3.1 Allgemeine Angaben zum Teilprojekt B4

3.11 Thema

Domain Specific Processing of Constraint-Based Grammars

3.12 Fachgebiet und Arbeitsrichtung

Computerlinguistik

3.13 Leiter

Prof. Dr. Erhard W. Hinrichs	Dr. Dale Gerdemann
Seminar für Sprachwissenschaft der	Seminar für Sprachwissenschaft der
Universität Tübingen	Universität Tübingen
Kl. Wilhelmstr. 113	Kl. Wilhelmstr. 113
72074 Tübingen	72074 Tübingen
Tel: 07071/29-74279	Tel: 07071/29-74967

3.14 Aktenzeichen bei bisheriger Förderung in einem anderen Verfahren der DFG

Eine bisherige Förderung in einem anderen Verfahren der DFG liegt nicht vor.

3.15 Angaben zu Versuchen und Untersuchungen

In dem Teilprojekt sind vorgesehen

Untersuchungen am Menschen	( )	ja	(x)	nein
klinische Studien im Bereich der somatischen Zell- oder Gentherapie	( )	ja	(x)	nein
Tierversuche	( )	ja	(x)	nein
gentechnologische Untersuchungen	( )	ja	(x)	nein

3.16 Bisherige und beantragte Förderung des Teilprojektes im Rahmen des Sonderforschungsbereichs

(jeweils Ergänzungsausstattung; Beträge in DM 1.000)

Haushaltsjahr	Personalkosten	Sächliche Verwaltungs-ausgaben	Investitionen	gesamt
bis 1994	498,2	18,0	24,6	540,8
1995	239,6			239,6
1996	250,8			250,8
1997	250,8			250,8
Zwischensumme	1239,4	18,0	24,6	1282,0
1998	284,1	38,5		322,6
1999	284,1	16,0		300,1
2000	284,1	16,0		300,1

3.2 Zusammenfassung

The next phase of the B4 project will design and implement a specialised computational formalism for handling linear precedence and linearisation domains, and a specialised abstract machine for processing typed feature-based linguistic grammars, into which all components of the grammar signature, including the linearisation formalism, can be compiled. The result will be a system capable of efficiently parsing with large, linearisation-based Head-driven Phrase Structure Grammars (HPSG).

In the last phase of this project, we developed the ConTroll system, a constraint-resolution system for processing with typed feature-based grammars. This system, implemented in Prolog, compiles grammars into definite clauses, which are then transformed to optimise performance relative to a mode of processing (parsing or generation) and a mode of interpretation. It was developed in tandem with the HPSG German syntax fragment of project B8, and has been tested extensively by the implementors of that HPSG grammar.

Within any large-scale implementation task, of course, the goal is to recognise background knowledge in the empirical domain which can be brought to bear early enough to refine the search for a solution. As the B8 fragment grows larger, this goal will become increasingly more vital for continuing to ensure efficient processing.

To date, however, ConTroll only uses two specifically linguistic sources of knowledge in its compilation. The first is (partial) knowledge about the logic of typed feature structures. Using the restrictions inherent to the logic of King (1989), we can partially pre-compile common operations such as unification and feature access more than would otherwise be possible. Given that these are the basic operations with which processing at every level of the system occurs, even slight improvements to their efficiency yield enormous benefits to linguist users.

The second is knowledge about lexical rules. The linguist user can define a set of lexical rules, which generalise over some set of lexical entries. This is particularly important as the size of the lexicon grows large. The meaning of these lexical rules is defined in terms of the underlying relational language of ConTroll in a straightforward way, transparent to the user. The lexical rule compiler, however, makes use of the special properties of lexical rules and compiles them into a more efficient internal representation, using finite state techniques to compute ahead of time the possible lexical rule interactions.

The next phase of the B4 project will extend this use of domain knowledge in two directions:

3.21 Linear Specification Language (LSL)

We add linear precedence and linearisation domains to the list of utilised linguistic knowledge, by designing and implementing a language just for those constraints. Specifically, LSL will be a conservative extension of phrase structure rules by a set of simple, near-atomic statements about linear precedence and domain formation, in order to be adequate for freer word-order languages. At the same time, these near-atomic statements will be capable of identifying sources of complexity and sources of deterministic information to a compiler which can translate these rules into efficient code for parsing. LSL specifications, along with the rest of the grammar, will be compiled into feature structure abstract machine code.

Linearisation grammars have recently emerged as a successor to ID/LP grammars in constraint based analyses of (semi-) free word order languages. In contrast to ID/LP grammars, there is no phrase structure skeleton on which the LP component operates. Instead, so-called linearisation domains form the basic units for determining word order, and it is these that LP constraints are defined on. Local phrase structure trees and linearisation domains do not necessarily coincide. Meurers et al. (1997) for example propose a grammar of German where the Mittelfeld has a binary branching phrase structure, yet the whole Mittelfeld is one linearisation domain. Word order in the Mittelfeld is thus totally independent from phrase structure.

Linearisation in the B8 fragment grammar was, in fact, a significant source of complexity for parsing in the last phase; and it is about to become much worse. The combination of having a linear specification component, and having an abstract machine into which they can be compiled will allow us to handle this complexity. In the next phase, the A3 and B8 projects will together consider phenomena which will present two sources, in addition to the Mittelfeld, of linearised freedom. The first is parentheticals. To date, there is no known analysis of parentheticals in German which reduces the problem to a context-free grammar; and it is unlikely that such an analysis will emerge, particularly one where the phrase structure rules have any natural correspondence to the assemblage of a semantic interpretation. Parentheticals are, at once, very free in their position, but not totally unconstrained, so that their position cannot simply be ignored.

The second source are data often analysed as cases of extraposition to the Nachfeld. These are found both in the verbal complex (e.g. incoherent verbal complements, non-verbal elements of the verbal complex), and at the clausal level of structure (e.g.

verb-second complement clauses, free and complement daß clauses). Not only does a proper treatment of extraposition require a linearisation-based analysis, but arguably requires the extra notion of *liberation*, the ability for some constituents to break away from their otherwise cohesive domain objects.

3.22 Feature Structure Abstract Machine (FSAM)

We push our use of logical knowledge even further by creating an *abstract machine* designed expressly for feature-based parsing of natural language grammars. The efficiency gain derived from these machines in all components of the grammar is an essential part of coping with the complexity added by the existence of linearisation components. Abstract machines are idealised machines commonly used in logic programming compilation in order to separate principled, platform-independent, compilation techniques from lower-level optimisations and computer architecture-dependent decisions. An abstract machine consists of all of the components of a computer: memory module, control module, instruction set, registers, etc., which are ideally suited to the compilation of a particular programming language. Abstract machine code can be further compiled at a later stage into machine-dependent code, thus serving as a convenient intermediate conceptual stage, as well as a convenient state in which to reason about and enhance the performance of a user's code at a lower level. The use of an FSAM will allow us to pre-compile type specifications, constraints and also expressions from LSL more extensively and more efficiently than would be possible in a Prolog-based implementation.

Previous logic programming systems were simply designed for less specialised logics, with either fixed arity terms or attributed structures with potentially infinite arities. The use of *appropriateness conditions*, declarations which specify which types can have values for which features, has consistently proven to be of a practical advantage to both ConTroll and ALE, which use very similar logics. Appropriateness allows these systems to perform a very strong kind of inferencing on descriptions over feature structures, which brings type and value information to bear as soon as possible in order to force unification failures and thus prune unnecessary search. These benefits accrue, furthermore, regardless of whether the description is part of a linear precedence constraint, a phrase structure rule, or a lexical item.

Currently, ConTroll and ALE, both of which are implemented in Prolog, place a layer of extra inferencing and data structures on top of Prolog's fixed arity logic in order to simulate this reasoning. FSAM data structures will purposefully be designed to suit the needs of an appropriateness-endowed feature logic, minimising the overhead of every unification operation performed in every component of the grammar. Recent benchmarks on ALE show that the system spends over 90% of its time simply performing unifications.

With an abstract machine level at our disposal, other more familiar concerns take on a very different form. Parsing control, for example, is typically regarded as consisting of several basic operations, such as the selection of a rule, matching a linguistic token from a chart or agenda against a rule's token, and a variety of prediction steps. This is essentially a vestige of the days when tokens were just that: NP, VP, S, etc. Matching a token consumes a significant amount of resources when the tokens are feature structures with over 200 nodes, all of which need to be unified. In this case, the traditional division of labour simply is not appropriate to the task. At the level of the FSAM, unification operations occur side-by-side with low-level control operations, allowing a much tighter, specialised interleaving to suit feature-based processing with various optimisation techniques similar to those other logic programming languages' abstract machines. Research from generation, the treatment of empty categories, co-routining for constraint resolution, and, of course, lexical rules and linear precedence constraints can be utilised at this level for the typed feature-based case.

At the FSAM level, we can also provide enhancements and implement alternative processing methods more sensibly than would otherwise have been possible, both more general logic programming enhancements and typed feature structure specific ones. We can make backtracking more efficient because we have direct access to the memory where the results of that work are recorded. With the liberty to create more specialised data structures, we can encode types and features as bit vectors to make unification much more efficient. We can use specialised indexing strategies for access to parsing charts and the lexicon. We can also begin to reconsider seriously the possibility of parallel algorithms at all levels of processing. These enhancements are described in detail in Section 3.52.

A collection of LSL specifications will be the core input to our FSAM compiler, carrying with it procedural and constraint attachments. The combination of a specialised linearisation language (which, in turn, permits specialised compilation) and lower-level logical optimisations in the FSAM will allow us to provide efficient parsing for large-scale grammars, including the HPSG German fragment being implemented by project B8. The B4 and B8 projects will continue their close cooperation by using the fragment grammar as a comprehensive test case of the adequacy of the LSL formalism, as well as of the performance of the FSAM.

3.3 Stand der Forschung

3.31 Linearisation grammars

In Generalised Phrase Structure Grammar (GPSG, (Gazdar et al. 1985)), the notion of phrase structure grammar as a set of phrase structure rules was generalised to a set of Immediate Dominance (ID) and Linear Precedence (LP) rules. The ID/LP format allows the succinct description of constituent order variations of sister nodes in a local tree.

The ID/LP approach was in turn generalised by Zwicky (1986). Zwicky had the idea of *liberating* nodes from a local tree to allow them to be ordered at a higher level. His approach worked on a rule by rule basis. Rules were either ordinary ID-rules, or they were marked as liberating their daughter constituents. Dowty (1989) took up

Zwicky's ideas, but made a careful distinction between *tectogrammar* (the rules that license linguistic structures) and *phenogrammar* (how the strings of substructures are combined when a rule is applied). This is not a useful distinction for a phrase structure or ID/LP grammar, since the only string combination operation for those kinds of grammars is concatenation. This distinction has also been successfully applied to a linearisation-based Serbo-Croatian HPSG by Penn (1997).

Dowty also added the notion of *bounding categories* to his system. Bounding categories do not allow their strings to merge with outside string material. Such restrictions can not be expressed in Zwicky's framework, since liberation there works on a rule-to-rule basis. Finally, Dowty introduced the distinction between *precedence* and *attachment*. Precedence constraints just require one node to precede the other, but other material may intervene. Attachment requires the yield of some node to *immediately* precede some other, so that no intervening material is allowed.

Within the HPSG community, linearisation grammars have recently emerged as a successor to ID/LP grammars for the analysis of (semi-) free word order languages. The ideas of this line of research are based on the work of Reape (1989, 1992), who replaced the phrase structure based notions of ID/LP grammars by so-called word-order domains. The extension to ID/LP grammars consists mainly in the fact that word-order domains are a non-hierarchical level of linguistic structure that allow non-sister constituents to be ordered. (Reape 1994) showed how word-order domains can be applied to an HPSG analysis of German clause structure, the verb cluster, and scrambling in the Mittelfeld.

(Pollard et al. 1993) build on the notions developed by Zwicky, Dowty and Reape. They propose to build a HPSG theory of linearisation and apply it to a wide range of data, including German extraposition phenomena. We will cooperate with the research group at the Ohio State University to ensure that our linearisation specification language is powerful enough to express the necessary constraints for the analyses developed by that group.

(Kasper et al. 1995) showed that the formal integration of word-order domains into HPSG theory was a non-trivial exercise. It is particularly the presence of structure sharing information and calls to relational goals in linear precedence constraints that pose difficult formal (and thus, computational) problems.

Decisive progress in terms of formal specification and empirical adequacy of HPSG linearisation grammars was made in the context of the B8 project. (Richter and Sailer 1995) shows how a theory of linearisation can be correctly formalised in SRL (Speciate Re-entrant Logic, (King 1994)), a logic specifically designed to be the formal basis of HPSG, which is also the basis of ConTroll. This theoretical proposal was then implemented in the linearisation component of the fragment grammar.

A different approach was taken by Manandhar (1995b), who showed that the feature logic of set descriptions (Manandhar 1993; Manandhar 1995a) can be used to encode linear precedence constraints over linearisation domains. The basic idea is for the user to provide a relational "feature," *prec*, which holds over two nodes, *a prec b*, provided the yield of *a* precedes the yield of *b*. With set descriptions, it provides two basic additional constraints: *domain inclusion*, in which one includes one feature structure's yield into another's, and *domain precedence*, in which two set-valued feature values stand in the *prec* relation. Notice that this work does not provide a theory of linearisation, but just the means in which to express some theories.

3.32 Compilation

The first computational generalisation of ID/LP grammars was an implementation which incorporates LP constraints into lexical entries (Engelkamp et al. 1992). The approach is interesting in that it allows the ordering of non-sister constituents. Engelkamp et al. (1992) give an example of a binary branching analysis of the German Mittelfeld, where it is still possible to express LP constraints on the complements and adjuncts. This is achieved by extending the ordering domains from local trees to so-called head-domains (the head and its complements and adjuncts).

In the first phase of the B4 project, Morawietz (1994) showed how standard ID/LP grammars could be implemented efficiently by using delaying techniques for evaluating LP statements. The implementation was integrated with the Troll system (Gerdemann et al. 1995).

The only implementation of a linearisation grammar other than the B8 Fragmentgrammatik that we are aware of was written in the context of the RGR (Reusabiliy of Grammatical Resources, (Erbach et al. 1995)) project at the Universities of Saarbrücken and Edinburgh. The implementation is a combination of a constraint solver for (Manandhar 1995b) and the ProFIT system (Erbach 1994). Preliminary results with an experimental linearisation grammar, however, indicated slow run-time performance.

Our approach is distinguished from this in that linearisation statements from our specification language operate over the yields of phrase-structure-like trees, not over sets within the feature logic. The goal of Manandhar (1995b) was to provide a generalisation of feature logic itself to allow a declarative expression of linear precedence constraints. We are providing a means for *processing* linear precedence at the level of derivation trees to ensure efficiency, at the price of departing from the logical framework of HPSG itself. In this respect, our language falls between Manandhar (1995b) and the rules used as input to the "head-corner" parser of van Noord (1991). This parser uses a head-driven, rather than string-position-driven approach to parsing, combined with a more general means of phonological combination, in order to capture discontinuous constructions. Unlike these rules, ours abstract away from data structures and string operations, and will handle a much wider range of linearisation-constrained theories.

Processing linear precedence with a set-augmented feature logic presents some challenges which we can easily avoid in our approach to linearisation and abstract machines. First, representing linear precedence as a feature, and order-underspecified linear domains as set values within the feature logic, involves linear order and domains in the operation of feature structure unification. In the case of feature-based linear precedence, one can only encode non-immediate precedence by adding functional uncertainty over feature paths to the description language. Although no proof yet exists, it is generally believed that this significantly extends the expressive power of the description language (e.g. it is possible to describe all and only lists of finite length), as well as the computational cost. In the case of sets, the only known implementation of this logic requires an all-pairs check over the number of sets in order to decide when to enforce set unification. Even then, there is an additional non-deterministic cost in how to combine two sets' values consistently. In our approach, domains are manipulated outside the feature structures themselves; and the unification of two domains is never required.

Second, as noted by Manandhar (1995b), immediate precedence can be difficult to enforce in a system which requires linearised models. For example, if $<<$ represents immediate precedence, then the expression:

$$(a << b) \wedge (c << d) \wedge (a < d) \wedge (c < b)$$

actually has a solution, in which a is unified with c and b is unified with d. In general, the most natural interpretation of feature-based set logic without arity constraints entails that a set with two elements, a, b, subsumes a set with one element, $a \wedge b$. The phonological yields we manipulate in our approach exist in a logic over derivation trees in which there is no unification of domain elements. In this respect, we are incorporating the linguistic knowledge that words in a string can automatically be taken as distinct. To do this in a feature logic would require the explicit addition of pairwise inequational constraints, such as ALE provides Carpenter and Penn (1994).

By using linearisation constraints as annotations on phrase-structure-like rules, we are, at the same time, delineating a family of explicit control strategies, namely generalisations of the familiar parsing algorithms for context-free languages (e.g. Earley deduction, chart parsing, "head-corner" parsing), which, at any rate, bound search in that they specify linear precedence relative to specific statements about immediate dominance. Previous approaches have either required the user to use genuine phrase structure rules, or provided no control strategy whatsoever, simply a declarative specification. An example of the former would be the familiar parsing algorithms themselves; and, of the latter, feature-based precedence with set descriptions. In the latter case, of course, such declarations can be used inside phrase structure rules whose categories themselves are feature structures; but in this case, linearisation information is merely one undistinguished source among many at the level of feature structure unification, and does not figure in the selection of control strategy for its rules or their compilation. These two must be interleaved to ensure efficient processing — precisely the sort of tight interleaving which can be achieved at the abstract machine level. One possibility we will consider is a weaker distinction of ID/LP information in the grammar signature relative to other grammatical principles, which can be compiled into our linear specification language.

This annotation method can also be distinguished from the context-free extraction and annotation method of Briscoe and Carroll (1993), which has been implemented in their Alvey Natural Language Tools. There, an atomic-category context-free backbone is extracted from a feature-based phrase structure grammar. The categories are drawn from the largest maximally specific set of disjoint descriptions which cover the descriptions used in the feature-based grammar. The remaining unifications required by the feature-based grammar are "attached" to this atomic backbone. These unifications are postponed even if they are completely deterministic, and cheap ways of constraining search, the reason simply being that LR(1) compilation, the technique they use on these atomic rules, has never been extended to the case of feature-based rules. In our approach, all feature unifications are still part of the categories in the phrase-structure-like rules, and are treated as part of them during compilation. Our feature structure abstract machine allows us to achieve this interleaving without resorting to common, but less well-suited devices. The attachments consist only of linearisation constraints and possibly procedural attachments.

3.33 Feature Structure Abstract Machines

The appeal of the FSAM is that the very logical restrictions which linguists need in order to refine their theoretical models, are exactly those restrictions which implementors need to extend the WAM conservatively enough to preserve efficiency. To date, no feature-based implementation has done that, either because their logics were concerned with a broader, or different class of applications than linguistic ones, or because their implementation language would not allow them.

The history of Feature Structure Abstract Machines (FSAMs), thus, actually begins with the Warren Abstract Machine (WAM)(Warren 1983) itself. Prolog was originally designed for natural language processing, but with first-order terms; and the WAM was designed as a convenient, platform-independent intermediate stage of compilation for efficient search with Prolog programs. The WAM provides many of the basic ingredients required for processing logic programs of any kind, e.g. heap data structures for partially instantiated terms (such as Prolog terms with variables), stack and call-frame management for passing arguments and handling disjunction, and trailing for backtracking on unification. All of this must be used by a FSAM as well; and much of it can be adopted without change.

Along with the WAM comes over ten years' worth of refinements to it, many of which have direct application to a FSAM, or can easily be adapted. These include optimisations such as creating separate streams for read and write-mode unification (Amraoui 1988; Meier 1990; Mariën and Demoen 1991), improved register allocation algorithms (Mellish 1985; Debray 1986; Janssens, Demoen, and Mariën 1988), and optimised trailing. They also include extensions of the WAM to coroutining (Carlsson 1987; Aggoun, Chan, Dufresne, Falvey, Grant, Herold, Macartney, Meier, Miller, Mudambi, Perez, van Rossum, Schimpf, Tsahageas, and de Villeneuve 1995) and constraint solving (Carlsson 1991; Holzbaur 1995; Diaz and Codognet 1993) which are related to our proposed treatment of linearisation constraints.

Prolog, however, presents a number of disadvantages for feature-based processing.

First, its values are positionally encoded, rather than named. Second, its correlate to types, predicate names, are fixed arity terms which unify only with variables and themselves. In the logic of typed feature structures, this would correspond to a type hierarchy with a most general type, which immediately subsumes an infinite number of incomparable types. It is possible to encode typed feature structures into Prolog terms in a manner which preserves subsumption, inheritance and unification. Gerdemann (1995), in fact, shows that this can be done without sacrificing multiple inheritance or value refinement constraints, which all term-encoding-based systems, e.g. ProFIT (Erbach 1994), still do. For large feature structures and large-scale applications, however, these are simply too large and awkward to be practical.
Many feature-based implementations (e.g. ALE (Carpenter and Penn 1994), ConTroll (Götz 1997), Troll (Gerdemann, Götz, Griffith, Kepser, and Morawietz 1995)) use Prolog itself as an intermediate code for compilation, by generating Prolog code from type signatures and grammars which is then compiled further by a Prolog compiler. Prolog also presents a number of disadvantages as an intermediate code. It does not directly provide destructive operations, or provide direct access to pointers, heap structures, control, selective copying mechanisms etc. in order to use restrictions such as appropriateness, declarations which specify which types can have values for which features, to their fullest.
The first truly feature-based abstract machine was the one used for the LIFE logic programming language (Aït-Kaci and DiCosmo 1993). LIFE provides an attributed record structure, called an order-sorted feature term (OSF), with the following constraints:

- sort constraints, which add a type to a feature structure,
- feature constraints, which add a feature and a value for it to a feature structure, and
- arity constraints, which fix the range of possible features for which values can be added to a given set of features.

OSF constraints are fundamentally different from HPSG's notion of appropriateness in two ways.
First, feature constraints are independent from sort constraints. In LIFE, an OSF term of any type can, in principle, take a value for any feature in spite of a sort constraint. In HPSG, appropriateness is used to decide which features must have values, and as a function of type. This interpretation is called *totality*.
Second, arity constraints are independent from sort constraints. When an arity constraint is placed on an OSF term, that feature structure may only take values for features in the set specified by that constraint, no matter what its type may be promoted to later. In HPSG, every type implicitly has such an arity constraint, as provided by appropriateness. This interpretation is called *well-typing*. In HPSG, however, well-typing only fixes arity insofar as a feature structure of a given non-maximal type cannot have any additional features except those provided by its subtypes. The

bound on arity is, thus, interpreted with respect to the position of the type in the inheritance hierarchy. Totality plus well-typing is known as *total well-typing.*

(Aït-Kaci and DiCosmo 1993) generalised the WAM to handle named attributes, sort hierarchies and feature inheritance, but not arity constraints. It is a very clever extension, which ameliorates much of the generality of OSF terms, but there are certain inescapable costs for processing with it. The names of attributes, for instance, must be stored with a structure's representation, and searched at run-time, because one cannot predict at compile-time which features will necessarily be present (even with arity constraints, in the general case). Roughly the same language (including arity constraints) has also been incorporated into the abstract machine of Oz, AMOZ (Mehl, Scheidhauer, and Schulte 1995; van Roy, Mehl, and Scheidhauer 1995).

SICStus Prolog has an attributed variables library based on (Holzbaur 1990). This allows one to associate a fixed number of values with any attribute on an unbound variable, along with hook predicates for unification, goal conversion and constraint projection, for use in wide range of coroutining and constraint solving applications. The use of unbound variables allows one to carry out the destructive operations needed for efficient constraint-based implementations using chains of updates in a non-destructive logic programming language. This library, thus, permits a very smooth integration of attribute values with Prolog, and, in particular, SICStus's abstract machine. One can also use its hook predicates to simulate appropriateness and total well-typing, in which case this library would simply carry out the work that systems like ALE and ConTroll perform on top of Prolog, only internally to SICStus's own compilation and execution. There is still no built-in notion of appropriateness, or for that matter, typing, at the abstract machine level.

For large-scale grammars and applications based on typed feature structures, such as the B8 fragment grammar, the prediction of feature presence with type-based declarations that can be compiled off-line is absolutely critical for efficient processing. Each component of the grammar assembled thus far uses this information to the extent that ConTroll allows; and as the fragment grows larger and the search space expands, the effective use of this information will become even more vital. In the B12 project, which is based on LFG, a grammatical theory with no feature-bearing types and with a phrase structure backbone, a *de facto* notion of appropriateness (i.e. restrictions on feature co-occurrence) is also utilised in order to perform efficient unification. Of course, for HPSG, total well-typing also refines the logic so that well-formed objects in the model always correspond to linguistic entities: every object, e.g. a *sign*, has a value for all and only the attributes which it should have, e.g. PHONOLOGY, SUBCATegorisation frames etc.

3.4 Eigene Vorarbeiten

3.41 Linearisation grammars

To the best of our knowledge, the Fragmentgrammatik of B8 implemented in ConTroll is the only large, working linearisation grammar. The reader is referred to the Ergebnisberichte for the most recent phase of both projects for more details.

3.42 Compilation

The first phase of the B4 project developed Troll (Gerdemann and King 1994; Gerdemann, Götz, Griffith, Kepser, and Morawietz 1995), a parser and logic programming system based on HPSG's typed feature structures, as formalised by (King 1989).[1] This system compiles type signatures and appropriateness declarations off-line into efficient Prolog code for unification of feature structures, which is then compiled further by a Prolog compiler. In the same manner, it also compiles phrase structure rules whose categories are feature structure descriptions, and expressions from a feature structure-based procedural attachment language into Prolog code. Griffith (forthcoming) also provides a means for compiling named disjunctions.
ALE (Carpenter 1992a; Carpenter and Penn 1994) has essentially the same functionality as Troll, only based on Carpenter's logic of typed feature structures (Carpenter 1992b) rather than King's. It also has a small constraint resolver, and supports inequational descriptions and extensional typing.
The more recent phase of the B4 project concentrated on HPSG constraint resolution. The ConTroll system (Götz 1997) permits the user to specify grammars directly with feature-based implicational constraints and functional attachments, augmented with delay statements. It transforms these constraints into expressions of a feature-based logic programming language, which are then further transformed into efficient parsing code. ConTroll also has a graphical development and debugging environment, and has been extensively used in the development of the B8 fragment grammar.[2]
The present proposal represents a novel compromise of these approaches. A phrase-structure-like backbone is provided, but with generalised linear precedence, provided by a special declarative constraint language. All other principles can be provided either as implicational constraints or relational attachments. In this way, we hope to capture the efficiency which more rule-based approaches provide, while still retaining the convenience and modularity of a constraint-based system.
We will also explore an even more constraint-based extension of this, whereby even the ID/LP principles of a grammar can be encoded with constraints, provided that they are distinguished as being ID/LP related. This is an extension of Aldag (1997), who, in the most recent phase of the B4 project, provided such a translation from

[1] The reader is referred to the Ergebnisbericht of the first phase of B4 for a complete description.
[2] The reader is referred to the Ergebnisberichte of the second phase of B4 and the first phase of B8 for a complete description.

constraints in the case where there no procedural attachments, and where the ID/LP principles were strong enough to render the induced language context-free.

3.43 Feature Structure Abstract Machines

(Penn 1993) suggested that the WAM could be adapted to Carpenter's logic of typed feature structures, which has the same interpretation of appropriateness as HPSG's. Typed feature structures fall somewhere between Prolog terms and the more general record structures discussed in Section 3.33. On the one hand, they allow named attributes, and their strong typing permits the user to refer to feature structures by means of a language of partial descriptions which makes specifications much easier to work with. They also have inheritance-based polymorphism, and can add features as their types are refined, i.e. they are not fixed-arity terms. On the other hand, for any fixed type, all structures of that type have the same arity because of appropriateness, which allows for a positional encoding of structures (although still named), random access to feature values and more efficient unification at the abstract machine level, much like the WAM.

(Wintner and Francez 1994), reporting on work developed in collaboration with the B4 project, presented an abstract machine instruction set for unification over Carpenter's logic, which uses appropriateness to provide a very efficient, conservative extension of heap data structures for the WAM to the attributed case. These instructions are low level operations involved in every feature-based processing task, such as loading a register with a structure's value for a particular feature, adding a type to a feature structure, unifying two feature structures relative to a type signature, or adding a chart edge to a parsing chart. This allows one to abstract away from the machine or language-dependent considerations involved with performing any one of these tasks, while still being able to refer to operations at this low level.

AMALIA also uses a lazy enforcement strategy for total well-typing, which reduces some of the cost of enforcing it with large structures, while still retaining the benefits described above. It was also the first to adapt a FSAM to feature-based chart parsing, by representing rules with *multi-rooted structures*, an extension of OSF terms without arity constraints. (Wintner 1997) describes this in more detail, along with a complete implementation, AMALIA, which includes an FSAM-level debugger with a graphical user interface. These two contributions, appropriateness-based logical compilation and chart parsing, are the core around which we intend to make a series of domain-specific optimisations in the next phase of the B4 project. Our experience with AMALIA has already confirmed the value of these contributions, with parsing performance over 10 to 15 word sentences already averaging a factor 15 improvement. The chart parser, however, works with traditional context-free rules, and has no facility for linearisation constraints, general implicational constraints, or relational attachments. Its unifier and chart parser can also be optimised much further.

(Qu 1994) presented a slightly different abstract machine architecture for unification, which has only been partially implemented. (Penn in prep.(b)) will extend the architecture of (Qu 1994) to cover the full expressive power of ALE or Troll, plus provide methods for empirically refining FSAM data structures over large corpora. In the most recent phase of the project, (Zahnert 1997) implemented an abstract machine for ConTroll's feature-based logic programming language, including unification operations for feature structures. It also implements ConTroll's coroutining mechanism. Unlike AMALIA and (Qu 1994), the abstract machine code generated by this compiler is not interpreted at run-time, but rather compiled straight into Objective C, which can then be compiled by a C compiler such as `gcc`. This strategy produces more efficient, very portable run-time code, which builds on dynamic memory and stack management techniques already extensively developed for C. This system is merely a feature-based logic programming system, however.

3.5 Ziele, Methoden, Arbeitsprogramm und Zeitplan

3.51 Ziele

The next phase of the B4 project will have these goals:

1. Design a specialised *linear specification language* (LSL) for linearisation grammars in general and the B8 fragment grammar in particular.

2. Extend the AMALIA system to a *feature structure abstract machine* (FSAM) for typed feature structure grammars. This will include support for:

    - basic data structures and control techniques for a feature-based logic programming language
    - coroutining
    - optimised chart-parsing
    - general implicational constraint solving
    - training code for statistically optimising the abstract machine relative to a fixed grammar
    - parallel processing

3. Design and implement a compiler from ConTroll grammars, including LSL specifications, to FSAM code. This will include:

    - extensive type and appropriateness compilation
    - interleaving LSL constraint propogation with traditional head-driven chart parsing to process linearisation grammars efficiently
    - discovering near-optimal feature structure indexings relative to fixed grammars
    - logic-based grammar transformation for improved efficiency

- abstract machine code transformation for parallel processing

4. Integrate the existing graphical user-interface with the new system

The result will be a feature-based grammar development system which is capable of efficient parsing with the B8 fragment's linearisation-intensive grammar and which sets a new standard for efficient feature-based parsing.
We will also investigate the possibility of compiling constraint-based grammars whose ID- and LP-related constraints are specially marked by the user into equivalent grammars with LSL specifications.

3.52 Methoden und Arbeitsprogramm

Linearisation Specification Language

The Linearisation Specification Language (LSL) will incorporate the basic insights of Zwicky (1986), Dowty (1989), and Reape (1994). LSL will have the following properties:

- Support efficient parsing algorithms.
- Provide expressive power to support the analyses of Zwicky (1986), Dowty (1989), Engelkamp et al. (1992), Reape (1994), Pollard et al. (1993), and Richter and Sailer (1995).
- Have a declarative semantics.

LSL will be an extended phrase-structure-based formalism for describing the composition of constituents with linearisation. We envisage this formalism as having the form of a sequence of phrase-structure-like rules, augmented with an inventory of basic expressions for describing linear precedence constraints in natural language, composable with conjunction and disjunction.
In general, the essential tension which will shape the final structure of this language is between the need for sufficient generality to express realistic linearisation grammars on the one hand, and the need for computational tractability on the other. Although phrase-structure-based, our rule system will provide powerful linearisation constraints, thus making the transition from purely relational and constraint-based descriptions into this language relatively easy. It will thus subsume ID/LP and conventional phrase structure grammars as well. The specification of linearisation constraints relative to particular rules, however, provides an inventory of subtrees to constrain, provides a backbone along which linearisation constraints can be propagated for efficiency, and also significantly restrictions the range over which the constraints need to be considered. In particular, although constraints can, to a certain extent, be propagated up and down a derivation tree, they cannot be propagated across to sister trees. This small amount of extra structure can have large consequences for constraining the search for heads of domains or constituents, for example.

$Yield:$	subtree yield
$v,\ v \in V_y$	variable yield
$w,\ w \in Phon \cup \{\epsilon\}$	phonological yield
$Desc:$	feature term
$v,\ v \in V_d$	description variable
$\sigma,\ \sigma \in Type$	type
$f : d,\ d \in Desc$	feature selection
$d_1 \wedge d_2,\ d_1, d_2 \in Desc$	conjunction
$\neg\, d,\ d \in Desc$	negation
$Rule:$	phrase structure rule
$\exists y_0, \ldots, y_n \in Yield : y_0 \to y_1, \ldots, y_n.c,\ c \in Cons$	
$Cons:$	constraint
$\top$	truth
$y_1 < y_2,\ y_1, y_2 \in Yield$	precedence
$y_1 \ll y_2,\ y_1, y_2 \in Yield$	immediate precedence
$y \sim d,\ y \in Yield,\ d \in Desc$	root description
$\langle y_1 \rangle\, (: y_2),\ y_1, y_2 \in Yield$	isolation (and liberation)
$y_1 = y_2,\ y_1, y_2 \in Yield$	equality
$y_1 \neq y_2,\ y_1, y_2 \in Yield$	inequality
$\xrightarrow{n} y,\ y \in Yield, n \in \mathcal{N}$	nth from left
$\xleftarrow{n} y,\ y \in Yield, n \in \mathcal{N}$	nth from right
$\forall y_1 \in y_2.c,\ y_2 \in Yield, c \in Cons$	universal quantification
$\exists y_1 \in y_2.c,\ y_2 \in Yield, c \in Cons$	membership
$\exists y_1, \ldots, y_n \in Yield :$ $y_0 \asymp \{y_1 \ldots y_n\}.c,\ y_0 \in Yield,\ c \in Cons$	decomposition
$c_1 \wedge c_2,\ c_1, c_2 \in Cons$	conjunction
$c_1 \vee c_2,\ c_1, c_2 \in Cons$	disjunction

Abbildung 1: Syntax of LSL (preliminary)

Figure 1 shows a preliminary proposal for how such a language could look. While the details of this language are, of course, very tentative at this point, it must crucially support the notions of liberation (Zwicky 1986), isolation (called *bounding* by Dowty (1989)), linear precedence and immediate precedence (called *attachment* by Dowty (1989)), and constraints such "nth from the left (or right)" (Dowty 1989).

Notice that LSL does not explicitly allow universal LP statements such as $A \prec B$. These statements actually have a number of possible interpretations; but the common ones do have correlates in LSL, and these can be placed on every rule in the grammar

to achieve the same effect.
A few examples will illustrate the expressive power of LSL. Suppose we would like to encode the context-free rule $A \rightarrow B\ C\ B$. We could express this as (without explicit quantification)

$$
\begin{aligned}
& y_0 \rightarrow y_1\ y_2\ y_3. \\
& \qquad y_0 \sim A \ \wedge \ y_1 \sim B \ \wedge \ y_2 \sim C \ \wedge \ y_3 \sim B \wedge \\
& \qquad y_1 \ll y_2 \ \wedge \ y_2 \ll y3
\end{aligned}
$$

This rule licenses a local tree with 3 daughters, where the mother is an A, the daughters are a B, a C and a B, respectively, the y_1 daughter immediately precedes the y_2 daughter, and the y_2 daughter immediately precedes the y_3 daughter. The first thing to note is that our precedence rules operate on tree nodes, and do not express constraints like "all As must precede all Bs". Our next example will show how this can be done. The other important thing to note is that this is a rule that cannot be expressed in an ID/LP grammar, since it violates the Exhaustive Constant Partial Ordering (ECPO) property (Gazdar, Klein, Pullum, and Sag 1985). We reject the idea of making the ECPO property part of the grammar formalism. The grammar writers should be free to enforce it if they want.
A more linguistic example shows a (much simplified) treatment of Dutch cross serial dependencies. Consider the following rules.

$$
\begin{aligned}
& y_0 \rightarrow y_1\ y_2\ y_3. \\
& \qquad y_0 \sim S \ \wedge \ y_3 \sim S \ \wedge \ y_1 \sim NP \ \wedge \ y_2 \sim V \ \wedge \\
& \qquad y_1 < y_2 \ \wedge \ y_1 < y_3 \wedge \\
& \qquad \forall x \in y_3.\ ((x \sim NP) \rightarrow (x < y_2)) \wedge \\
& \qquad \forall x \in y_3.\ ((x \sim V) \rightarrow (y_2 < x))
\end{aligned}
$$

$$
\begin{aligned}
& y_0 \rightarrow y_1. \\
& \qquad y_0 \sim S \ \wedge \ y_1 = \epsilon
\end{aligned}
$$

The first rule describes the recursive construction of cross serial dependencies, where a NP y_1 and a verb y_2 are added to a series of NPs and verbs y_3. The third line says that the new NP must precede all other material. The fourth line says that the new verb must follow all NPs in y_3, and the final line requires the new verb to precede all other verbs. Notice that this is much stronger than standard ID/LP format, where one could require all NPs to precede all verbs, but not in which order. The second rule just provides the base case to recursive definition.
As a final example, consider the following set of rules. It encodes the optional right extraposition of $\bar{S}$ complements of nouns in German.

1) $y_0 \rightarrow y_1\ y_2.$

$$
\begin{aligned}
& y_0 \sim NP \ \wedge \ y_1 \sim \bar{N} \ \wedge \ y_2 \sim \bar{S} \wedge \\
& \stackrel{1}{\leftarrow} y_2 \wedge
\end{aligned}
$$

$$(\langle y_0 \rangle \ \vee \ \langle y_0 \rangle \ : y_2)$$

2) $y_0 \rightarrow y_1 \ y_2.$

$$y_0 \sim S \ \wedge \ y_1 \sim NP \ \wedge \ y_2 \sim VP$$

3) $y_0 \rightarrow y_1 \ y_2.$

$$y_0 \sim VP \ \wedge \ y_1 \sim V \ \wedge \ y_2 \sim NP$$

4) $y_0 \rightarrow y_1 \ y_2.$

$$\begin{aligned} & y_0 \sim \bar{S} \ \wedge \ y_1 \sim Comp \ \wedge \ y_2 \sim S \ \wedge \ y_1 \ll y_2 \ \wedge \\ & \exists x \in y_2.\ x \sim V \ \wedge \ (\ (\exists y \in y_2.\ y \sim \bar{S} \wedge \ \overset{2}{\leftarrow} x) \vee \\ & \qquad\qquad\qquad\qquad (\forall y \in y_2.\ y \sim \neg \bar{S} \wedge \ \overset{1}{\leftarrow} x)\ ) \end{aligned}$$

5) $y_0 \rightarrow y_1.$

$$\begin{aligned} & y_0 \sim \bar{S} \ \wedge \ y_1 \sim S \ \wedge \\ & \exists y \in y_1.\ y \sim V \ \wedge \ \overset{2}{\rightarrow} y \end{aligned}$$

Rule 1) combines an $\bar{N}$ with its $\bar{S}$ complement. The $\bar{S}$ must be the rightmost element in its domain. However, what its domain is remains underspecified at this point. If the whole NP is isolated ($\langle y_0 \rangle$), then the $\bar{S}$ must be adjacent to the $\bar{N}$. Otherwise, only the $\bar{N}$ is isolated, and the complement sentence liberated. This has the effect that whichever domain the $\bar{S}$ finally ends up in, it will be realised as the rightmost constituent. Rule 2) combines a VP with its subject, and rule 3) a verb with its object (we only treat transitive verbs for the purposes of this example). Notice that neither rule specifies anything about linear precedence. Rule 4) combines a S with a complementiser to give an $\bar{S}$. The complementiser must immediately precede the S. The rest of the rule says that if there is a free $\bar{S}$ floating around, then the verb takes second but last position. The $\bar{S}$ comes last. Otherwise, there is no $\bar{S}$, and the verb comes last. Rule 4) finally licenses V2 clauses.
We will also provide a logical semantics for our language, along with an analysis of its formal language properties.
It is interesting to note how our approach to linearisation is different from the one of Manandhar (1995b). The main difference is that Manandhar (1995b) includes domain objects in the feature logic, rather than at a separate level. This is also the approach we took in the last phase, which turned out to be simply too slow for grammars of the size we processed. Instead of providing a completely general linear precedence module, we propose to integrate linearisation constraints with a phrase structure backbone, thus providing a basis for efficient processing. In fact, the language we propose is designed for parsing; although, because of its declarative definition, we expect it to be easily adaptable to generation.
On the other hand, the feature logic is still available for other non-linearised dependencies such as a SLASH-passing mechanism, quantifier storage, argument raising etc. Each node in a derivation tree produced by these rules is a typed feature struc-

ture, over which principles both inside and outside the logic operate.

The Manandhar (1995b) approach also provides no primitive for liberation, a notion built into our language. In both cases (linear precedence and liberation), we provide special primitives for these outside the logic of typed feature structures. This will allow us to provide special run-time support for these computationally difficult notions.

By providing linearisation constraints on tree nodes, we solve another problem of (Manandhar 1995b): we have no problem with linearisable models and thus, immediate precedence. This is simply a consequence of the fact that we impose a phrase structure backbone, thus enforcing linearised models.

The interpretation of the phrase structure-like rules to which the linearisation constraints are attached is one of otherwise unconstrained immediate dominance schemata. In order to encode a normal set of phrase structure rules, one would need to place appropriately strong linear precedence constraints on the daughters of each of them. It is important to remember, with expressions from this language, that more constraints do not necessarily mean more work. Some of these linear precedence constraints, such as isolation and immediate precedence in particular, *constrain* search; and, in general, more of these actually lead to greater efficiency in a traditional chart parser. As a result, there will naturally be rule systems from this augmented language which are inherently intractable to parse. The object of defining this language is to make explicit those constraints which can be used to improve the situation.

Previous formal approaches to semi-free word order languages have taken either of two tracks. One has been to ignore the free word order generalisations, use only phrase structure rules, and simply write a multitude of them to achieve descriptive adequacy. The disadvantage of this approach is clearly that linguistic generalisations are lost, grammar writing is more difficult, and the grammar explodes in size. The other approach has been to abandon positional encoding (e.g., “the verb comes second”) and only allow relative ordering statements (e.g., “all As precede all Bs”). We believe that for both linguistic and computational purposes, a combination of these approaches promises the best results. If a rule naturally makes a statement about a fixed position, then the formalism should be able to express this directly. If, on the other hand, one needs to talk about relative order, then one needs to be able to do so.

Compilation

The general scheme for compilation is shown in Figure 2. LSL specifications, together with any attached relational or implicational constraints, will be compiled into FSAM code, and from there, in the manner of Zahnert (1997), into C code, which can be compiled with a C compiler into an executable system. The compilation takes place relative to a control strategy, which for the purposes of this project, will only be parsing. Once the executable has been created, its performance over a supplied corpus can be used to optimise the abstract machine further, in the manner of Penn

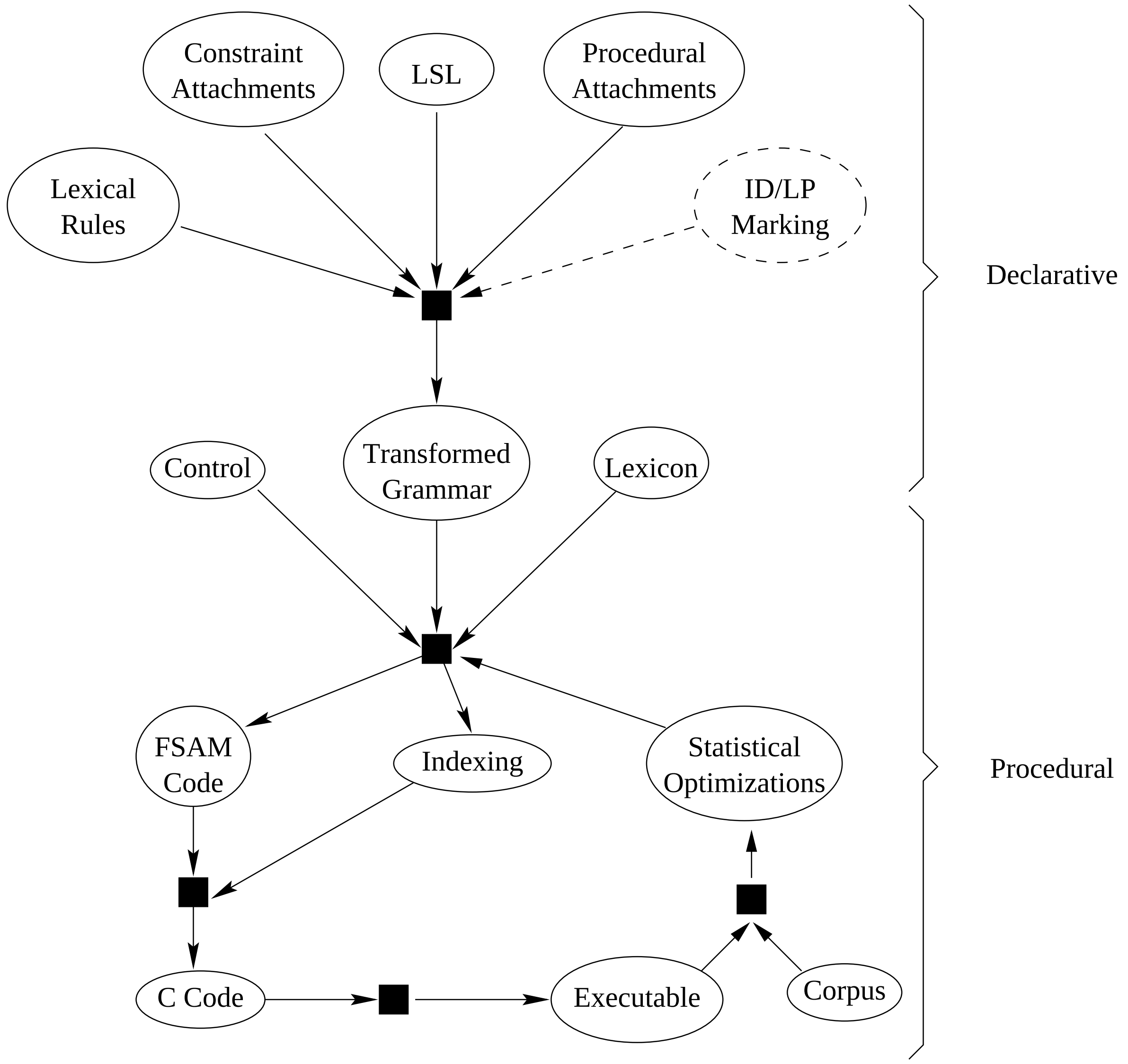

Abbildung 2: Compilation Overview.

(in prep.(b)). We will also explore the use of normal constraint-based grammars augmented with indications of which constraints or relations pertain to ID/LP instead of complete LSL specifications; but this is still speculative.
Before anything is compiled into FSAM code, the supplied grammar itself will be transformed at the level of its type signature and feature geometry. This is discussed in the next section.
Our goal is ultimately to provide portable compiled code for parsing with large-scale grammars which is as efficient as those grammars' linear constraints allow them to be. In particular, we expect the constraints from natural languages to admit efficient algorithms. The auxiliary attachments on the rules from LSL are simple, near-atomic statements about linear precedence. These provide as conservative of an extension

to phrase structure as possible to be adequate for freer word-order languages, while identifying to the next compilation step the sources of deterministic information to be used to constrain search, and the sources of complexity to be postponed.

Our strategy for compiling LSL specifications will be a combination of techniques from constraint logic programming and head-driven, or so-called “head-corner,” parsing. LSL does not actually have a bias towards a parsing strategy. The reason why we will pursue a head-driven approach is that we are working with HPSG, whose heads contain the necessary subcategorisation frames.

At compile-time, our first task is to determine what mother-daughter combinations of phrase structure rules are consistent with the constraints given. This is very similar to the pre-compilation of lexical rules devised by the most recent phase of the B4 project,[3] and also to Torisawa (1996). Having found those combinations, the next step is to *propagate* constraints, given a particular combination of mother and daughter, in order to maximise efficiency. Finding a language-preserving propogation strategy which improves efficiency will be one of the major tasks of the proposed phase of this project. Here the object is to push the more deterministic and computationally cheaper constraints *down* the derivation tree (i.e. towards the terminal nodes, which correspond to lexical items), and the more non-deterministic or computationally expensive constraints *up* the derivation tree, as much as possible. It is also important to perform this propogation efficiently at compile-time (naïvely considering all possible local subtrees, for example, could be quite expensive if the rules have large branching factors).

Which constraints are expensive or cheap will be determined, of course, by the final structure of LSL. A brief glance at our preliminary syntax on page 16, however, already gives some indication. Root descriptions, which are always non-disjunctive, play a very important role in refining search, and must be pushed down. That there is any flexibility to make such a push should be evident from the existence of constraints such as membership or decomposition, which allow reference to yields further down than the immediate daughters of a given rule.

Isolation must also be pushed down. Isolation corresponds to “bounding” or “compaction” in linearisation-based theories — it at least partially prevents scrambling, and therefore refines search. Immediate precedence, and to a lesser extent, general linear precedence, should also be pushed down. Membership, on the other hand, is inherently non-deterministic, and should generally be pushed up towards the root of the derivation, i.e. postponed. Universal quantification is not necessarily non-deterministic, but can be quite expensive, especially over large, unisolated domains, and must also be postponed.

Once these constraints have been properly distributed, the compiler must then generate FSAM code which enforces them inside a roughly head-driven parsing strategy. At run-time, this code must seek out potential heads, identify the smallest possible isolated domains in which their arguments can be guaranteed to be found, and find

[3]The reader is referred to the Ergebnisbericht of the most recent phase of the B4 project for a complete discussion.

those arguments, while collecting linear precedence information throughout to ensure consistency with the grammar. Argument and adjunct combination rules may also need to be distinguished by the user in order to avoid problems with potentially unbounded adjunction. It may be possible to identify certain cases automatically.

It is, of course, not unreasonable to expect that, in those cases where the linearisation constraints are strong enough to render a rule equivalent to a traditional phrase structure rule, that the system treat the rule as such. This is also one of our goals — to avail ourselves of true phrase structure, where it exists, even if the entire grammar may not be a phrase structure grammar. In the case of the B8 fragment grammar, for instance, the structures of determiner phrases and the verbal complex are analysed without recourse to linearisation or scrambling. In these cases, once a DP or VC head is detected, parsing of those constituents should proceed as in a normal head-driven parser.

Because we will be compiling LSL expressions into abstract machine code, it will also be possible to integrate (root) descriptions very tightly with the parsing control strategy itself, to postpone unification work as well. There is no reason, for example, why an entire unification must be performed when matching a daughter category, before peeking ahead to unify part of another daughter's category, if that part is more likely to produce a failure (which also prunes search). When working with normal context-free rules or even most definite clause grammars, the unification of categories is quite cheap. With typed feature-based grammars, however, the terms themselves grow very large very quickly. The abstract machine level will allow us to postpone part of that cost, in the hope that a failure will prevent us from paying all of it. The effects of postponing feature structure unification work during parsing will also be cumulative with the benefits of our linear constraint propogation.

Feature-based Grammar Transformation

Typed feature logics provide several degrees of freedom for determining how to represent linguistic information in a grammar. Although the familiar adage that types and features can be freely interchanged is, in fact, not true (because of the potential for structure sharing), there is some flexibility there. For example, in Pollard and Sag (1994), *index* objects could be entirely represented by subtypes rather than with features, because the theory never requires person, number and gender to be independently token-identical.

From the standpoint of linguistic theory, there may be other reasons to prefer a single correct way to represent some of these cases, in spite of the freedom which the logic provides. Because discovering when the translation between types and features is possible can be very difficult, and because making the translation itself can be equally difficult, an implementation should allow the linguist user to allocate information with complete freedom.

From the standpoint of an implementation, in general, we want to prefer types over features whenever possible because type-based information can be pre-compiled more

extensively and represented more atomically. This translation can, in fact, be made automatically relative to a fixed grammar (Penn in prep.(a)), without changing its predictions. Provided that we keep the total number of types within an acceptable processing bound (in the worst case, this translation causes an exponential explosion in the number of types), the translation will also increase efficiency. We will develop and implement an efficient algorithm for our abstract machine which will automatically find the optimal translation for a given grammar, translate the grammar, and construct a mapping of solutions back to their corresponding solutions in the user's original grammar for presentation to the user.
Similar methods have already been suggested at a higher level of processing, e.g. the use of "sponsorship" to eliminate empty categories (Johnson and Kay 1994), the use of lexical rules rather than unary phrase structure rules or the use of phrase structure to eliminate certain classes of constraints (Ergebnisbericht of the most recent phase of B4). The transformation proposed here has exactly the same motivation, only occurs at a lower level within the logic itself. The effects of these transformations, furthermore, will be cumulative with the statistical optimisations of Penn (in prep.(b)).

Feature Structure Abstract Machines

Our work on abstract machines for the next phase will extend AMALIA (Wintner 1997) to include:

- Feature-based constraint logic programming (e.g. ConTroll),
- Traditional parsing with feature-based grammar (e.g. Troll, ALE),
- LSL compilation (as described above),
- Optimisations:
  - type and appropriateness compilation
  - indexing strategy discovery
  - parallel processing

With respect to all of these, the object will be to integrate them into a single FSAM with a C-based compiler such as that of Zahnert (1997).
We will add to AMALIA a constraint resolution component with the functionality of ConTroll's, including its coroutining mechanism. AMALIA currently has no constraint resolver. Since HPSG is a constraint-based grammar formalism, the appeal of having a constraint resolving component should be quite clear. Historically, the problem has been that constraint resolution does not provide a reliable enough control strategy to handle a wide range of large-scale grammars efficiently.
Our experience with ConTroll in the context of the B8 fragment grammar has shown that the proper management of immediate dominance and linear precedence is the key

to doing this properly. The existence of an LSL component, or at least an indication of where ID/LP information is located, will ameliorate this problem. Non-ID/LP-related constraints can be compiled into clauses in a feature-based relational language, as described by Götz and Meurers (1995). An alternative computational model would be that of AMOZ (Mehl, Scheidhauer, and Schulte 1995), the abstract machine for Oz.

AMALIA also contains no relational extension to its phrase structure rules. Our experience with ConTroll and the B8 fragment has shown the necessity of using relations alongside constraints in large-scale grammars (Götz and Meurers 1997). Relations are, in fact, widely used in HPSG linguistic theories as well, not just implementations, although not always in a manner which is well understood. Our approach is to use an extra-logical procedural language, as in Troll or ALE, where predicates themselves are first-order-like predicates whose arguments are typed feature structures.

Compilation of this procedural language to the FSAM level proceeds very much as in the WAM itself, since the attachments themselves are not typed feature structures but Prolog-like predicates. Penn (in prep.(b)) is adding a similar language to an abstract machine for ALE, and will be our principal reference for this. An alternative is the language implemented in Zahnert (1997) or ConTroll, where predicates are interpreted in the logic of typed feature structures, but executed with the customary SLD-resolution of Prolog.

With respect to both our linear specification language (versus Manandhar (1995b)), and our procedural attachment language (versus other theoretical appeals to relations), the object is to remove information from the feature logic itself, where it is expressed declaratively, and explicitly relocate that information to a derivation rule, where it can be enforced more efficiently, using linear precedence and immediate dominance as guides.

We now consider each of the three optimisations in turn:

— Type and Appropriateness Compilation

The key to efficient processing of HPSG grammars is the exhaustive application of type and appropriateness information to data structure design and compilation. To this end, we will develop and implement a method of type and appropriateness representation based on the use of bit vector and other compact encodings.

Bit vector encodings were originally applied to inheritance-based polymorphic programming languages by Aït-Kaci, Boyer, Lincoln, and Nasr (1989), later implemented in the LIFE system. This method uses optimised subsumption vectors to represent type information. For a type signature with n types, an $n \times n$ binary matrix is created, in which a 1 occurs at row i, column j if and only if type i immediately subsumes type j. An example type hierarchy (taken from Aït-Kaci, Boyer, Lincoln, and Nasr (1989)) is shown in Figure 3, and its corresponding matrix in Figure 4. This matrix is then reflexively and transitively closed using a variant of boolean matrix multiplication. The closure of Figure 4 is shown in Figure 5. Now, each row i represents all

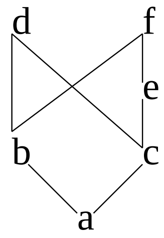

Abbildung 3: An example type hierarchy.

	a	b	c	d	e	f
a	0	1	1	0	0	0
b	0	0	0	1	0	1
c	0	0	0	1	1	0
d	0	0	0	0	0	0
e	0	0	0	0	0	1
f	0	0	0	0	0	0

Abbildung 4: The immediate subsumption matrix of Figure 3.

of the types which i subsumes, including itself.
The significance of this encoding is that the unification of type i and j is represented by the result of a boolean AND operation on the codes for i and j. This means that we do not need to look up the unification of i and j in a table, but can keep their codes inside the heap structures themselves and unify with simple logical operations, available to any computer's own instruction set. This preserves the highly desirable property of *locality of reference*. When an abstract machine consults any kind of table or array, the memory occupied by that array must be swapped into the computer's working memory. For very large numbers of types, the size of this array (which is quadratically larger) can be so large that it causes important, frequently used structures, such as representations of feature structures, to be swapped out of the computer's working memory. This loading and reloading of memory consumes a great deal of time at run-time. The use of a locally available encoding eliminates the need for this.

	a	b	c	d	e	f
a	1	1	1	1	1	1
b	0	1	0	1	0	1
c	0	0	1	1	1	1
d	0	0	0	1	0	0
e	0	0	0	0	1	1
f	0	0	0	0	0	1

Abbildung 5: The refl.-trans. closed subsumption matrix of Figure 3.

Appropriateness conditions are declarations which specify which types can have values for which features. In our interpretation of them, they allow an abstract machine to represent a feature structure of a fixed type as a frame of values of a fixed size, since all objects of that type must have values for the same features. They also allow abstract machines quicker access to feature values, since the position of these values is also fixed — no search through a list of feature names is necessary, in contrast to OSF terms. One of the drawbacks of having appropriateness, however, is that whenever a feature structure's type changes, its appropriateness might change too; and then an abstract machine must look up those changes in a table. In the same way, bit vector encodings of appropriateness can also preserve locality of reference.

Of course, the size of these vectors is not small either: n bits for n types. Aït-Kaci, Boyer, Lincoln, and Nasr (1989) suggest a method for *modularising* type signatures into sub-signatures which can recursively be encoded, thereby reducing the size to at best $log n$ bits per type. This modularisation works quite well with HPSG type hierarchies, which are typically meet semi-lattices with sparse subsumption matrices and few if any instances of non-trivial type unification. The method of Aït-Kaci, Boyer, Lincoln, and Nasr (1989) also provides an encoding of disjunctive types, which are important for efficient enforcement of HPSG's type logic.

The right modularisation algorithm can also be used very effectively for incremental compilation. (Fall 1996) presents a modularisation algorithm which degrades gracefully with on-line changes to a type hierarchy, and also considers several even more compact term encoding techniques, extending the work of (Mellish 1992). In every current feature-based system, when a change is made to the type hierarchy during development, the entire grammar must be recompiled, because of the potentially far-reaching changes to feature inheritance and unification, which is clearly undesirable.

To date, the only feature-based abstract machines that use bit-vector encodings are LIFE, Oz, and PAGE (Krieger and Schäfer 1994). These encodings are loosely based on (Aït-Kaci, Boyer, Lincoln, and Nasr 1989), whose proposal contained several theoretical and practical errors. The ProFIT system (Erbach 1994), which lacks an abstract machine, uses the original sparse term encoding of (Mellish 1992). No system has yet implemented the work of (Fall 1996) (other than the author's own experimentation with fixed test signatures). We propose to make an empirical comparison of the various improvements proposed by (Fall 1996) that extend to incremental compilation, and use the best as the basis for local representation of types in the FSAM.

In addition, Aït-Kaci, Boyer, Lincoln, and Nasr (1989), its realisations in LIFE, Oz and PAGE, and Fall (1996) all regard typing as potentially arity- and feature-independent information about data structures, just like OSF terms. We will also extend the encodings of Aït-Kaci, Boyer, Lincoln, and Nasr (1989) and Fall (1996) to compute appropriateness requirements locally. This will involve the addition of a minimal amount of extra information to the encodings which can be combined by boolean operations to predict the addition of appropriate features and value restrictions, value unifications, and value mappings to resulting terms during the unification of two heap representations. It will also involve creating extra matrix-theoretic compilation steps to generate the encodings and enforce the well-formedness of appropriateness declarations. This extension combines the benefits of locality of reference and totally well-typed interpretation of appropriateness, thus setting a new standard for signature compilation and unification efficiency for HPSG-like grammars.

— Indexing

In any large database indexed by feature structures (e.g. for generation, a lexicon; for parsing, a parsing chart), the naïve method of retrieval is simply to unify a candidate feature structure with every index. In the case of typed feature structures, one can first index by type and then unify only with feature structures of that type; but in a theory such as HPSG, where nearly every element in a parsing chart will be of the same type (*phrase*), this is a small improvement. Nevertheless, this has been the strategy chosen by almost every feature-based NLP system.

The one exception, ALE (Penn and Popescu 1997), does provide a somewhat better indexing method, by compiling a lexicon for generation into a large tree whose branches are types and whose nodes are feature paths. During the search for a matching feature structure, at every node in the tree, ALE must find the type at the path given by that node, and then follow the branch corresponding to that type. The leaves of this tree are lexical items themselves. By placing feature paths along the tree so as to reduce the average length of traversal through the tree, we reduce the time of access. The same strategy could also be used for indexing a parsing chart.

ALE, however, does not place feature paths in this way, but places them so that shorter paths come first and paths of equal length are placed alphabetically. It does make sense to place prefixes of long paths before the paths themselves, because they must be traversed anyway; but otherwise, this strategy is completely arbitrary. We will develop a better indexing method for feature structures, which will find a near-optimal indexing structure, given a fixed database of feature structures. Because we are concerned with parsing and parsing charts are not fixed in the same way as lexica, we will also be concerned with how to adjust and maintain changes to indexing trees efficiently at run-time.

We intend to develop an approach similar to those of Quinlan (1983), Utgoff (1988), and Chou (1991). All of these use impurity-based classification techniques to refine the structure of a decision tree efficiently into a near-optimal solution. The object is to position nodes so as to maximise purity (in our case, purity corresponds to a measure of how closely a set of feature structures satisfy a description) as quickly as possible. This can be contrasted with approaches from theoretical statistics, e.g. Breiman, Friedman, Olshen, and Stone (1984), which focus on the use of impurity-based methods to discover absolutely optimal classifications with no attention to computational cost on the one hand; and with other approaches from artificial intelligence which use conceptual clustering as a means of classifying large databases, particularly lexica (Carpineto and Romano 1992).

— **Parallel Processing**

With the advent of the Symmetric Multi Processing (SMP) architecture for desktop computers (e.g., SUN workstations and x86-based PCs), there is renewed interest in parallel processing, as parallel operating systems become more mature and hardware becomes cheaper. The FSAM provides us with the right level for researching parallelising optimisations to our architecture. We will consider three areas for parallelisation:

- Parallelisation of unification.
- Parallelisation of parsing.
- Parallelisation of constraint solving.

Early research on feature-based grammar saw the development of algorithms for parallel unification on large untyped feature structures with no appropriateness, on which the consensus has been that re-entrant structures simply provide too many bottlenecks to justify the latency and overhead costs associated with parallel architectures. We will re-investigate lower level parallelisation in the light of using type and appropriateness information to predict bottlenecks at compile-time in order to distribute processing more effectively. Citrin (1988) showed how to do this for Prolog, using static program analysis. The key that opens these results to us is, again, our use of appropriateness, which will allow us to adapt Citrin's static methods to our case.
With regard to parsing algorithms, parallelisation efforts were mainly concentrated on the context-free languages or even subsets thereof (cf. Nijholt (1991) and references cited therein). Although it has been shown that context-free parsing can theoretically be very fast on parallel machines[4], and many different methods have been proposed, practical results have not been promising. This is due to the fact that for context-free grammars with their small rules, grain size is often too small to be efficient. As the basic operations in our case will be much more expensive (feature structure unification), we can expect to have no problems with grain size.
Just as in logic programming, the are two forms of parallelism inherent to constraint solving for relational attachments: OR-parallelism (all clauses for a goal can be tried in parallel) and AND-parallelism (all goals can be solved in parallel). The more obvious choice for an actual implementation is OR-parallelism, since in a pure model, the execution of an OR-branch is independent of all other OR-branches. In a Prolog system, however, the situation is not as simple as in a pure model. The Aurora OR-parallel Prolog system (Lusk et al. 1988; Carlsson and Szeredi 1995) must deal with side effects such as system calls, cuts etc. Since we provide a pure language, this is not an issue for us. We will adapt OR-parallel methods from the Aurora abstract machine (Carlsson and Szeredi 1995) to the typed feature-based case.

[4]Context-free recognition can be achieved in logarithmic time using polynomially many processors (in relation to the length of the input string), see (de Vreught 1993) and references cited therein.

3.53 Zeitplan

1998
- (a) Design linearisation specification language (LSL)
- (b) Investigate formal properties of LSL
- (c) Design and implement improved type and appropriateness compiler for feature structure abstract machine (FSAM)
- (d) Add constraint resolution, relational attachments, and coroutining to AMALIA (henceforth, FSAM)

1999
- (a) Design and implement LSL compiler
- (b) Add constraint resolution, relational attachments, and coroutining to FSAM (continued)
- (c) Design and implement FSAM control strategy for LSL grammars
- (d) Design and implement indexing strategy

2000
- (a) Extend FSAM to parallel processing
- (b) Implement grammar transformation and statistical optimisations
- (c) Design and implement C-based compiler for FSAM

3.6 Stellung innerhalb des Programms des Sonderforschungsbereichs

A3 Via the implementation in the B8 project, we will support A3's analyses of parentheticals, adverbial clauses, and so-called "weiterführenden Nebensätze" (subordinate clauses). We expect particularly the analysis of parentheticals to provide difficult test cases for our system.

A8 We will consult with A8 on all aspects relating to the logical semantics of LSL. In particular, because LSL can, in some sense, be seen as a logic over derivations, it is relevant to the class of formalisms being researched in A8.

B7 We will continue the successful collaboration of the last phase and further integrate shallow parsing techniques (chunk parsing) with our system. Pending the integration of a numerical component, we will be able to use the statistically annotated lexica induced by B7.

B8 The B8 project will be the most important partner of B4. LSL will be developed in close cooperation with B8, ensuring the linguistic adequacy of the language. We will provide the tools for the development and implementation of the Fragmentgrammatik. Conversely, the Fragmentgrammatik will be the major test-piece for our compilers and abstract machine.

B9 As in the last phase, B4 will provide the interface for the interaction between B8 and B9.

B12 B4 and B12 will share their experiences with enhanced phrase structure systems for feature-based grammars. The work of B12 will provide insights into where free word order is particularly problematic for a context-free backbone.

3.7 Ergänzungsausstattung für das Teilprojekt

es bedeuten:

PK: Personalbedarf und –kosten (Begründung vgl. 3.71)

SV: Sächliche Verwaltungsausgaben (Begründung vgl. 3.72)

I: Investitionen (Geräte über 20.000; Begründung vgl. 3.73)

PK	Bewilligung 1997		1998			1999			2000		
	Vergü-tungs-gruppe	An-zahl	Vergü-tungs-gruppe	An-zahl	Betrag in DM	Vergü-tungs-gruppe	An-zahl	Betrag in DM	Vergü-tungs-gruppe	An-zahl	Betrag in DM
			SV (C3)	0,25	33,3	SV (C3)	0,25	33,3	SV (C3)	0,25	33,3
	BAT IIa	2	BAT IIa	2	199,2	BAT IIa	2	199,2	BAT IIa	2	199,2
	Wiss. HK	1	Wiss. HK	1	32,4	Wiss. HK	1	32,4	Wiss. HK	1	32,4
	Stud. HK	1	Stud. HK	1	19,2	Stud. HK	1	19,2	Stud. HK	1	19,2
	zusammen	4	zusammen	4,25	284,1	zusammen	4,25	284,1	zusammen	4,25	284,1

SV		1998		1999		2000	
		Kostenkategorie	Betrag in DM	Kostenkategorie	Betrag in DM	Kostenkategorie	Betrag in DM
	515	Kleingeräte	22,5	Kleingeräte		Kleingeräte	
	522	Verbrauchsmittel	4,0	Verbrauchsmittel	4,0	Verbrauchsmittel	4,0
	527	Reisemittel	10,0	Reisemittel	10,0	Reisemittel	10,0
	531b	Kopien	2,0	Kopien	2,0	Kopien	2,0
		zusammen	38,5	zusammen	16,0	zusammen	16,0

I		1998	1999	2000
		Investitionen insgesamt:	Investitionen insgesamt:	Investitionen insgesamt:

3.71 Begründung des Personalbedarfs

	Name, akad. Grad, Dienststellung	engeres Fach des Mitarbeiters	Institut der Hochschule oder der außeruniv. Einrichtung	Mitarbeit im Teilprojekt in Stunden pro Woche (beratend: B)	auf dieser Stelle im SFB tätig seit	derzeitige Einstufung und beantragte Vergütungsgruppe
Grundausstattung						
3.71.1 *wissenschaftliche* Mitarbeiter (einschl. Hilfskräfte)	1. Hinrichs, Erhard, Prof. Dr.	Syntax, Semantik, Comp.-Ling.	SfS	5	1992	
	2. Gerdemann, Dale, Dr.	Comp.-Ling. Syntax	SfS	10	1992	
3.71.2 *nichtwissenschaftliche* Mitarbeiter	3. Saile, Jochen	Syst. Prog.	SfS	10	1992	
Ergänzungsausstattung						
3.71.3 *wissenschaftliche* Mitarbeiter (einschl. Hilfskräfte)	4. Wintner, Shalom, Dr.	Comp. Ling., Compiler				BAT IIa
	5.a. Penn, Gerald, MA	Comp.-Ling., Syntax, Logik	SFB		1996	1/2 BAT IIa
	5.b. Götz, Thilo, MA	Comp.-Ling., Logik	SFB		1992	1/2 BAT IIa
	6. Alt, Natalie, MA	Comp.-Ling., Syntax	SFB	19	1994	Wiss. HK
	7. Bühler, Dirk	Logik-Programmierung	SFB	19	1996	Stud. HK
3.71.4 *nichtwissenschaftliche* Mitarbeiter						

Aufgabenbeschreibung:

1. Hinrichs (project leader): Consultation on linguistic and computational aspects of research activities, teaching responsibilities, leadership of the project.

2. Gerdemann (project leader): Consultation on logical and computational aspects of research activities, teaching responsibilities, leadership of the project.

3. Saile: System programming, system maintenance, backups, updating of software.

4. Wintner: Work on abstract machine compilation and parsing techniques. Development of indexing strategy for large-scale chart access. Design and implement

algorithms for processing linear specification language. Porting abstract machine to different computer platforms. Maintain and further develop abstract machine debugger.

5. Penn: Work on integration of ConTroll and ALE functionality into AMALIA. Development of statistical optimisations and grammar transformation techniques for abstract machine. Interface to the B8 fragment project to ensure adequacy of linear specification language and computational system to linguistic accounts of linearisation and other phenomena.

6. Götz: Design of linear specification language, and its logical semantics. Extension of ConTroll logical relation and constraint compiler to augmented linear specification language. Work on parallel processing techniques.

7. Alt: Work on extensions of typed feature logic to directly encode linearisation phenomena.

8. Bühler: Bibliographical work; minor implementation tasks.

Gastprofessor (Sondervertrag): The goals of the project would be furthered by consultation and joint work with outside researchers such as Bob Kasper and Suresh Manandhar. Such collaboration could also help to make the work of B4 more widely know outside the SFB. In previous phases of the project, we have benefitted greatly from visits by Bob Carpenter, Carl Pollard and Bob Kasper. In the next phase of the project, we expect to collaborate especially closely with Bob Kasper on linguistic and computational aspects of linearisation grammars. And we expect to collaborate with Suresh Manandhar on extensions of typed feature logic, including the application of these logics for linearisation.

3.72 Aufgliederung und Begründung der Sächlichen Verwaltungsausgaben (nach Haushaltsjahren)

	1998	1999	2000
Für Sächliche Verwaltungsausgaben – stehen aus der *Grundausstattung* voraussichtlich zur Verfügung	15,5	4,0	4,0
– werden als *Ergänzungsausstattung* beantragt	38,5	16,0	16,0

Vorhandene Grundausstattung

The project has available for the duration of the next phase: a Sun Sparcstation 4, a PowerMac 7200/90 (Grundausstattung for Dale Gerdemann), an Apple Laserwriter (available to all project members) and an Objectcenter C++ Development environment.

Begründung der beantragten Ergänzungsausstattung

- (512) Bücher und Zeitschriften:
  For acquisition of new project specific publications (including dissertations, working papers and technical reports).

- (515) Kleingeräte:
  Three new Linux PCs are needed primarily to serve as replacements for 2 Sparcstation ELC Workstations (Grundausstattung), which were purchased in April 1992. The warranty for these machines is running out and cannot be extended. If these machines should fail, there are no existing funds to replace them.

  The machines to be purchased are:

  - Two 2-processor Linux PCs, with 96 megabyte Ram and 2 gigabyte hard disk (to be shared by Shalom Wintner, Thilo Götz and Gerald Penn).
  - One Linux PC with 64 megabyte RAM and 2 gigabyte harddisk (for the wiss. and stud. Hilfskräfte).

  The three machines will be used only for B4 project work.

  The additional processing capacity of these machines is necessary since they will be used for the ConTroll system (including graphical interface) on larger linearisation-based grammars developed in the B8 project.

- (522) Verbrauchsmaterial:
  Office and computer supplies.

- (527) Reisemittel:
  To present our work at relevant conferences and workshops, including:

  - ACL annual meeting
  - COLING
  - DGfS Fachtagung in Computerlinguistik
  - ACL International Workshop on Parsing Technologies
  - Mathematics of Language Workshop
  - HPSG Workshop

  To take part in ESSLLI summer schools.

  To interact with related ongoing projects at other universities (Saarbrücken, Groningen, etc).

- (531b) Kopien:
  Copies of articles and papers relevant to the project

3.73 Investitionen

(Geräte über DM 20.000 brutto und Fahrzeuge)
Es werden keine Investitionsmittel beantragt.

Literatur

Aggoun, A., D. Chan, P. Dufresne, E. Falvey, H. Grant, A. Herold, G. Macartney, M. Meier, D. Miller, S. Mudambi, B. Perez, E. van Rossum, J. Schimpf, P. A. Tsahageas, and D. H. de Villeneuve. (1995). *ECLiPSe User Manual* (3.5 ed.). ECRC.

Aït-Kaci, H., R. Boyer, P. Lincoln, and R. Nasr (1989). Efficient implementation of lattice operations. *TOPLAS 11-1-1989*.

Aït-Kaci, H. and R. DiCosmo (1993). Compiling order-sorted feature term unification. Technical report, DEC Paris Research Laboratory.

Aldag, B. (1997). Compiling HPSG grammars into context-free grammars. Arbeitspapiere des SFB 340, Seminar für Sprachwissenschaft, Universität Tübingen.

Amraoui, M. (1988). *Une Expérience de Compilation de Prolog II sur MALI*. Ph. D. thesis, Unversité de Rennes I.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and Regression Trees*. Pacific Grove, California: Wadsworth and Brooks.

Briscoe, T. and J. Carroll (1993). Gernalized probabilistic lr parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics 19*(1).

Carlsson, M. (1987). Freeze, indexing, and other implementation issues in the wam. In J.-L. Lassez (Ed.), *International Conference on Logic Programming, Melbourne*, pp. 40 – 58. MIT Press.

Carlsson, M. (1991). Boolean constraints in sicstus prolog. Technical Report T91:09, Swedish Institute of Computer Science.

Carlsson, M. and P. Szeredi (1995). The Aurora abstract machine and its emulator. Technical report, Swedish Institute of Computer Science and University of Bristol. Draft version.

Carpenter, B. (1992a). ALE user's guide. Laboratory for Computational Linguistics Report CMU-LCL-92-1, Laboratory for Computational Linguistics, Carnegie Mellon University.

Carpenter, B. (1992b). *The logic of typed feature structures*, Volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.

Carpenter, B. and G. Penn (1994). ALE – The Attribute Logic Engine, User's Guide, Version 2.0.1, December 1994. Technical report, Carnegie Mellon University.

Carpineto, C. and G. Romano (1992). Galois: An order-theoretic approach to conceptual clustering. In *Proceedings of AAAI.*

Chou, P. A. (1991). Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence 13*(4).

Citrin, W. (1988). *Parallel Unification Scheduling in Prolog.* Ph. D. thesis, University of California at Berkely.

de Vreught, J. P. M. (1993). *Parallel Parsing.* Ph. D. thesis, Technische Universiteit Delft.

Debray, S. K. (1986). Register allocation in a prolog machine. In *IEEE Symposium on Logic Programming.*

Diaz, D. and P. Codognet (1993). A minimal extension of the wam for clp(fd). In D. S. Warren (Ed.), *Logic Programming, Proceedings of the Tenth International Conference*, pp. 774 – 790. MIT Press.

Dowty, D. (1989). Toward a minimalist theory of syntactic structure. Paper prepared for the Tilburg Conference on Discontinous Constituency. Corrected version February 24, 1992.

Engelkamp, J., G. Erbach, and H. Uszkoreit (1992). Handling linear precedence constraints by unification. In *30th annual meeting of the ACL*, pp. 201–208.

Erbach, G. (1994). Profit: Prolog with features, inheritance and templates. Technical report, CLAUS.

Erbach, G., M. van der Kraan, S. Manandhar, H. Ruessink, C. Thiersch, and W. Skut (1995). Extending unification formalisms. In *Proceedings of the 2nd Language Engineering Convention*, London.

Fall, A. (1996). *Reasoning with Taxonomies.* Ph. D. thesis, Simon Fraser University.

Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985). *Generalized Phrase Structure Grammar.* Harvard University Press.

Gerdemann, D. (1995). Term encoding of typed feature structures. In *Proceedings of the Fourth International Workshop on Parsing Technologies*, pp. 89 – 98.

Gerdemann, D., T. Götz, J. Griffith, S. Kepser, and F. Morawietz (1995). *Troll manual* (draft ed.). Seminar für Sprachwissenschaft, Universität Tübingen.

Gerdemann, D. and P. J. King (1994). The correct and efficient implementation of appropriateness specifications for typed feature structures. In *Proceedings of COLING-94*, Kyoto, Japan.

Götz, T. (1997). *The ConTroll manual and user's guide* (Draft ed.). Seminar für Sprachwissenschaft, University of Tübingen.

Götz, T. and W. D. Meurers (1995). Compiling HPSG type constraints into definite clause programs. In *Proceedings of the Thrirty-Third Annual Meeting of the ACL*, Boston. Association for Computational Linguistics.

Götz, T. and W. D. Meurers (1997). Interleaving universal principles and relational constraints over typed feature logic. In *Proceedings of the 35th Annual Meeting of the ACL and the 8th Conference of the EACL*, Madrid, Spain.

Griffith, J. (forthcoming). *DUSt: a disjunctive approach to unification.* Ph. D. thesis, Universität Tübingen.

Holzbaur, C. (1990). *Specification of Constraint Based Inference Mechanism through Extended Unification.* Ph. D. thesis, Dept. of Medical Cybernetics & AI, University of Vienna.

Holzbaur, C. (1995). Ofai clp(q,r) manual. Technical Report TR-95-09, Austrian Research Institute for Artificial Intelligence, Vienna.

Janssens, G., B. Demoen, and A. Mariën (1988). Improving the register allocation in wam by reordering unification. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium.*

Johnson, M. and M. Kay (1994). Parsing and empty nodes. *Computational Linguistics 20*(2), 289 – 300.

Kasper, R., A. Kathol, and C. Pollard (1995). A relational interpretation of linear precedence constraints. In *Proceedings of the workshop on the mathematics of language.*

King, P. J. (1989). *A logical formalism for head-driven phrase structure grammar.* Ph. D. thesis, University of Manchester.

King, P. J. (1994). An expanded logical formalism for head-driven phrase structure grammar. Arbeitspapiere des SFB 340 Nr. 59, Universität Tübingen.

Krieger, H.-U. and U. Schäfer (1994). TDL — a type description language for hpsg; part 1: Overview. Technical report, DFKI.

Lusk, E. L., R. Butler, T. Disz, R. Olson, R. A. Overbeek, R. Stevens, D. H. D. Warren, A. Calderwood, P. Szeredi, S. Haridi, P. Brand, M. Carlsson, A. Ciepielewski, and B. Hausman (1988). The Aurora or-parallel Prolog system. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, pp. 819–830.

Manandhar, S. (1993). *Relational Extensions to Feature Logic: Application to Constraint Based Grammars.* Ph. D. thesis, Ph.D. thesis, Department of Artificial Intelligence, University of Edinburgh.

Manandhar, S. (1995a). An attributive logic of set descriptions and set operations: Extended report. *Edinburgh Working Papers in Cognitive Science Nonclassical Feature Systems*(10), 39–60.

Manandhar, S. (1995b). Deterministic consistency checking of LP constraints. In *proceedings of the 7th conference of the EACL*, Dublin, Ireland.

Mariën, A. and B. Demoen (1991). A new scheme for unification in wam. In *Logic Programming, Proceedings of the 1991 International Symposium*, pp. 257 – 271.

Mehl, M., R. Scheidhauer, and C. Schulte (1995). An abstract machine for oz. Technical report, DFKI.

Meier, M. (1990). Compilation of compound terms in prolog. In *Logic Programming, Proceedings of the 1990 North American Conference*, pp. 63 – 79.

Mellish, C. (1985). Some global optimizations for a prolog compiler. *Journal of Logic Programming* (1), 43 – 46.

Mellish, C. (1992). Term-encodable description spaces. In D. R. Brough (Ed.), *Logic Programming: New Frontiers.* Kluwer Academic Publishers.

Meurers, D., F. Richter, M. Sailer, and H. Winhart (1997). Ein HPSG-Fragment des Deutschen, Teil 1: Theorie. Arbeitspapiere des SFB 340, Universität Tübingen.

Morawietz, F. (1994). Formalization and Parsing of Typed Unification-Based ID/LP Grammars. Master's thesis, Universität Tübingen.

Nerbonne, J., K. Netter, and C. Pollard (Eds.) (1994). *German in Head-Driven Phrase Structure Grammar.* CSLI Publications. Lecture Notes 46.

Nijholt, A. (1991). Overview of parallel parsing strategies. In M. Tomita (Ed.), *Current Issues in Parsing Technology*, Chapter 14, pp. 207–229. Kluwer Academic Publishers.

Penn, G. (1993). A utility for typed feature structure-based grammatical theories. Master's thesis, Carnegie Mellon University.

Penn, G. (1997). Multiple wh-extraction and clitic placement in serbo-croatian. Linguistics Qualifying Paper, Carnegie Mellon University.

Penn, G. (in prep.(a)). Parametric types for attribute-value logics. *To submit to Journal of Logic Language and Information.*

Penn, G. (in prep.(b)). *Statistical Optimizations in a Feature Structure Abstract Machine.* Ph. D. thesis, Carnegie Mellon University.

Penn, G. and O. Popescu (1997). Head-driven generation and indexing in ale. Technical report, Language Technologies Institute.

Pollard, C., R. Kasper, and R. Levine (1993). Studies in Constituent Ordering. Toward a Theory of Linearization in Head-Driven Phrase Structure Grammar. Research Proposal to the National Science Foundation, Ms. Ohio State University.

Pollard, C. and I. A. Sag (1994). *Head-Driven Phrase Structure Grammar.* Chicago: University of Chicago Press.

Qu, Y. (1994). An abstract machine for typed feature structure grammar theories. Master's thesis, Carnegie Mellon University.

Quinlan, J. (1983). Learning efficient classification procedures. In Michalski, Carbonell, and Mitchell (Eds.), *Machine Learning: an artificial intelligence approach*. Morgan Kaufmann.

Reape, M. (1989). A logical treatment of semi-free word order and bounded discontinuous constituency. In *Proceedings of the 4th Conference of the European Chapter of the ACL*, pp. 103–110.

Reape, M. (1992). *A Formal Theory of Word Order: A Case Study of West Germanic*. Ph. D. thesis, University of Edinburgh.

Reape, M. (1994). Domain Union and Word Order Variation in German. In (Nerbonne, Netter, and Pollard 1994), pp. 151–197.

Richter, F. and M. Sailer (1995). Remarks on Linearization. Reflections on the Treatment of LP-Rules in HPSG in a Typed Feature Logic. Magisterarbeit, Seminar für Sprachwissenschaft, Universität Tübingen.

Torisawa, K. (1996). Off-line raising, dependency analysis, and partial unification. In *Third Annual Conference on Natural Language Processing*.

Utgoff (1988). Id5: an incremental id3. In *International Machine Learning Conference, Ann Arbor*.

van Noord, G. (1991). Head corner parsing for discontinuous constituency. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 114–121.

van Roy, P., M. Mehl, and R. Scheidhauer (1995). Integrating efficient records into concurrent constraint programming. Technical report, DFKI.

Warren, D. H. D. (1983). An abstract Prolog instruction set. Technical note 309, SRI International.

Wintner, S. (1997). *An Abstract Machine for Unification Grammars*. Ph. D. thesis, Technion.

Wintner, S. and N. Francez (1994). Abstract machine for typed feature structures. Technical report, Laboratory for Computational Linguistics, Technion.

Zahnert, A. (1997). fl2c – ein Compiler für CLP(TFS). Diplomarbeit, Fakultät für Informatik, Universität Tübingen.

Zwicky, A. (1986). Concatenation and liberation. In *Proceedings of the Twenty-Second Regional Meeting of the Chicago Linguistic Society*, pp. 65–74.