3.1 Allgemeine Angaben zum Teilprojekt B4

3.1.1 Thema

From Constraints to Rules: Efficient Compilation of HPSG Grammars

3.1.1.1 Fachgebiet und Arbeitsrichtung

Computerlinguistik

3.1.2 Leiter

Prof. Dr. Erhard W. Hinrichs
Seminar für Sprachwissenschaft der Universität Tübingen
Kl. Wilhelmstr. 113
72074 Tübingen
Tel.: 07071 / 29-4279

Dr. Dale Gerdemann
Seminar für Sprachwissenschaft der Universität Tübingen
Kl. Wilhelmstr. 113
72074 Tübingen
Tel.: 07071 / 29-4279

3.2 Zusammenfassung

Efficient feature-structure based parsing/generation systems typically employ a phrase-structure or definite-clause backbone. Modern constraint-based grammars such as HPSG, on the other hand, are expressed primarily in terms of constraints upon types of objects. Thus, implementing an HPSG grammar in a system such as ALE (Carpenter 1993), Troll (Gerdemann & Götz 1993) or CUF (Eisele & Dörre 1990) involves a good deal of error-prone loose interpretation on the part of the grammar implementor. The goal of this project will be to eliminate this grammar implementation step, by providing a compiler which takes as input an HPSG grammar as written by a linguist and produces as output an implemented and efficient grammar for ALE, Troll or CUF, depending upon the choice of the user. In addition to type constraints, HPSG grammars also include linear precedence constraints and lexical rules. Thus, the compiler will necessarily convert all of these constraints into an efficient set of rules for the chosen processing system.

3.3 Stand der Forschung

In the first phase of the B4 project, we have focused on control issues involved in generation with typed feature structure grammars. In particular, we have concentrated on off-line compilation techniques for top-down processing (Minnen et al. 1993, Minnen and Gerdemann 1993) and also on compilation for bottom-up processing (Minnen in preparation). Many such compilation techniques have wider application than just their use in generation. In the next phase of the project, we propose to use such compilation techniques to compile "pure" constraint-based HPSG grammars into efficiently processable definite-clause or phrase structure programs that may be run on a number of typed feature-structure implementations for HPSG.

There are currently a fair number of processing systems available which implement approximations to HPSG. Such systems typically provide a constraint language in combination with phrase structure rules and/or definite clauses. ALE (Carpenter 1993) and Troll (Gerdemann & Götz 1993), for example, have both phrase structure rules and definite clauses, whereas CUF (Eisele & Dörre 1990) has only definite clauses. These systems provide only approximations to HPSG since HPSG grammars themselves are expressed neither as phrase structure rules nor as definite clauses. Rather, HPSG is expressed mostly in terms of implicational constraints upon feature structures, with lexical rules and linear precedence constraints falling somewhat outside of this paradigm. The above-mentioned systems capture the HPSG-constraints embodied in the type hierarchy and the appropriateness specification. However, phrase-structure rules and definite clauses cannot adequately express the general implicational constraints allowed by HPSG. The problem is that a definite clause must be attached to a feature structure or to a phrase structure rule before it will apply, whereas an implicational constraint is intended to apply wherever its antecedent is satisfied. The task of translating a constraint-based grammar into one of these rule-based systems is by no means straightforward. Some of the intricacies of translating the Hinrichs & Nakazawa (1994) grammar into Troll are reported in Meurers (1994).

The primary problem in converting constraint-based grammars into rule-based systems is that it is quite error prone. Whereas the grammar writer may have a general principle such as the head feature principle, the grammar implementor may have to express this as a rule and then attach it in every place where it may be applicable. This is quite different from the general principle that applies to every sub-structure of a feature structure. Clearly, it is possible to overlook some places in which the constraint should apply. And if the grammar writer later makes changes, it will be even easier to overlook attaching the constraints to the new parts of the grammar.

There are, of course, systems such as TFS (Emele & Zajac 1990, Zajac 1992), which are closer to HPSG in that they provide general constraints on types. However, such systems are designed as grammar specification systems rather than efficient processing systems. Moreover, even TFS does not provide a full model for HPSG. Implicational constraints, for example, are limited to a single type as an antecedent. Moreover, linear precedence constraints and lexical rules are not readily expressed in such a system at all.

Rather than working directly in such a "grammar specification" system, however, it would be more efficient to translate HPSG grammars into a system of phrase structure rules, which could then be processed by specialized parsing algorithms that can take advantage of features such as charts and string position indexing. Up to now no one has developed such a compiler, though many of the necessary compilation techniques have been investigated or developed in the previous phase of B4.

Of course, one should realize that systems such as ALE, Troll and CUF already provide a major portion of the requirements for HPSG in that they provide for type inferencing with respect to a type hierarchy and an appropriateness specification. Our own work in the first phase of the project has focused to a large extent on improving the type inferencing as exhibited in ALE (Gerdemann & King 1993). We showed, in this work, that the type inferencing in ALE was insufficient to maintain satisfiability of feature structures, i.e., there are "well-typed" feature structures in ALE which have no maximal extensions. Any satisfactory approach to maintaining satisfiability must consider the various ways that the feature structure can be extended. Therefore, in the Troll system, implemented by Gerdemann and Götz and based on the logic of King (1989, 1994), ALE's type inferencing is replaced by a limited amount of named disjunctions representing the various possibilities of extending the feature structure. This approach is made possible by the use of finite "resolved" feature structures, which represent potentially infinite classes of completely extended feature structures. The amount of disjunction needed is kept limited first by only resolving as much as necessary (Gerdemann & King 1993), and second, by unfilling redundant branches in the feature structure (Götz 1993). This approach efficiently implements an important part of HPSG, but it leaves aside the question of implicational constraints. The continuation of B4 is intended as a continuation of this line of research to develop a true environment for HPSG and other constraint-based formalisms.

3.4 Eigene Vorarbeiten

King's (1989, 1994) development of a logical formalism for HPSG theories served as the basis for our computational Troll system. The use of this logic in the Troll system is described in Gerdemann & King (1993, 1994), in which it is shown how satisfiability of feature structures can be maintained with an efficient approach to unification. It is also shown how feature coocurence restrictions are encoded in the type hierarchy and appropriateness specifications. These papers also discuss the technique of unfilling feature structures, which is crucial for efficient compact representations. Unfilling is also examined from a logical perspective in Götz (1993).

The work in the next phase of this project will also rely heavily on the treatment of typed feature structure negation described in Kepser (forthcoming). Our approach to negation will also depend on the *partition* and *all-or-nothing* conditions on the signature as defined in Gerdemann & King (1993, 1994) and described more extensively in section 3.5.2. Since we propose compiling negations into potentially large disjunctions, an efficient approach to disjunction as described in Griffith (forthcoming), will also be of use.

In the current phase of the B4 project we have investigated off-line compilation techniques for parsing and generation. This led to the development of a number of techniques which compile a declarative rule-based grammar specification into a representation that is particularly suitable for processing in a specific direction, i.e., parsing or generation (Minnen and Gerdemann 1993, Minnen et al. 1993, Minnen in preparation). The developed techniques are based on a specification of the intended input for a specific processing direction. In case of parsing this means that an input grammar is translated into a possibly procedurally annotated but semantically equivalent representation that can be used efficiently in case the input string to be parsed is known a priori. We intend to use these techniques in the next phase of B4 in order to transform definite clauses into a form that can be efficiently executed in CUF or in the definite clause component of Troll or ALE. The transformation will also use techniques of folding/unfolding, which we have investigated in Gerdemann (1994). In addition, previous work on normal forms for definite clause grammars by Martini (1994) might be of benefit.

In the next phase of B4, we also intend to incorporate a treatment of linear precedence constraints. To this end, we will partially rely on the previous work of Minnen (1992,1994), in which LP-constraints are incorporated into TAG grammars. We will also further explore the approach of Morawietz (forthcoming), in which it is shown how LP-constraints can be dynamically checked during parsing to rule out LP violations as early as possible.

3.5 Ziele, Methoden, Arbeitsprogramm und Zeitplan

3.5.1 Ziele

Our goal will be to provide a compiler that converts pure constraint-based HPSG grammars into more efficient rule-based grammars. By "rule-based", we mean minimally a set of definite clauses which have been transformed so as to run efficiently on a system such as CUF or in the definite clause component of ALE or Troll. Ultimately, however, we intend our rules to be feature structure based phrase structure rules with definite clause attachments which could then be interpreted by any number of very efficient parsing algorithms. We then intend to further compile the definite clause attachments so that they may be efficiently processed top-down, or by so-called magic

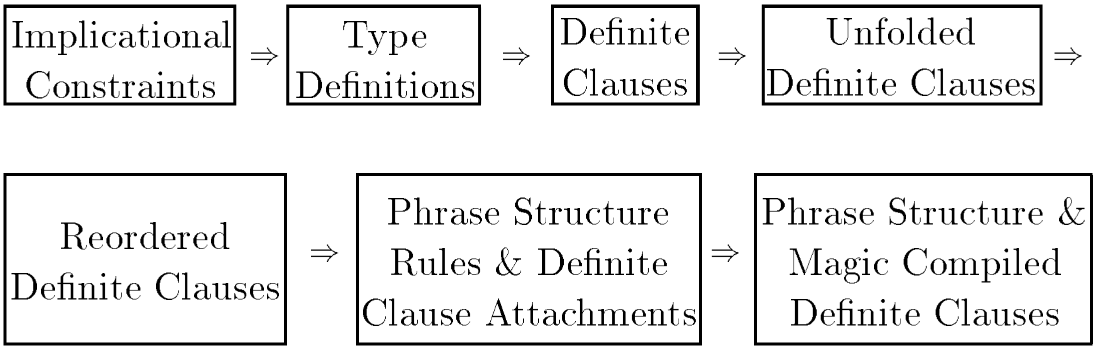


Figure 1: Compiler Architecture.

compilation, so that they may be efficiently processed bottom-up. Ultimately, this compilation will allow the grammar writer to express a grammar in a very high level constraint-based language without concern for efficiency and control issues.

We envisage this compilation as involving the steps laid out in the following diagram.

The first compilation step will be to compile into TFS-style type definitions, i.e., implications with just a single type as antecedent. These type definitions can then be compiled into a set of definite clauses. The resulting Prolog program is, however, not likely to be very efficient. By a process of fold/unfolding and reordering, we intend to transform these clauses into a program that would run efficiently in a system such as CUF or in the definite clause interpreter of Troll or ALE. Then, by isolating the definite clauses that manipulate parts of the phonology, we propose to extract phrase structure rules from these reordered definite clauses. The final compilation step will be to do magic compilation on the definite clause attachments to achieve an efficient implementation of the grammar.

All of this compilation starts with the implicational constraints. Since ID-schemata, grammar principles and relational extensions are expressed in terms of implications between feature structures, this will account for a major portion of HPSG. However, not all of HPSG is expressed in terms of such implicational constraints. LP-constraints and lexical rules, in particular, are expressed as different sorts of constraints which are triggered by subsumption. How to adequately incorporate LP-constraints and lexical rules into a logic for typed feature structures grammars is still an open research question. Here, we will investigate two possible approaches. First, it may be that there are ways of expressing such constraints either in the appropriateness specification or as implicational constraints. King (1989), for example, expresses some LP-constraints as implications on feature structures, and we believe that there are ways to extend this approach perhaps also to lexical rules. If LP-constraints and lexical rules can be incorporated into the appropriateness specification and/or the implicational constraints, then they could be handled in a uniform manner with the rest of the constraints in

HPSG. Otherwise, we will investigate ways of incorporating such constraints into the phrase structure output of our compiler. Since there are already methods available for incorporating LP-constraints into unification-based phrase structure grammars (e.g., Engelkamp et al. 1992), our approach of compiling purely constraint-based grammars into phrase structure rules would allow the application of such methods. Similarly, one may find approaches to integrating lexical rules into parsing, for example, along the lines of the run-time evaluation of meta-rules described by Weisweber & Preuß (1992). In the extension of B4, we will investigate these approaches to integrating LP-constraints and lexical rules into unification-based phrase structure grammars. However, we will prefer an approach that integrates these mechanisms into the standard kinds of HPSG constraints. In our investigation of these devices, we will work closely with the fragment project B8, which will be investigating linguistically adequate approaches to capturing word order and lexical generalizations.

3.5.2 Methoden

To be able to compile an HPSG grammar into ALE, CUF or Troll, we need to be precise about what constitutes an HPSG grammar. An HPSG grammar consists of several distinct modules or components that serve entirely different purposes. We will look at these modules one by one and then consider how they can be efficiently implemented.

1. **A type hierarchy with appropriateness specifications (signature).**

   The signature provides the structure of the domain, an ontology of linguistic entities. The type hierarchy defines which kinds of objects are to be considered and orders them hierarchically. The appropriateness conditions define which features objects of a certain type can and must have, and what values these features can have. Features in such a signature are normally interpreted as partial functions on the domain of objects. HPSG-II furthermore demands that linguistic objects be maximally specific, i.e., that every object is of exactly one type that doesn't subsume any more specific types. This means for example that there can be no signs that are neither words nor phrases. This is called the *partition condition* in Gerdemann & King (1993, 1994). A similar condition (the *all-or-nothing condition* of Gerdemann & King (1993, 1994)) is also required in HPSG-II. This condition says that for any given feature and any maximally specific type, either all of the objects of the type possess this feature, or none. An interpretation that satisfies these two conditions is often called a closed world interpretation. As Gerdemann & King (1993, 1994) have argued, both conditions are necessary so that the appropriateness conditions can fulfill their primary function of expressing feature cooccurrence restrictions.

As described above, this part of HPSG has already been implemented in the Troll system developed by Gerdemann & Götz based on the logic for HPSG of King (1994). The unifier developed for the Troll system in the first phase of the B4 project will continue to be used in the next phase since maintaining appropriateness conditions is a necessary feature of HPSG. Moreover, the partition and the all-or-nothing conditions, which can only be expressed in the type hierarchy and appropriateness specification, will have important consequences for the implementation of the implicational constraints described below. Specifically, these closed-world conditions make possible a simple approach to negation which can be used to express implication.

2. **A set of implicational constraints.**

   An HPSG grammar consists primarily of a set of implicational constraints. For example, instead of phrase structure rules, HPSG has an immediate dominance principle which is formulated in simplified form as follows:

$$\textit{phrase} \quad \Longrightarrow$$

$$\begin{bmatrix} \text{SUBCAT} & \langle\rangle \\ \text{DTRS}\,|\,\text{HEAD-DTR} & \textit{phrase} \end{bmatrix} \vee \begin{bmatrix} \text{SUBCAT} & \langle[\,]\rangle \\ \text{DTRS}\,|\,\text{HEAD-DTR} & \textit{word} \end{bmatrix}$$

   This rule says that anything that is a phrase must satisfy one of the consequent feature structures. By itself, this doesn't say very much. However, in conjunction with the subcategorization principle and LP-constraints, one can obtain phrase structure rules from this ID-principle.

   Similarly, the entire lexicon can be expressed by the single principle:

$$\textit{word} \quad \Longrightarrow \quad \textit{Lex1} \vee \textit{Lex2} \vee \textit{Lex3} \vee \textit{Lex4} \ldots$$

   where a Lex_i is a feature structure representing a lexical item. Thus any object of type *word* must be described by one of the lexical entries in the grammar.

   Both of these examples, however, are relative simple since the antecedent of the implication is a single type. These simple constraints may be viewed as *type definitions* as used, for example, in TFS. In this project, however, we will propose also to handle general implications, in which the antecedent is a complex feature structure. For example, the following formulation of the head feature principle has a complex feature structure of type *phrase* as its antecedent.

$$\begin{bmatrix} \textit{phrase} & \\ \text{DTRS} & \textit{headed-struc} \end{bmatrix} \quad \Longrightarrow$$

$$\begin{bmatrix} \text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{HEAD} & \boxed{1} \\ \text{DTRS}\,|\,\text{HEAD-DTR}\,|\,\text{SYNSEM}\,|\,\text{LOC}\,|\,\text{CAT}\,|\,\text{HEAD} & \boxed{1} \end{bmatrix}$$

> This principle indicates that if it is true of a linguistic object that it is of type *phrase*, and the value of its DTRS feature is of type *headed-struc*, then it must also be true of that object that its SYNSEM|LOC|CAT|HEAD value is its DTRS|HEAD-DTR|SYNSEM|LOC|CAT|HEAD value.
>
> Finally, HPSG's relational or functional extensions such as `append` and `collect-indices` may also be expressed as implicational constraints (King 1989, 1992). Of course grammar writers do not think of there being an "append principle" with the same ontological status as the head feature principle, so it makes sense for the user interface to provide another way of expressing such relational extensions. From the point of view of our compiler, however, such relational extensions will be considered as just another sort of implicational constraint.

Such implicational constraints will be compiled into definite clauses and further into phrase structure rules in several steps. The first step is to reformulate such conditionals into constraints on single types, i.e., to create type definitions as used, for example, in TFS (Emele & Zajac 1990, Zajac 1992). Then these type-definitions can be compiled into a set of Prolog clauses which effectively apply these type definitions to all sub-structures of an input query. This transformation will create a highly recursive program, which is not guaranteed to terminate and is certainly not guaranteed to be efficient. We will then apply off-line transformations to this program to ensure termination, where this is possible, and to ensure efficient execution. Then, since such transformation techniques are based on dataflow analyses, one can expect that the most efficient transformation of a program for parsing a grammar will be one in which the input is traversed from left to right. By isolating the right-hand side clauses which traverse part of the input, one can then extract a phrase structure backbone. The remainder of the clauses will then be considered as definite clause attachments to this grammar. Finally, we will compile these attachments so that they too may be efficiently processed either top-down, or—by using magic compilation—bottom-up.

The first step then will be to compile general implicational constraints, such as the head feature principle, into type definitions, i.e., implications with a single type as antecedent. Here, we can take advantage of our closed world assumptions: the partition and all-or-nothing conditions Gerdemann & King (1993, 1994). Although we previously argued for these conditions on the grounds that they help to formulate feature cooccurrence restrictions, it turns out that they also allow for a rather simple treatment of the negation that is needed for general implications (see Kepser forth. for the theoretical background to our treatment of negation). For example, the implication in the head feature principle (above) could be reformulated as the following four way disjunction, which says of every object:

1. The object is of a type not subsumed by *phrase*, or
2. the object is of some subtype of *phrase* which does not allow a DTRS feature, or
3. the object is of type *phrase*, with a DTRS feature that takes a value not subsumed by *headed-struct*, or
4. the path SYNSEM|LOC|CAT|HEAD is reentrant with the path DTRS|HEAD-DTR|SYNSEM|LOC|CAT|HEAD.

Suppose we abbreviate the disjunction of all species (maximally specific types) which are not subsumed by *phrase* as $phrase^-$ and all species not subsumed by *headed-struct* as $headed\text{-}struct^-$. Furthermore, abbreviate the disjunction of all daughterless subtypes of *phrase* as $phrase^{-dtr}$ (presumably the empty disjunction). The head feature principle can then be reformulated as the following disjunctive feature structure:

$$\left\{ phrase^-, headed\text{-}struct^-, phrase^{-dtr}, \begin{bmatrix} \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} \mid \text{HEAD} & \boxed{1} \\ \text{DTRS} \mid \ \ldots \mid \text{HEAD} & \boxed{1} \end{bmatrix} \right\}$$

Similarly, all other principles can be compiled into disjunctions, with the one complication that reentrancies in the antecedent of a principle will necessitate the introduction of anti-reentrancies as discussed in Carpenter (1992).

Once all principles are compiled into such disjunctions, they can then all be unified together by pairwise unifying the disjuncts. The result is then, that all of the principles, ID-schemata and lexicon will be expressed as one large disjunction. Given this disjunction, we can then extract out type definitions (or perhaps just species definitions, if that is all we need). The principle is quite simple. For each type t, make a single node feature structure T whose node is labeled with t. Then unify T with the large disjunction for our compiled principles. The presumably small number of disjuncts that successfully unify with T are then the type definitions for T.

Once all implications are compiled into type definitions, we have the grammar in a form in which it could be interpreted by a system such as TFS, if TFS were extended to handle anti-reentrancies. However, this sort of grammar specification system is not adequate to use as an efficient parser. To transform the grammar into an efficiently processable set of rules, we first compile into definite clauses as follows. This compiler can best be explained by example. Suppose that we have the following type definition for *headed-phrase*.

$$headed\text{-}phrase \quad \Longrightarrow \quad \begin{bmatrix} \text{SYNSEM} \mid \text{LOC} \mid \text{CAT} \mid \text{HEAD} & \boxed{1} \\ \text{DTRS} \mid \text{HEAD-DTR} \mid \text{LOC} \mid \text{CAT} \mid \text{HEAD} & \boxed{1} \end{bmatrix}$$

We can then compile this type definition into a set of Prolog clauses which given a feature structure F, either instantiate F to a well formed feature structure of type *headed-phrase* or report failure. All other type definitions are compiled similarly. Of course this program will be quite recursive since a headed phrase likely contains other headed phrases. The compiled Prolog program will then essentially apply all type definitions in a fan-out order as follows:

1. First unify the root node with the type definition of *headed-phrase*.
2. For each top-level feature, find the type of the value of that feature and recursively apply the corresponding compiled type definitions to each of these values.

One should note here that `unify` means to use an algorithm for typed unification, so that appropriateness specifications are maintained in the process.

It's not terribly difficult to transform a set of type definitions into a set of Prolog clauses which implement this simple algorithm. The problem is that the program may not terminate. There are two aspects to why the program may not terminate. Firstly there is a theoretical issue due to the additional complexity which stems from the anti-reentrancies. Here, we expect to collaborate with the A8 project (Mönnich). We hope, thereby, to provide a proper fixed-point semantics for the algorithm using domain theoretic methods and thus to show that the algorithm as such will terminate. Secondly, there may be termination problems due to the introduction of recursion which may force Prolog into infinite loops. It is at this point that we propose to use the program transformation techniques that were investigated in the first phase of the B4 project. In fact, the transformation from type definitions to Prolog program, as simple as it is, really opens up a wealth of possibilities since there is now a very large body of literature on transformation techniques for Prolog clauses.

In fact, the compiled definite clauses could already be used as an input to CUF or to the definite clause interpreters of Troll or ALE. CUF or Troll would, in fact, be the best alternatives since they both employ a freeze mechanism, i.e., the delay patterns described by Dörre & Dorna (1993), which would allow some possibility for efficient execution. However, given the large amount of recursion that would arise in the compiled type definitions, it is doubtful whether delay patterns alone would be sufficient for efficient execution. Moreover, even in CUF delay patterns are not automatically generated, so the grammar writer is still not freed from processing concerns. Thus, even for CUF, an off-line transformed set of definite clauses could greatly increase efficiency.

As a crucial first step toward transformation, we will employ the unfold/fold transformations as described in Kanamori & Horiuchi (1987). It will be important to unfold calls to the unifier in order to isolate calls that involve unifications on the phonology feature and to have these calls separate from the rest of the feature structure unification, so that they can be reordered for left-to-right string traversal. These unfold/fold transformations are a generalization of partial execution as described in e.g., Pereira

& Shieber (1987). They have already been applied to grammar transformations for implementations of Government and Binding theory by Johnson & Stabler (1993) and they are used also in Gerdemann (1994), where it is shown how the main classes of parsers (top-down, bottom-up and left-corner) are related by simple differences in folding/unfolding of phrase structure constraints followed by transformations to eliminate left recursion.

After the unfold/fold transformation, the Prolog program will be transformed for efficient execution. Still there are two very different types of transformation that one could use. On the one hand there are transformation techniques for the purpose of top-down evaluation as in the Direct Inversion Approach (DIA; Minnen et al. 1993, Minnen and Gerdemann 1993) or there are transformation techniques for the purpose of bottom-up evaluation as in the magic compilation of Beeri and Ramakrishnan (1991) and Ramakrishnan (1991), which is extended to use with typed feature structures in Minnen (in preparation).

Initially, our work in the B4 project concentrated on transformations for top-down evaluation. We developed in this phase a new approach to parsing and generation with logic grammars, the Direct Inversion Approach (DIA; Minnen et al. 1993, Minnen and Gerdemann 1993). In the DIA a declarative grammar specification is compiled such that it can be successfully executed given a top-down processing strategy. The compilation process consists of:

1. *Intraclausal reordering*: change of the order of argument instantiation, by means of reordering literals within the clauses of the grammar. The optimal ordering for each clause in the grammar is computed by means of a dataflow analysis of the grammar with respect to the intended input specification.

2. *Interclausal reordering*: reordering the literals between clauses of the grammar, in order to ensure termination of recursive predicates. This transforms recursive predicates in order to form a "guide" in the sense of Dymetman et al. (1990), i.e., an argument such as a list that is one element shorter on each recursive call. This is particularly important for top-down evaluation of HPSG, where subcat lists are normally one element longer on each recursive call.

More recently, our work in B4 has concentrated on magic compilation (Beeri and Ramakrishnan 1991, Ramakrishnan 1991) and the extension of this technique to typed feature structures (Minnen in preparation). Since this approach assumes bottom-up evaluation, recursive predicates are less of a problem. But efficiency becomes a central concern since forward chaining is quite non-directed. This lack of guidance is overcome, however, by introducing so-called *magic* predicates into the program, which are in effect precompiled predictions as used in Earley deduction (Pereira and Warren 1983).

Both of these compilation techniques can be used to obtain a more efficiently processable set of definite clauses for top-down evaluation as in CUF or in our own bottom-up interpreter. Abstracting across both kinds of transformations, however, Beeri and Ramakrishnan (1991) have noted that the common central feature is a transformation to achieve the optimal sideways information passing (SIP). Thus right-hand side clauses are reordered so that output arguments of clauses to the left feed essential input arguments of clauses to the right, where "essential" means essential for efficient execution. If our definite clauses are intended to be used for parsing, then only the phonology will be initially instantiated and the optimal SIP will be one in which this phonology is scanned from left to right. Thus, one might expect a transformed clause to look something like the following clause underscores represent non-phonological arguments.

```
p(_,_,Phon0,Phon):-
   p1(_,_,_),
   p2(_,_,Phon0,Phon1),
   p3(_),
   p4(_,Phon1,Phon2),
   p5(_,_,_,Phon2,Phon).
```

Of course, such a program can be trivially transformed one step further to actually obtain a phrase structure backbone as in:

```
p(_,_) -->
   {p1(_,_,_)},
   p2(_,_),
   {p3(_)},
   p4(_),
   p5(_,_,_).
```

The significance of this transformation can hardly be overstated. Once the clauses are transformed to obtain such a phrase structure backbone, the whole realm of parsing techniques (top-down, bottom-up, left-corner, etc.) becomes available for efficient interpretation of the grammar. In particular, a phrase structure backbone allows very efficient memoing on the basis of string position indexing.

Furthermore, such a transformation provides a clean separation between those clauses that could be considered to be phrase structure categories from clauses which represent definite clause attachments. This allows for a separate investigation of the optimal strategy in evaluating these attachments. In this project, we intend to investigate the use of magic compilation specifically with these definite clause attachments.

3. **A set of linear precedence (LP) constraints.** LP-constraints are common to both HPSG and GPSG. Following Engelkamp et al. (1992), we may define an LP-constraint to be an ordered pair $\langle A, B \rangle$ of descriptions (feature structures), such

> that if A describes α[1] and B describes β, where α and β are in the same domain (normally a local tree), then α precedes β in that domain. LP-constraints are the primary mechanism used in HPSG for expressing word order generalizations.

LP-constraints were originally developed in GPSG in order to capture word order constraints across large numbers of phrase structure rules. They were then adopted into HPSG even though HPSG never had large numbers of phrase structure rules to generalize across. Rather than viewing HPSG LP-constraints as generalizations across phrase structure rules, it makes more sense to view them as generalizations across structures. Given this point of view, it would be attractive to incorporate LP-constraints into the implicational constraints that are normally used for expressing generalizations across structures.

In fact, it is possible to express an LP-constraint as an implicational constraint. As an example, suppose we wish to encode the LP-constraint $\langle FS1, FS2\rangle$. For this purpose, we define a type *local_domain*, which takes two features LIST_OF_SIGNS and PRECEDENCE. The feature PRECEDENCE takes as its value an object of type *precede*, which will be defined as a relational extension that picks out two signs in order from the list of signs. Then the LP-constraint can be encoded as follows:

$$\begin{bmatrix} \textit{local\_domain} & \\ \text{LIST\_OF\_SIGNS} & \boxed{3} \\ \text{PRECEDENCE} & \begin{bmatrix} \textit{precede} & \\ \text{SIGN1} & \boxed{1} \\ \text{SIGN2} & \boxed{2} \\ \text{LIST\_OF\_SIGNS} & \boxed{3} \end{bmatrix} \end{bmatrix} \Longrightarrow$$

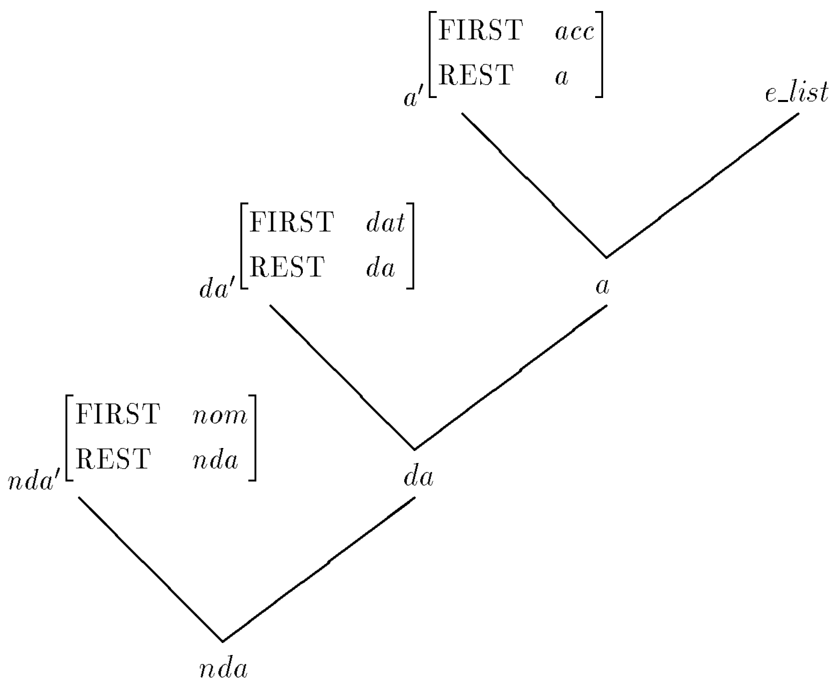
$$\left(\begin{bmatrix} \textit{local\_domain} & \\ \text{PRECEDENCE} & \begin{bmatrix} \textit{precede} & \\ \text{SIGN1} & \neg\texttt{FS2} \end{bmatrix} \end{bmatrix} \vee \begin{bmatrix} \textit{local\_domain} & \\ \text{PRECEDENCE} & \begin{bmatrix} \textit{precede} & \\ \text{SIGN2} & \neg\texttt{FS1} \end{bmatrix} \end{bmatrix} \right)$$

This approach has the virtue, then, of encoding LP-constraints just as any other principle. However, the approach is not likely to be very efficient. This constraint contains a considerable amount of negation, which would be compiled out to a rather large disjunction.

Another approach is to encode LP-constraints at least partially in the signature as if LP-constraints were another kind of feature cooccurrence restriction. Just as it is

[1]Engelkamp et al. say that A *subsumes* (rather than describes) α. However, subsumption is a relation between two feature structures and not a relation between a feature structure and an object. Note that a feature structure F describes an object just in case $\neg F$ does not describe the object, i.e., negation is classical.

possible to define a list of signs as a type, it is also possible to define a list of A's followed by B's followed by C's. Suppose, for example, that nominatives always come first, followed by datives and then accusatives. Then we can define the following types for lists: *nda* (for nominative-dative-accusative), *da* (for dative-accusative) and *a* (for accusative). And assume also primed types for proper lists as shown in the following signature.



As seen here, the types record which kinds of elements have been found so far in the list and thus they restrict what elements may be found in the rest of the list. The type *da*, for example, records that a dative element has been found and therefore rules out further occurrences of nominative. This is, in effect, an encoding in the signature of the more lexically-based approach of Engelkamp et al. (1992), which is also presented below.

This approach would certainly be more efficient than the approach of encoding LP-constraints as implicational constraints. The approach is restricted, however, since it can only order elements by their root type. To order signs on the basis of grammatical case, it would be necessary to make up new types such as *dat-sign* and *acc-sign*. Then implicational constraints would be needed to ensure that a *dat-sign* had *dat* as the value of the appropriate path. Still this approach employing a combination of appropriateness specifications and implicational constraints would likely be more efficient than the approach using only the latter. We will, therefore, pursue this line most actively in the continuation of the B4 project.

In addition to encoding LP-constraints into the implicational constraints or the signature, there are also approaches that involve using LP-constraints in conjunction with a phrase structure backbone. Since the ultimate aim of our project is to compile constraint-based grammars into such a framework, these possibilities would also be opened up for us. There are two basic alternatives along this line, which we will now consider. First, one could incorporate LP-constraints into the grammar as in Engelkamp et al. (1992) or second, one could use apply LP-constraints at run time to dynamically rule out ill-formed parses (Seiffert 1987; Minnen 1992, 1994; Morawietz forthcoming).

The approach of Engelkamp et al. (1992) incorporates LP-constraints into lexical entries. Each category encodes as a feature value, those categories that are not allowed either to the left or to the right. Then there is a feature LP-STORE which encodes as one goes up the tree in a local domain, what categories have been encountered so far. In effect, Engelkamp et al. are encoding with feature values, the same information that was encoded in the signature in the previous approach. This approach also has its own attractiveness. Since HPSG is noted for being a very lexically based theory, it might make sense to encode word order also on lexical items. On the other hand, this approach significantly complicates lexical entries. Engelkamp et al. state that they wouldn't expect grammar writers to state word order constraints in this manner, but rather that a compiler should compile standard LP-constraints into this format.

Finally, one may evaluate LP-constraints dynamically during parsing. Such an approach was originally suggested by Shieber (1984) for GPSG. Seiffert (1987), however, notes that non-local flow of information in unification-based grammars may make it necessary to delay checking LP-constraints until all feature values are instantiated. This approach is improved upon in Morawietz (forthcoming) where it is shown how LP-constraints can be delayed until the proper feature values become instantiated for the LP-constraint to be checked. As opposed to the generate-and-test approach of Seiffert, this approach allows LP-constraints to be used actively in ruling out possible parses. Perhaps the most attractive feature of this approach is that it stays very close to the original linguist's conception of LP-constraints. In Minnen (1992, 1994), Shieber's approach was applied to a variant of TAG which factors LP.

In summary, there are a variety of possible approaches to LP-constraints or to word order in general. Each approach has its attractive features. However, each approach also adds a rather significant level of processing complexity. In our investigation, we will determine which of these approaches leads to the most efficient encoding. We will, of course, also be guided by linguistic considerations, since our approach must be linguistically adequate. Thus, we intend to work in close cooperation with the B8 project (and other projects investigating word order) to find the most satisfactory approach.

4. **A set of lexical rules.**

Since HPSG is a very lexically based theory, lexical rules play an essential role in capturing generalizations. An example is the passivization rule from Pollard

& Sag (1987).

$$\begin{bmatrix} \mathit{base} \wedge \mathit{trans} & \\ \text{PHON} & \boxed{1} \\ \text{PAST-PART} & \boxed{2} \\ \text{SYN}\,|\,\text{LOC}\,|\,\text{SUBCAT} & \langle \ldots, [\,]_{\boxed{3}}, [\,]_{\boxed{4}} \rangle \\ \text{SEM}\,|\,\text{CONTENT} & \boxed{5} \end{bmatrix} \quad \mapsto$$

$$\begin{bmatrix} \mathit{passive} & \\ \text{PHON} & f_{PSP}(\boxed{1}, \boxed{2}) \\ \text{SYN}\,|\,\text{LOC}\,|\,\text{SUBCAT} & \langle (PP[BY]_{\boxed{4}}), \ldots, [\,]_{\boxed{3}} \rangle \\ \text{SEM}\,|\,\text{CONTENT} & \boxed{5} \end{bmatrix}$$

The left hand side of this rule may be viewed as the structural description, defining the environment in which passivization takes place. This rule applies to a basic or derived lexical entry when the left hand side of the rule subsumes the lexical entry. The lexical rule must be triggered by subsumption rather than unifiability, since it is intended to apply only to lexical entries that are *both* base and transitive. When the lexical rule applies, a new lexical entry is created corresponding to the right hand side of the rule. The tags, i.e., "reentrancy" numbers, indicate pieces of structure which are copied over from the input lexical entry to the output entry. Any pieces of structure wich are not specifically referred to are copied into the output *by default.*

Such lexical rules clearly have a very procedural flavor to them, which is quite inconsistent with the rest of HPSG. Nevertheless, such rules are so integral to HPSG that no system could claim to be an implementation of this theory without some account of these rules. Most available implementations, however, fall far short of really implementing such rules. The ALE system, for example, allows lexical rules to be triggered by unification and then also fails to maintain the default copying of unspecified path values into the output lexical entry. From a computational point of view, the most significant shortcoming of ALE is that it compiles out the full lexicon.

A major goal in the next phase of B4 will then have to be the development of a more declarative notion of lexical rules that will fit better into the overall framework of HPSG. Specifically, since the entire lexicon is expressed by the implicational constraint:

$$\mathit{word} \quad \Longrightarrow \quad \mathit{Lex1} \vee \mathit{Lex2} \vee \mathit{Lex3} \vee \mathit{Lex4} \ldots$$

it would make sense to also include lexical rules under such constraints.

To obtain this, one would first need a more complex notion of word. Just as the type *phrase* includes a DTRS feature to indicate how the phrase was built up, we expect that *word* should have a DERIVATION feature to indicate how the word was formed. Thus, a word might have the following form:

$$\begin{bmatrix} \textit{word} & \\ \text{ENTRY} & \boxed{1} \\ \text{DERIVATION} & \begin{bmatrix} \textit{lexical\_rule} & \\ \text{INPUT} & \textit{word} \\ \text{OUTPUT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

The precise form that a word should take, however, must be determined in consultation with other projects dealing with the lexicon, in particular the B8 project.

This approach would integrate lexical rules into ordinary constraints, which suggests that nothing special needs to be said about them. From a computational point of view, however, we know that this is not true. The lexicon is the largest part of any grammar, so it is very important that this information is stored efficiently and in such a way that it can be accessed efficiently. Our goal in this project will be to find a uniform way of expressing lexical rules, but at the same time to be able to treat the lexicon as special for the purpose of efficient access.

Just as with LP-constraints, an alternative approach might be to find an approach in which lexical rules are interpreted in association with phrase structure rules. Unlike LP-constraints, however, there has been very little research on how lexical rules might be dynamically applied during parsing. There has, however, been some work on how to apply metarules dynamically as in Weisweber & Preuß (1992). Since the GPSG metarules mostly correspond to the HPSG lexical rules, an approach to metarules may also be applicable as an approach to incorporating lexical rules. As with LP-constraints, however, our first preference will be to treat lexical rules within the standard mechanisms of HPSG, i.e., the signature and the implicational constraints, before we consider an approach that integrates interpretation of phrase structure and lexical rules.

3.5.3 Arbeitsprogramm

In order to reach the research goals stated in the previous section, the following subtasks need to be carried out. Since the B4 project combines aspects of logic, linguistics and computation these subtasks are listed by topic area.

Logic

A. Develop logical foundations for linear precedence constraints.
B. Develop logical approach to lexical rules.
C. Investigate (with project A8) the domain theoretic aspects and termination of recursive type constraints with anti-reentrancies.

Linguistics	A.	Ensure linguistical adequacy of the logical approach to linear precedence rules (in collaboration with the fragment projects B8 and B9).
	B.	Ensure linguistical adequacy of our approach to lexical rules (in collaboration with the fragment projects B8 and B9).
Computation	A.	Develop compiler from implicational constraints to type definitions. This includes compiling out all negation except for anti-reentrancies.
	B.	Develop a compiler from type definitions to definite clauses.
	C.	Develop techniques for automatically (or semi-automatically) unfolding calls to typed unification, in order to isolate unifications of phonology features.
	D.	Further develop program transformation techniques in order to produce a.) efficient programs for definite clause interpreters (CUF, Troll) and b.) a phrase structure backbone.
	E.	Develop techniques to perform magic compilation on the auxiliary predicates expressed in typed feature structures.

3.5.4 Zeitplan

1995	I.	Investigate linguistic and logical aspects of LP-constraints (in collaboration with the fragment projects B8 and B9).
	II.	Investigate linguistic and logical aspects of lexical rules (in collaboration with the fragment projects B8 and B9).
	III.	Investigate (in cooperation with project A8) the domain theoretic aspects and termination of recursive type constraints with anti-reentrancies.
	IV.	Complete compiler from general implicational constraints into type definitions.
	V.	Complete Compiler from type definitions into Prolog clauses.
1996	I.	Develop logical foundations for LP-constraints (with feedback from the fragment projects B8 and B9).
	II.	Develop logical approach to lexical rules (with feedback from the fragment projects B8 and B9).
	III.	Develop fold/unfold techniques to isolate unifications of phonology features.
	IV.	Apply transformation techniques to achieve efficient reorderings of the grammar represented in definite clauses. These transformed grammars will be input to either CUF or to the definite clause interpreter of Troll or ALE.
1997	I.	Implement our treatment of LP rules.
	II.	Implement our treatment of lexical rules.
	III.	Extract phrase structure backbone from transformed grammars.
	IV.	Perform magic compilation on the remaining auxiliary predicates represented in typed feature structures.

3.6 Stellung innerhalb des Programms des Sonderforschungsbereichs

One of the functions of the B4 project will be to serve as a bridge between the two fragment projects: the syntactic fragment B8 (Hinrichs) and the semantics fragment B9 (Kamp/Rohrer). B4 serves as a bridge between the two fragment projects in that it allows the specification of their results in a very high-level, i.e. HPSG-II, grammar specification that is subsequently automatically translated into an implementable grammar. In this way, B4 allows HPSG-II grammar specification to be used as a common ground for communication between the fragment projects. The correct and efficient implementation of the specified fragments can be delegated to the compiler to be developed by the B4 project. The compiler translates an HPSG-II grammar specification into an efficient grammar program for a system specified by the user. The grammars specified by the fragment projects will be automatically compiled into an optimized definite clause representation that can be used with the CUF system in case of the B9 project or the definite clause interpreter of Troll in the case of the B8 project. Subsequently, this Prolog program is compiled into in an optimized phrase structure rule-based representation that can serve as input to Troll or ALE.

Our project seeks to benefit from the fragment projects in that they provide large, empirically and theoretically interesting grammars that can be used to test and further optimize the grammar compiler. We also expect to benefit in the same manner from interaction with the B3 project, which will also provide grammars related to the fragment grammars. Simultaneously, we expect to gain insights especially from B8 on the correct, or linguistically most adequate formulation of LP-constraints and lexical rules. Given that ideas about such devices are currently in flux, it will be important to implement formulations of them in close cooperation with the linguists such that it meets the linguist's needs.

We expect also to cooperate with the A8 Project (Mönnich) from which we expect to have input on the adequacy of our underlying formalism. In particular, the domain theoretic methods employed in that project may be applied to provide a fixed-point semantics for our system employing type definitions and anti-reentrancies. These techniques may also be used to obtain termination results for our system.

Our project would also benefit from interaction with the B7 project of Rooth/Abney (Partial Parsing and the acquisition of Lexical Syntax and Semantics) which proposes to use HPSG style feature structures in a language acquisition context. The lexical entries to be acquired by the B7 project would provide an interesting empirical basis for our investigation of the formal properties of lexical properties. Moreover, the shallow stochastic parsing techniques to be developed by B7 could complement our more fine-grained unification-based parsing techniques in interesting ways.

Literaturnachweis

Beeri, Catriel & Ramakrishnan, Raghu (1991) On the Power of Magic. Journal of Logic Programming 10(2).

Carpenter, Bob (1992). *The Logic of Typed Feature Structures.* Cambridge Tracts in Theoretical Computer Science, number 32. Cambridge University Press, Cambridge, England.

Carpenter, Bob (1993). ALE*: the Attribute Logic Engine.* User's guide. Carnegie Mellon University, Pittsburg, Pennsylvania, USA.

Dörre, Jochen & Dorna, Michael (1993) CUF - A Formalism for Linguistic Knowledge Representation. In Dörre (Ed.) Computational Aspects of Constraint-Based Linguistic Description I, DYANA-2, Deliverable R1.2.A.

Dymetman, Marc; Isabelle Pierre & Perrault, François (1990). A Symmetrical Approach to Parsing and Generation. In *Proceedings of* COLING*-90.* Helsinki, Finland.

Eisele, Andreas & Dörre, Jochen (1990). *A Comprehensive Unification Formalism.* DYANA-2, Deliverable R1.2.A.

Emele, Martin & Zajac, Rémi (1990) Typed Unification Grammars. In *Proceedings of* COLING*-90)*, Helsinki Finland.

Engelkamp, Judith; Erbach, Gregor & Uszkoreit, Hans (1992) Handling linear precedence constraints by unification. In Proceedings of the 30th annual meeting of the ACL.

Johnson, Marc & Stabler, Ed (1993) *Course Notes on Principle Based Parsing*, LSA Summer Institute.

Gerdemann, Dale (1994) Parsing as Tree Traversal. In *Proceedings of* COLING*-94*, Kyoto, Japan.

Gerdemann, Dale & Götz, Thilo (1993) *Troll Manual.* Unpublished manuscript. Eberhard-Karls-Universität, Tübingen, Germany.

Gerdemann, Dale & King, Paul (1993). Typed Feature Structures for Expressing and Implementing Feature Cooccurrence Restrictions. In *Proceedings der 4. Fachtagung der Sektion Computerlinguistik der Deutschen Gessellschaft für Sprachwissenschaft*, pages 33–39. Hamburg, Germany.

Gerdemann, Dale & King, Paul (1994). The Correct and Efficient Implementation of Appropriateness Specifications for Typed Feature Structures. In *Proceedings of* COLING-*94*. Kyoto, Japan.

Götz, Thilo (1993). *A Normal Form for Typed Feature Structures*. Magisterarbeit. Eberhard-Karls-Universität, Tübingen, Germany. Published as SFB 340 technical report, number 40. IBM Deutschland GmbH, Heidelberg, Germany.

Griffith, John (forthcoming) *Disjunction and Efficient Processing of Feature Descriptions*. Doctoral dissertation. Eberhard-Karls-Universität, Tübingen, Germany.

Hinrichs, Erhard & Nakazawa, Tsuneko (1994). Partial-VP and Split-NP Topicalization in German: An HPSG Analysis. In Hinrichs, Meurers & Nakazawa *Partial-*VP *and Split-*NP *Topicalization in German: An* HPSG *Analysis and its Implementation*. SFB 340 technical report. IBM Deutschland GmbH, Heidelberg, Germany.

Kanamori, Tadashi & Horiuchi, Kenji (1987) Constructions of Logic Programs Based on Generalized Unfold/Fold Rules. In *Proceedings of Conference on Logic Programming*. The MIT Press, Cambridge,Mass.

Kepser, Stephan (forthcoming) *A Satisfiability Algorithm for a Logic for Typed Feature Structures*. Magisterarbeit. Eberhard-Karls-Universität, Tübingen, Germany.

King, Paul (1989) *A logical formalism for head-driven phrase structure grammar*. Doctoral Dissertation. Manchester University, Manchester, England.

King, Paul (1992). Unification Grammars and Descriptiove Formalisms. Unpublished Lecture Notes. Tübingen, Germany

King, Paul (1994). Typed feature structures as descriptions. In *Proceedings of* COLING-*94*. Kyoto, Japan.

Martini, Dieter (1994). *Elimination von Linksrekursion bei Definite Clause Grammatiken*. Magisterarbeit. Eberhard-Karls-Universität, Tübingen, Germany.

Meurers, Detmar (1994). *On Implementing an HPSG theory*. In Hinrichs, Meurers & Nakazawa *Partial-*VP *and Split-*NP *Topicalization in German: An* HPSG *Analysis and its Implementation*. SFB 340 technical report. IBM Deutschland GmbH, Heidelberg, Germany.

Guido Minnen (1992). Earley-type Parsing of TAG and Factoring Linear Precedence. In *Proceedings of the 2nd* TAG+ *Workshop*. Philadelphia, Pennsylvania, USA.

Minnen, Guido (1994) *Predictive Left to Right Parsing of a Restricted Variant of* TAG(LD/LP). SFB 340 technical report, number 40. IBM Deutschland GmbH, Heidelberg, Germany.

Minnen, Guido (in preparation) Bidirectionality: Compilation Techniques for Analysis and Synthesis of Natural Language. Doctoral dissertation.

Minnen, Guido; Gerdemann, Dale & Hinrichs, Erhard (1993) Direct Automated Inversion of Logic Grammars. In Proceedings of the 4th International Workshop on Natural Language Understanding and Logic Programming, Nara, Japan.

Minnen, Guido & Gerdemann, Dale (1993) *Direct Automated Inversion of Logic Grammars.* SFB 340 technical report, number 35. IBM Deutschland GmbH, Heidelberg, Germany

Morawietz, Frank (forthcoming) *Direct Parsing of Typed* ID/LP *Unification Grammars.* Magisterarbeit. Eberhard-Karls-Universität, Tübingen, Germany.

Pereira, Fernando & Shieber, Stuart (1987). *Prolog and Natural Language Analysis.* CSLI Lecture Notes, number 10. CSLI, Stanford, California, USA.

Pereira, Fernando & Warren, David (1983) Parsing as Deduction. In *Proceedings of* ACL-*83*, Cambridge, Ma.

Pollard, Carl & Sag, Ivan (1987). *Information-based Syntax and Semantics.* CSLI Lecture Notes, number 13. CSLI, Stanford, California, USA.

Pollard, Carl & Sag, Ivan (1994) *Head-driven Phrase Structure Grammar.* University of Chicago Press, Chicago, Illinois.

Ramakrishnan, Raghu (1991) Magic Templates: a Spellbinding Approach to Logic Programs. In Kowalski & Bowen (eds.) Logic Programming-Proceedings of the 5th International Conference and Symposium, vol. 1.

Shieber, Stuart (1984). Direct Parsing of ID/LP Grammars". In *Linguistics and Philosophy*, volume 7, number 2.

Seiffert, Roland(1987) Chart-Parsing of Unification-Based Grammars with ID/LP-Rules. In Klein & van Benthem (eds.) Categories, Polymorphism & Unification. Centre for Cognitive Science, University of Edinburgh Institute for Language, Logic & Information, University of Amsterdam.

Weisweber, Wilhem & Preuß, Susanne (1992) Direct Parsing with Metarules. In

Proceedings of COLING-92, Nantes, France.

Zajac, Rémi (1992) Inheritance and Constraint-Based Grammar Formalisms. In *Computational Linguistics*, volume 18, number 2, pages 181–203.