *disambiguation* of the lexical input, since if an input word has two lexical readings (both belonging to two different types) the type incompatible reading can be discarded for further processing. Thus, during instantiation of a retrieved template a disambiguated lexical input will be considered.

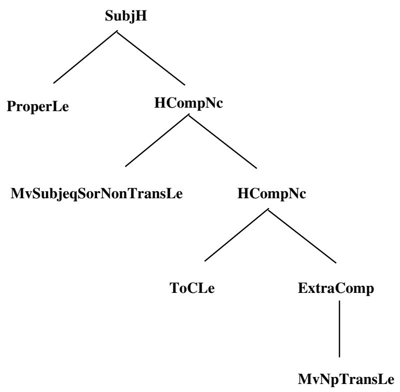
The core idea behind partial indexing is that in case the whole input path does not lead to a template, we want at least as many phrasal templates as be possible to be retrieved. Phrasal indexing is achieved by considering potentially each sub-path of the input path as a possible index in the decision tree. In order to achieve this, we have to recursively repeat retrieval of the decision tree as long as the remaining suffix is not empty. In other words, the decision tree is now a finite representation of an infinite structure, because, implicitly, each endpoint of an index bears a pointer to the root of the decision tree. It is assumed that the following template/index pairs have been inserted into the decision tree: $\langle ab, t_1\rangle$, $\langle abcd, t_2\rangle$, $\langle bcd, t_3\rangle$. Then retrieval using the path *abcd* will return all three templates, retrieval using *aabbcd* will return template t_1 and t_3, and *abc* will only return t_1.[3]

Retrieval for the type chain of the current example β proceeds as follows: We start with the whole input path p. Since, the first element $DetSgLe$ does not match the root of the decision tree, we reduce p by $DetSgLe$ which gives p_1. None of the alternatives of p_1's first element match the root, so we reduce p_1 by its first element, which gives p_2. Now, for the prefix p_3

$$\langle ProperLe, MvSubjeqSorNonTransLe, \\ ToCLe, MvNpTransLe\rangle$$

the phrasal template dt is found:

[3] It is possible to parameterize our system whether to perform an *exhaustive* or a *non-exhaustive* strategy. In the non-exhaustive mode, longest matching prefixes are preferred.



Assuming we follow the non-exhaustive strategy, the new suffix p_4 is obtained by deleting the prefix p_3 from p_2.[4] Since, no more template will be retrieved retrieval stops and returns dt.

Instantiation A successfully retrieved template dt is instantiated in two steps, called *expansion* and *terminal matching*. A template is expanded by *deterministically* applying the rules denoted by the non-terminal elements from the top downwards exactly in the order specified by dt. In some sense, expansion just re-plays the derivation obtained in the past. This will result in a grammatically fully expanded feature structure, where only lexical specific information is still missing. But note, that through structure sharing the terminal elements will already be constrained by syntactic information.[5]

In the last step of the application phase, the *disambiguated* lexical input (which is determined using the type chain leading to the template, in our example p_3) is unified with the constraints of the terminal yield of the expanded template. We also call this step *terminal-matching*. In our current system terminal-matching is performed from left to right. Since the ordering of the terminal yield is given by the template, it is also possible to follow other selection strategies, e.g., a head-driven strategy, which could lead to more efficient terminal-matching, because the head element is supposed to provide selectional restriction information for its dependents.

This ends the application phase for our second example β. The found, instantiated template together with the lexical input are passed as *passive edges* to the chart parser's agenda. Note, that this means that for the substring "Kim wanted to try" we already have found one solution.

[4]In case of an exhaustive mode, the new suffix would be computed from p_2 by deleting only p_2's first element.

[5]It is possible to perform the expansion step already offline during the training phase, in which case the application phase can be speeded up, however at the price of increased storage requirements.

After general parsing the training phase of TM is started, and performs more or less in the same way as for the first example α. In particular, TM will recognize that the derivation subtree dt (retrieved during AM) is already known as a template, and consequently will avoid redundant storing.

5 Implementation and performance

Implementation The EBL method just described has been fully implemented and tested with a broad coverage HPSG-based English grammar, medium and small grammars for German and Japanese.[6] The TDL grammar formalism is very powerful. It supports distributed disjunction and full negation, as well as full boolean type logic. In our current system, an efficient chart-based bidirectional parser is used for performing the training phase.

Each grammar which can be processed by the parser can also directly run with the learning module without any specifications requiering to be made by a user. However, the user can control the generalization and extraction process by providing type abstraction criteria and by defining filters (or by selecting a subset of predefined filters).

Performance Preliminary experiments were made using the English grammar and a training set of 214 sentences. This set was originally defined for testing the range of different grammatical constructions. During the training phase only the best (manually selected) readings were chosen. The training phase performed by TM is very fast (average real-time 0.15 sec.) The application phase (retrieval and instantiation) is also very fast. Applying the training set only by AM, processing is 5 to 20 times faster than parsing (real-time). This is due to the fact, that the used grammars usually use many disjunctive constraints which have to be resolved by the constraint-solver, and for which the parser has no control. Where parsing can re-use previously learned structures processing time increases by 2 to 5 times. These first results are encouraging. We have now begun broader experiments including testing the domain-specific coverage of the subgrammar and a detailed comparison of performance of different grammatical frameworks. We will report on these results in the final version of the paper.

6 Extensions

The basic architecture has already be extented in three directions, which we will briefly describe now.

[6]The English grammar has been developed at CSLI, Stanford, and kindly been made available to the author.

Dynamic memory management This extension concerns a method for controlling the size of the decision tree. For each template the access frequency, the last time of access, and the number of failed and successful instantiations is recorded. This information is then used to decide on the deletion of pointers (i.e., paths of the decision tree) leading to "un-productive" templates, i.e., templates which either are used very rarely or fail in most cases of instantiation. In our current system, this information is used after training of a corpus has completely been processed. In one experiment using a training corpus of 214 sentences a threshold was set such that each template which was not used at least twice during training was deleted. This led to a reduction of the decision tree of about 80 %.[7]

Morphology In case a morphological component is used as a preprocessor of the lexical component (which is the case for the English and German grammars), we also adapted the on-line method for morphological processing. The core idea here is to maintain an additional decision tree d_{morph} which stores the unambiguous chain of the morphological result of a sentence α obtained after parsing. During the application phase d_{morph} is used for disambiguation of morphological readings much in the same way as the decision tree obtained during parsing is used for the disambiguation of lexical elements. This leads to a *cascaded* flow of control. First experiments showed an doubling of the overall performance, however with the lost of some coverage. This is due to the fact, that d_{morph} actually serves as a filter, and hence can fade out some correct analysis.

Generation We have already implemented a version of the on-line method for tactical generation, where the task is to generate all possible strings for a given logical form. The core idea is to attach to each node of an extracted derivation tree a generalized form of the (local) semantic expression found in the corresponding feature structure. This generalized semantic expression is then used for indexing the template. Expansion and instantiation of a retrieved template is performed in the same way as for parsing. The main difference is that additional lexical lookup has to be done which is directed by information provided through the input semantics and by the constraints attached to the terminal yield of an expanded derivation tree. The EBL-based generator has been tested with the English grammar which uses a "flat" representation of event semantics to represent logical forms (see [Kay, 1996; Copestake *et al.*, 1996]). Although a flat semantic expression is operationally be analyzed as a free word-order expression [Kay, 1996], it has been possible to use the same decision tree mechanism as the one used for parsing.

[7] More precisely, the number of nodes of the decision tree decreased from 1358 to 148 (89.1%) and the number of templates from 372 to 68 (81.72%). This large amount of "unproductive" templates is due to the fact that the training set was originally defined for testing the grammar's coverage of different grammatical phenomena, not for testing the grammar's range of prototypical constructions.

7 Related work

Recently, EBL has been applied to the problem of automatically extracting subgrammars by [Rayner, 1988; Samuelsson and Rayner, 1991; Neumann, 1994; Srinivas and Joshi, 1995; Rayner and Carter, 1996]. However, neither of them follow an on-line strategy, i.e., they are forced to parse the whole training corpus before the subgrammar extraction process starts. Furthermore, our approach is the first *uniform* one, in the sense that it is used in the same way for parsing and generation.

8 Conclusion

We presented an *on-line* method of automatic subgrammar extraction based on EBL, in which the general parser and the subgrammar parser are strictly interleaved allowing for sub-sentential interactions.

A central aspect of our approach for the development of future NLP systems is that it is directed towards the incremental adaptation of an NLP to a particular language use of a user or group of users. In this sense, our method is an intelligent storage unit which maintains linguistic knowledge obtained through experience. More generally, we view our approach as an important step towards a computational model of experience-based natural language systems.

Our near-time goals will concentrate on the improvement of the dynamic memory management, the integration of our system with shallow strategies, and the adaptation of the pruning methods described in [Rayner and Carter, 1996] especially for robust processing.

References

[Briscoe and Carroll, 1993] T. Briscoe and J. Carroll. Generalized probabilistic lr parsing of natural language (corpora) with unification-based grammars. *Computational linguistics*, 19(1):25–59, 1993.

[Copestake *et al.*, 1996] A. Copestake, D. Flickinger, R. Malouf, S. Riehemann, and I. Sag. Translation using minimal recursion semantics. In *Proceedings, 6th International Conference on Theoretical and Methodological Issues in Machine Translation*, 1996.

[Grishman and Sundheim, 1996] R. Grishman and B. Sundheim. Message Understanding Conference – 6: A Brief History. In *Proceedings of* *the 16th International Conference on Computational Linguistics (COLING)*, pages 466–471, Kopenhagen, Denmark, Europe, 1996.

[Kay, 1996] M. Kay. Chart generation. In *34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, Ca., 1996.

[Krieger and Schäfer, 1994] Hans-Ulrich Krieger and Ulrich Schäfer. $\mathcal{TDL}$—a type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94*, pages 893–899, 1994.

[Minton *et al.*, 1989] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R.Kuokka, O. Etzioni, and Y.Gi. Explanation-based learning: A problem solving perspective. *Artifical Intelligence*, 40:63–115, 1989.

[Neumann, 1994] G. Neumann. Application of explanation-based learning for efficient processing of constraint-based grammars. In *Proceedings of the Tenth IEEE Conference on Artifical Intelligence for Applications*, pages 208–215, San Antonio, Texas, March 1994.

[Pollard and Sag, 1994] C. Pollard and I. M. Sag. *Head-Driven Phrase Structure Grammar.* Center for the Study of Language and Information Stanford, 1994.

[Rayner and Carter, 1996] M. Rayner and D. Carter. Fast parsing using pruning and grammar specialization. In *34th Annual Meeting of the Association for Computational Linguistics*, Morristown, New Jersey, 1996.

[Rayner, 1988] M. Rayner. Applying explanation-based generalization to natural language processing. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1988.

[Samuelsson and Rayner, 1991] C. Samuelsson and M. Rayner. Quantitative evaluation of explanation-based learning as an optimization tool for a large-scale natural language system. In *IJCAI-91*, pages 609–615, Sydney, Australia, 1991.

[Samuelsson, 1994] C. Samuelsson. *Fast Natural-Language Parsing Using Explanation-Based Learning.* PhD thesis, Swedish Institute of Computer Science, Kista, Sweden, 1994.

[Srinivas and Joshi, 1995] B. Srinivas and A. Joshi. Some novel applications of explanation-based learning to parsing lexicalized tree-adjoining grammars. In *33th Annual Meeting of the Association for Computational Linguistics*, Cambridge, MA, 1995.