Ergebnisbericht

Teilprojekt: B4

Thema: From Constraints to Rules: Efficient Compilation of HPSG Grammars

Leiter: Prof. Dr. Erhard Hinrichs
Dr. Dale Gerdemann

Mitarbeiter: Thilo Götz, Detmar Meurers, Guido Minnen, Gerald Penn

Inhaltsverzeichnis

1 Kenntnisstand bei der letzten Antragstellung und Ausgangsfragestellung

1.1 Ausgangsfragestellung

The aim of our project was the development of a computational system to process HPSG constraint grammars. While computational feature structure systems generally offer phrase structure rules as the main building block for a grammar, linguists conceive of an HPSG grammar as

- a set of implicational constraints,
- a set of linear precedence constraints, and
- a set of lexical rules.

It was, thus, our goal to design a system which could handle grammars consisting of these components, and to be able to process efficiently with such grammars. For processing, we intended to investigate the following techniques:

- Top-down interpretation using co-routining to avoid termination and efficiency problems.
- Goal-directed bottom-up interpretation using Magic compilation.
- Grammar transformation techniques such as extraction of phrase structure.

Since the fragment grammar of the B8 project depended on the practical results achieved in our project, we pursued all of these avenues in parallel.

1.2 Kenntnisstand bei der letzten Antragstellung

1.21 Constraint grammars

General implicative constraints

In the previous phase of this project, we developed the Troll system (Gerdemann et al. 1995). Troll is a phrase structure based typed feature logic system, based on the logical formalism of King (1989;1994). In particular, the Troll system implements efficient satisfiability checking based on the exhaustive type discipline of King (1989) as described in (Gerdemann and King 1993; 1994; Götz 1994; Gerdemann (1995a)), using the approach to named disjunctions from Gerdemann (1991).

The Troll system faithfully implemented the typing system of King (1989; 1994). It did not, however, allow the use of general implicational constraints as they are used in HPSG linguistics and formalised by King. Such constraints allow the formulation of principles which generalise over sets of rules. Grammatical principles such as Binding Theory or X-bar Theory can be formalised in a natural way and don't have to be encoded for every individual rule. As an example, consider the Semantics Principle of (Pollard and Sag 1987), p. 110.

DTRS:*headed-struc* $\Rightarrow$
  SEM:(CONT: successively-combine-semantics(X,Y) $\wedge$
    INDICES: collect-indices(Z)) $\wedge$
  DTRS:(Z $\wedge$
    HEAD-DTR:SEM:CONT: X $\wedge$
    COMP-DTRS: Y)

The principle states how to compute the semantics of any headed constituent as a function of the semantics of the daughters. A straightforward encoding of this principle as procedural attachment to phrase structure rules is possible. However, this is not very modular, and as the number of principles increases, becomes impractical. It is preferable to be able to state such principles directly and leave their integration with other constraints to an automatic compilation process. It was thus our goal to generalise the Troll system to be able to handle implicational principles as the one shown above.

At the time of writing the proposal, no computational system existed that could handle constraints like the one shown above. Only TFS (Emele 1994) allowed the linguist to define implicational constraints. However, TFS only allowed types as antecedents and was thus restricted in expressive power. It was also very slow, which made it impractical for large grammars. It was clear that the TFS system could be improved upon in terms of compilation techniques and processing control.

Linear precedence constraints

LP constraints are usually seen to hold between sisters of a local tree. A constraint $A \prec B$ is intended to mean that if in a local tree, a node a occurs that satisfies A and a node b occurs that satisfies B, then b must not precede a. The interpretation of such constraints was reasonably clear for simple case. Problems, however, turned up with structure sharing across A and B, as well as with calls to relational goals. This was later corroborated by Kasper et al. (1995). The situation was compounded by the fact that the formalism we developed did not have a phrase structure backbone.

It was unclear which domain LP constraints should even operate on. We thus needed to develop a new treatment of LP constraints in the context of constraint grammars. As it turned out, the fragment grammar developed in B8 did not use LP statements as described above at all, but a much stronger notion of linearisation grammars.

Lexical rules

Lexical rules play an important role in the highly lexicalised theoretical frameworks such as HPSG (Pollard and Sag 1987, 1994) and LFG (Bresnan 1982), where the information-rich lexical specifications often drive the analyses.
At the time of the proposal for the current phase of the B4 project, the logical and computational side of lexical rules, however, was largely unexplored. A formal foundation for the general HPSG architecture was available (King 1989, 1994), but even though certain aspects of lexical rules had been discussed in the literature, for example in (Flickinger, Pollard, and Wasow 1985), (Flickinger 1987), and (Pollard 1993), no formal semantics for the lexical rule mechanism as used in HPSG had been provided.
On the computational side, the systems based on typed feature structures in which HPSG-like grammars could be encoded, such as CUF (Dörre and Eisele 1991; Dörre and Dorna 1993), or TFS (Emele and Zajac 1990; Emele 1994) usually offered no computational treatment of lexical rules at all. To nevertheless encode the generalisations behind lexical rules, they are commonly encoded as unary phrase structure rules (see, for example, (Copestake 1992)).
Another kind of treatment is exemplified by the ALE system (Carpenter and Penn 1994), where a simple front end for specifying fully specific lexical rules is included and the lexical rules are used to expand out the full lexicon at compile time. This saves the user from having to specify everything by hand, but a compile-time expansion of a lexical rules encoding an infinite lexicon may lead to a loss of completeness. ALE neither offers the automatic property transfer desirable in a lexical rule implementation, nor does it give a formal semantics for its lexical rules.

1.22 Transformation and interpretation of constraint grammars

At the start of the 95-97 phase of the B4 project it was unclear how it would be possible to process large definite clause programs efficiently implementing the type of constraint grammars that the B8 project was going to implement. The main problem was the fact that the constraint grammars as we proposed to implement do not necessarily have a phrase structure backbone.

Top-down interpretation

A form of interpretation that immediately presents itself is top-down interpretation. Just as in a logic programming language such as Prolog, top-down interpretation

may be implemented as depth-first search with backtracking.
Unguided depth-first search for natural language grammars is known to suffer from termination and efficiency problems. In the previous phase, we implemented a top-down interpreter for the definite clause attachments of Troll. It used delay statements to control the order of execution of subgoals in order to avoid termination problems. We thus had some prior experience implementing co-routining constraint solvers for typed feature logic systems.
However, for the this phase, we not only had to handle grammars without phrase structure backbone, they would also be much bigger. In order to make up for the efficiency problems that we could expect with respect to pure constraint grammars, we knew that we had to extend our previous work using various compilation techniques.

Goal-directed bottom-up interpretation

Program transformation can be used for more goal-directed bottom-up interpretation, an interpretation that is particularly suitable for highly lexicalised HPSG grammars. The transformation for this purpose is the so-called *magic* transformation (for an introduction, see Nilsson and Maluszynski (1995) and references cited therein). Magic transformation (or "magic compilation") was a natural avenue for us to explore, since we had in the previous phase experimented with the related technique of Earley deduction (Pereira and Warren 1983) for the purpose of natural language generation (Minnen et al. 1995).
Magic transformation differs from Earley deduction, however, in that magic takes the on-line filtering (search space pruning) out of the interpreter and puts it into the program. When the program is then interpreted bottom up, the processing steps correspond very closely to Earley deduction. The difference is, however, that having the filtering incorporated into the grammar opens up the possibility of having further transformation steps explicitly manipulate the filtering portion of the program. This was a possibility that we wanted to explore.
In the previous phase of the project we developed a magic compiler for rule-based typed feature structure grammars. At the start of the project it was unclear though to what extent magic compilation could be used for efficient processing of large definite clause programs implementing HPSG grammars.

Extraction of phrase structure

The point of having a phrase structure backbone is that standard parsing techniques (e.g., chart parsing) may be used. The essential advantage of a parser is that it can use the input string to index the search space. This is most evident in a chart parser that for an input string of length n uses an $(n+1) \times (n+1)$ matrix to store partial parse results, thus avoiding unnecessary recomputation of partial results. We thus planned to extract a phrase structure backbone wherever possible.
At the time of the last proposal, it was already clear that the extraction of phrase structure was, in general, an undecidable problem. It was our belief, however, that

for realistic grammars, and with various auxiliary markings supplied by the user, such as where phonology was represented, phrase structure could still be extracted to improve parsing performance over large-scale grammars.

2 Ergebnisse und ihre Bedeutung

2.1 Implicational constraints

Implicational constraints are the basic elements of an HPSG grammar. Together with definite relations over feature terms, they allow a concise and perspicuous encoding of grammatical rules and principles (Götz and Meurers 1997). To illustrate this, consider the small example grammar in Fig. 1 and Fig. 2. Constraints 1-4 in Fig. 1 encode

1. $bar : zero \rightarrow word$
2. $bar : one \rightarrow head\text{-}dtr : bar : (\neg two)$
3. $bar : two \rightarrow subcat : [\,]$
4. $phrase \rightarrow comp\text{-}dtr : bar : two$
5. $phrase \rightarrow head : X \wedge head\text{-}dtr : head : X$
6. $phrase \rightarrow comp\text{-}dtr : X \wedge subcat : Y \wedge head\text{-}dtr : subcat : [X|Y]$

Abbildung 1: X-bar theory, head feature principle and subcat principle

a simple version of X-bar theory, constraint 5 ensures the propagation of categorial information along the head path, and constraint 6 ensures that complements obey the subcategorisation requirements of the heads.
The lexicon and phrase structure rules are encoded in the *wfs* relation shown in Fig. 2. The *wfs* predicate takes three arguments: a difference list pair threading the string through the tree in the style of DCGs, and the syntactic category (*sign*) as functional style result argument.[1] The analysis tree is encoded in two daughters features as part of the syntactic categories in the style of HPSG. Clause 1 of *wfs* combines a verbal projection with its subject, and clause 2 with its complements. The lexical entries for the verbs “loves” and “sleeps” are specified in clauses 3 and 4, respectively. Finally, clause 5 defines lexical entries for the proper names “Arthur” and “Tintagel”.
The interaction of implications and relations in such grammars is rather intricate and compilation proceeds in several passes. The first steps consist in equivalence transformations on the implicational constraints. The correctness of these transformations is shown in (Götz in prep.).

1. Elimination of complex antecedents. All implications are simplified s.t. the antecedent is only a type. An example of this is shown in Fig. 3.

[1] We use standard abbreviatory bracket notation for lists.

```
1.  wfs( (phrase,
          head: verb,
          subcat: [],
          bar: two,
          comp-dtr: CD,
          head-dtr: HD), P0, P1) :- wfs(CD, P0, P1), wfs(HD,P1,P).

2.  wfs( (phrase,
          head: verb,
          subcat: ne-list,
          head-dtr: HD,
          comp-dtr: CD), P0, P1) :- wfs(HD, P0, P1), wfs(CD, P1, P).

3.  wfs( (word,
          head: verb,
          bar: zero,
          subcat:[(head: noun),(head: noun)],
          phon: (loves, X)), [X|Y], Y).

4.  wfs( (word,
          head: verb,
          bar: zero,
          subcat: [head: noun],
          phon: (sleeps, X)), [X|Y], Y).

5.  wfs( (word,
          head: noun,
          bar: two,
          subcat: [],
          phon:((arthur ; tintagel), X)), [X|Y], Y).
```

Abbildung 2: Phrase structure rules and the lexicon

2. Constraint inheritance. Having reduced all antecedents to simple types, one can then collect all constraints on maximally specific types. For example, suppose we have three types, a, b and c s.t. a subsumes b and c. Furthermore, suppose we have the set of constraints $C_1 = \{a \rightarrow \phi,\ b \rightarrow \psi,\ c \rightarrow \xi_1,\ c \rightarrow \xi_2\}$. Then C_1 is equivalent to the set $C_2 = \{b \rightarrow (\phi \wedge \psi),\ c \rightarrow (\phi \wedge \xi_1 \wedge \xi_2)\}$.

3. Disjunctive normal form. The consequents of the implications can now be

brought into DNF, so that we now have a set of constraints of the form $a \rightarrow (\phi_1 \vee \ldots \vee \phi_n)$, where a is a maximally specific type, and each ϕ_i contains no disjunction or negation[2]. For each maximally specific type, there is at most one such constraint.

$$\begin{aligned} bar : zero \rightarrow word &\equiv sign \wedge bar : zero \rightarrow word \\ &\equiv sign \rightarrow \neg(bar : zero) \vee word \\ &\equiv sign \rightarrow bar : (one \vee two) \vee word \end{aligned}$$

Abbildung 3: Complex antecedent elimination

In this form, the implications can be interpreted as special kinds of unary relations that define the set of well-formed objects of the antecedent type. As we described in (Götz and Meurers 1995), this view opens up the possibility to compile calls to the constraint solver into the code, thus eliminating the run-time overhead of computing where to apply implicational constraints. An important optimisation to this approach is the application of lazy constraint unfolding in the sense of Aït-Kaci et al. (1993). In (Götz and Meurers 1996), we showed how the effects of lazy unfolding can be achieved by a special compilation technique, thus avoiding the on-line overhead that laziness usually incurs. The mathematical properties (correctness) of the lazy approach are investigated in (Götz 1995).
A further step was taken by Gerdemann (1995b), who showed how to compile typed feature structures into Prolog terms so that Prolog unification can be used for feature structure unification. Replacing explicit calls to a unifier with Prolog unification is the optimum solution for a Prolog system. Only compilation to a lower level (abstract machine code) can improve on this.
We showed both theoretically and practically that a unified treatment of implicational constraints and definite clauses is possible. Our implementation demonstrates that the use of advanced compilation techniques allows the efficient handling of large grammars.

2.2 Linear precedence constraints

We followed two lines of inquiry for LP constraints. The first one was to integrate (Manandhar 1995b)'s work with our feature logic. The basic insight of Manandhar was that *set constraints* (Manandhar 1993; 1995a) in feature logic can be used to model LP constraints. This is achieved, e.g., by providing any node a with a set-valued feature *prec* that contains all the nodes that a precedes. A natural way to view such a feature is as a relation between nodes. If it is to encode precedence, then

[2]This is only the case if we allow no variables in the scope of negation.

it must be transitive and irreflexive. Manandhar's constraint language is powerful enough to be able to encode these restrictions.
We extended Manandhar's work, which is all in the context of an untyped logic, to work with our type system. (Alt 1997) proved decidability of satisfiability for this extended logic and provided a constraint solver. This allows the specification of LP statements in an arbitrary constraint grammar, as no reference to nodes in a tree is needed. Just as the grammar writers can freely choose an encoding of grammatical structures, they can also choose which objects need to stand in a precedence relation, and which don't.
A different line of research was pursued in the B8 project. (Richter and Sailer 1995) showed how LP constraints can be defined within the logic of (King 1989), without the need for extensions such as set-valued features. This method was then used in the Fragmentgrammatik and implemented in ConTroll. This, however, turned out to be very slow and led to an integration of work on chunk parsing (Abney 1991) from the B7 project. A chunk parser was implemented that runs as a preprocessor on the input string to provide a first, shallow analysis. This practical work then found its way back into theory in (Richter and Sailer 1996) and (Sailer 1997), where chunks are related to topological fields.
Our results for linear precedence are mixed. On the theoretical side, we were able to show that that an interpretation of LP constraints within HPSG grammars is possible. This is born out by our results concerning the integration of Manandhar's work with our own, and the Fragmentgrammatik, a linearisation grammar with no phrase structure backbone.
On the practical side, however, it became clear that it is a mistake to treat LP constraints on a par with any other constraint. To guarantee efficient parsing, the processing of linearisation and dominance constraints must be tightly interleaved and controlled by a special processor. Otherwise, too much speculative and redundant computation is performed. We will address this problem in the next phase of our project, where we will compile linearisation grammars down to abstract machine code, on which level this sort of tight interleaving can be much easier achieved.

2.3 Lexical Rules

In the course of the 95–97 phase of the B4 project we developed a new computational treatment of lexical rules which, instead of expressing relations between lexical entries, encodes possible variations resulting from lexical rule application inside of the entries. We establish a formal link between the lexical rule mechanism and definite relations encoding covariation in lexical entries. The interaction of a set of lexical rules and the transfer of all properties unchanged by a lexical rule are automatically deduced from the set of lexical rules provided.
A compiler was developed that translates a set of lexical rules and their interaction into definite relations constraining lexical entries. We show that the definite relations produced by the compiler can be refined by program transformation techniques to

increase run-time efficiency without losing an independent representation of lexical rules. In addition the compiler adapts the lexicon such that lexical entries directly bear all specifications not changed by the lexical rules. This permits delayed evaluation of lexical covariation, i.e., "on the fly" application of lexical rules, which avoids expanding out the lexicon.

The conception of lexical rules underlying our research makes it possible to deal with the full lexical rule mechanism within the feature logic for HPSG proposed in (King 1989, 1994). In (Meurers 1995) we provided a formal semantics of lexical rules upon which our computational treatment is based. Due to this theoretical foundation, the computational treatment of lexical rules proposed can be seen as an extension to the principled method behind the ConTroll compiler for encoding the main building block of HPSG grammars – the implicative constraints – as a logic program, which is described in Sec. 2.1.

The computational treatment of lexical rules described in (Meurers and Minnen 1995a, 1995b) consists of four compilation steps which translate a set of lexical rules as specified by the linguist and their interaction into definite relations to constrain lexical entries. The way the four compilation steps interconnect is illustrated in figure 4. The first compilation step translates lexical rules into a definite clause representation and derives for each lexical rule a predicate that ensures transfer of properties that remain unchanged. This idea to preserve unchanged specifications can be considered an instance of the well-known unsolved *frame problem* in AI (McCarthy and Hayes 1969). The use of *appropriateness*, a set of declarations which specify which features can and must have values for every given type, however, renders the frame problem solvable for our particular logic. In the second compilation step, we determine the possible interaction of the lexical rules. This results in a finite state automaton representing global lexical rule interaction, i.e., the interaction of the lexical rules irrespective of the lexicon. Figure 5 illustrates the ConTroll representation of global lexical interaction using the visualisation of compiler graphs (VCG) package. In the subsequent step of abstract lexicon expansion this finite state automaton is fine-tuned for each of the natural classes of lexical entries in the lexicon. In the fourth compilation step these automata are translated into definite relations and the lexical entries are adapted to call the definite relation corresponding to the automaton fine-tuned for the natural class to which they belong. The reader is referred to (Minnen in prep.) for a more elaborate description of the compiler as a sequence of program transformations.

In (Meurers and Minnen 1996) we discuss in detail, how the general logic programming technique of constraint propagation (le Provost and Wallace 1993; Marriott, Naish, and Lassez 1988) which we only described briefly in the original presentation of the lexical rule compiler can be used to advance the covariation encoding of lexical rules such that an efficient on the fly application of lexical rules is possible.

Finally, further work in the context of the B4/B8 projects in (Hinrichs and Nakazawa 1996a, 1996b) addresses another aspect of the interpretation of lexical rules. The question discussed in these papers is whether the application criterion of lexical rules is a subsumption or a unification test. While the latter case is less restrictive and used

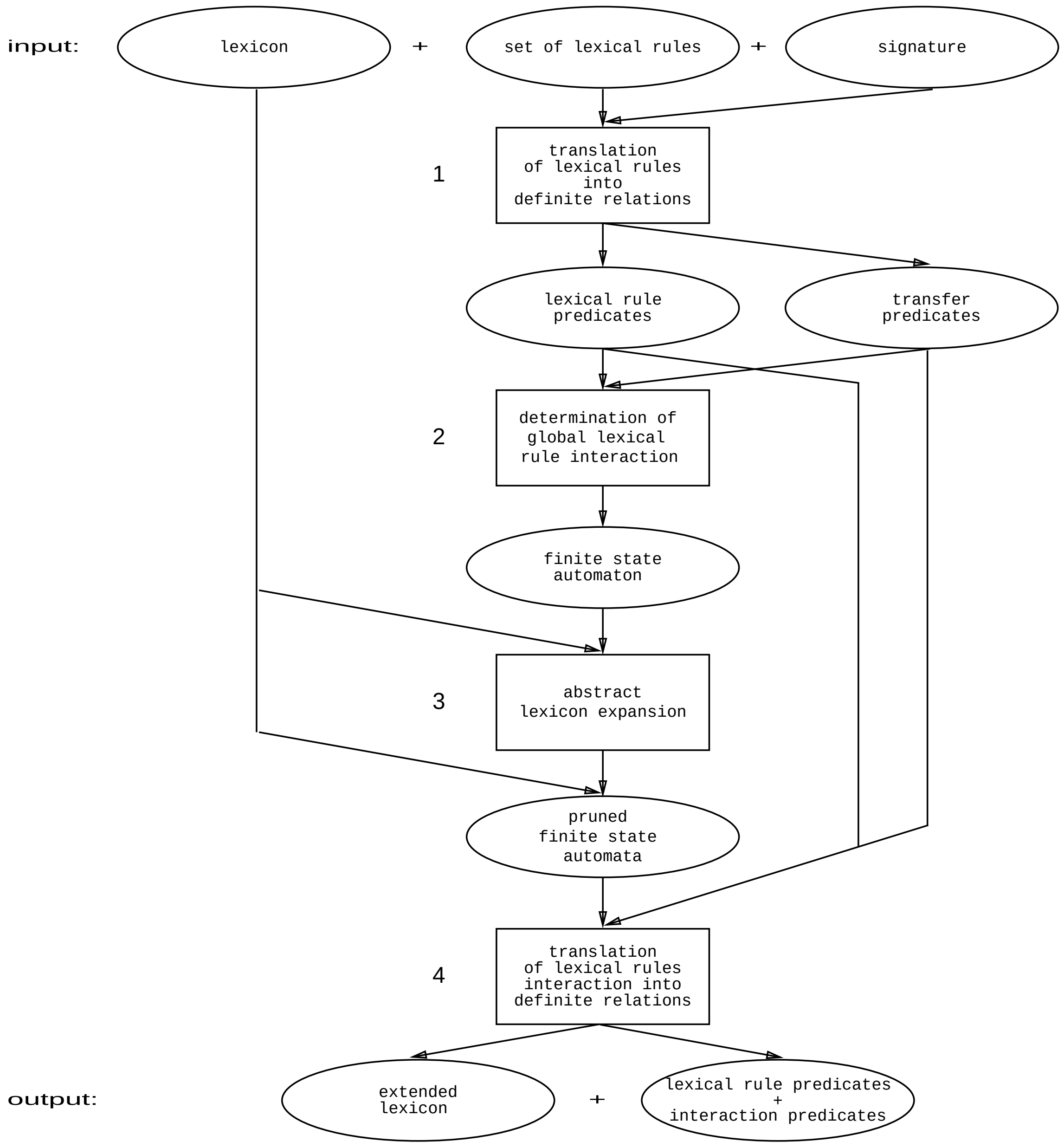

Abbildung 4: Overview of the lexical rule compiler

in many current HPSG theories, Hinrichs and Nakazawa show that using unification as application criterion can lead to incorrect results.

The computational treatment of lexical rules was integrated with the ConTroll system. Test results on a large grammar implementing a complex HPSG theory covering the, so-called, aux-flip phenomenon and partial-VP topicalisation in the three clause types of German (Meurers 1994) show that parsing with a covariation encoding of

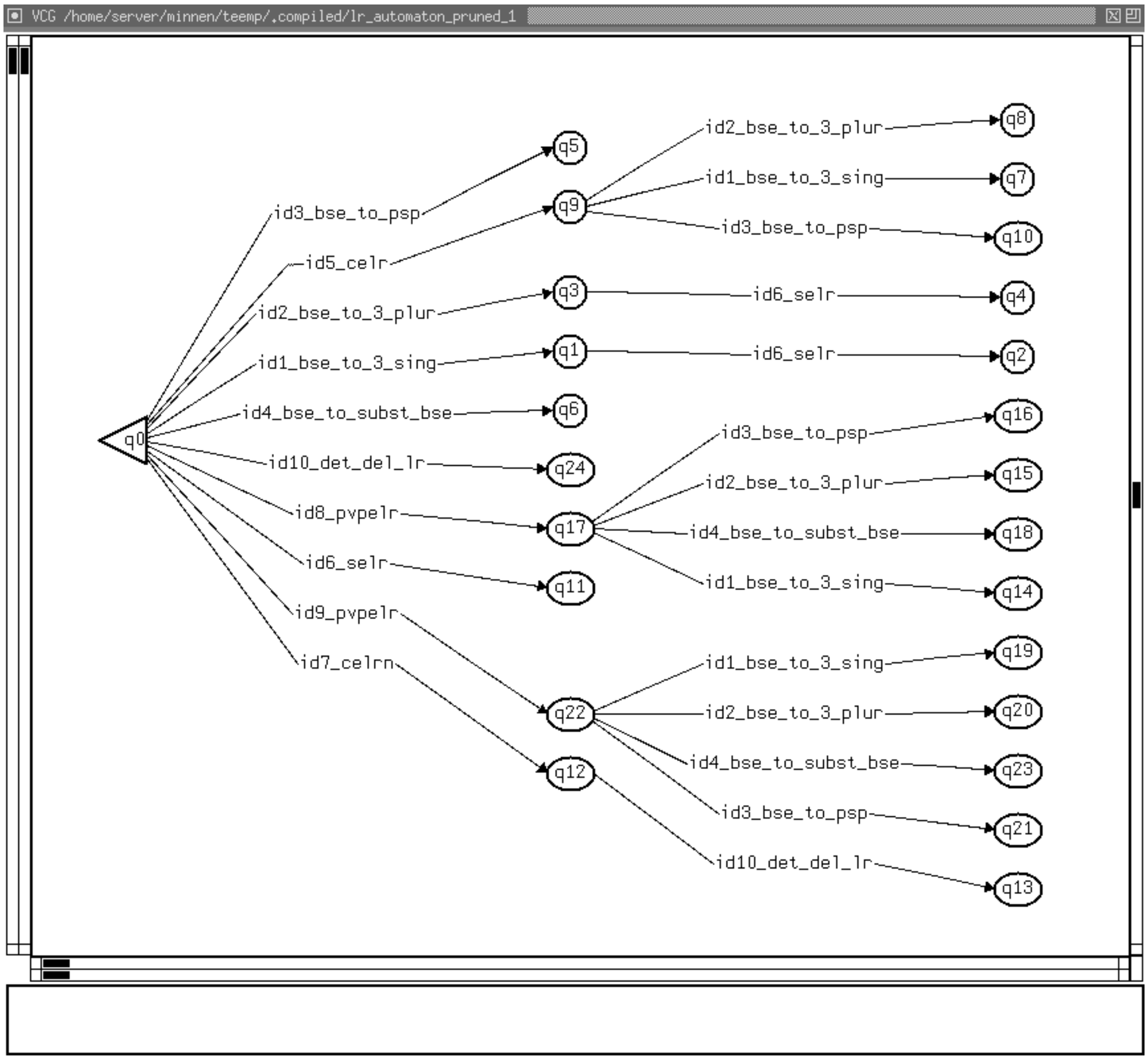

Abbildung 5: VCG display of finite state automaton representing global lexical rule interaction

lexical rules leads to parsing times which are in many cases better than parsing with a lexicon that was expanded off-line, although the covariation encoding is many times smaller than the fully expanded lexicon.

2.4 Top-down interpretation

As described above, grammars are compiled into intermediate code that can be viewed as a constraint logic program over typed feature logic. To run the generated code, we have investigated several options.

1. Translation to CUF. We built a translator to generate a CUF program from the intermediate code and the type hierarchy specification. This works well for small examples, but has turned out not to be feasible for realistic grammars. The reason is that the CUF programs generated by the translator a simply too big for the CUF compiler to handle in a reasonable time.

2. Direct interpretation. For the ConTroll system, we wrote a Prolog interpreter that directly emulates the intermediate code. We successfully applied the following techniques:

   (a) Goal delaying. The interpreter delays the execution of goals with insufficiently instantiated arguments. The user defines wait-declarations to tell the interpreter what "sufficiently instantiated" is.

   (b) Andorra principle. Similar to CUF, we apply the basic ideas of the Andorra model (cf. (Haridi and Janson 1990) and references cited therein). The interpreter always executes determinate goals before non-determinate ones. We consider determinacy only with respect to head unification: a goal is recognised to be determinate if there is at most one clause head that unifies with it. This evaluation strategy has two advantages: it reduces the number of choice points to a minimum, and it leads to early failure detection. In our implementation, the overhead of determining which goals are determinate has turned out to be by far outweighed by the reduction in search space for our linguistic applications.

   (c) Deep indexing. To optimise clause selection, the user may specify arbitrarily many "points of indexing" for each relation/type. Other than in Prolog, which uses indexing of the outermost functor of the first argument, we allow indexing on any argument at an arbitrary depth. This way, the linguists may organise their data according to linguistic criteria, and not according to processing ones.

3. Compilation to C (Zahnert 1997). We developed a compiler for a proper subset of ConTroll that takes intermediary code and compiles it to C. The generated C program is then compiled with a standard C compiler such as gcc and linked with a run-time environment to form an executable. The compiler is incomplete in that it does not follow the interpreter's use of the Andorra model, and in that it provides no indexing functionality. It does, however, do coroutining with delays. The fact that it doesn't use the Andorra model means that a program compiled with the compiler will in general need to do more speculative computation, thus increasing the search space.

   Independently, Shalom Wintner (the Technion, Haifa, Israel) developed an abstract machine based compiler for a subset of ALE (Carpenter and Penn 1994), a system closely related to ConTroll. While researching his ideas, Wintner spent half a year at the University of Tübingen on a MINERVA grant as a guest of our project.

The combination of goal delaying, Andorra, and indexing proved to be an efficient way to process even with large grammars. Particularly the omission of either the Andorra principle or goal delaying invariably led to poor performance or non-termination for the grammars we tested.

2.5 Bottom-up Interpretation

Magic compilation is a general technique for more efficient bottom-up interpretation of logic programs. Given a logic program, magic compilation produces a new program in which the filtering as normally resulting from top-down interpretation is explicitly characterised through, so-called, *magic predicates*, which produce variable bindings for filtering when interpreted bottom-up. The original rules of the program are extended such that these bindings can be made effective.
We faced two important problems:

> **Efficiency**. We knew that bottom-up interpretation, by itself, was unlikely to lead to efficient parsing. Though magic compilation significantly improves goal-directedness given dynamic bottom-up interpretation, the cost of tabling items can deluge the advantage of increased goal-directedness, especially in the case of large grammars.
>
> **Termination**. Despite a number of advantages over Earley deduction, we knew that bottom-up interpretation of magic-compiled grammars has its difficulties with particular types of recursion that constitute problems also for Earley deduction.

In the current phase of project B4 we developed a *magic* component for the ConTroll system: a dynamic bottom-up/top-down interpreter that uses selective magic compilation on the basis of so-called, *parse types* and a preselection of relevant lexical entries to increase parsing efficiency. Figure 6 gives an overview of the magic component of the ConTroll system.
Parse types are used to allow the user to determine how the constraints in the grammar are to be interpreted. The underlying idea is that some constraints are best interpreted bottom-up, whereas others are best interpreted top-down. More specifically, we assumed that given the high level of lexicalisation of the fragment grammar, constraints on the type *word* and other constraints on subtypes of *sign* increase the goal-directedness of the parsing process when interpreted bottom-up, whereas relational constraints often suggest a top-down interpretation.
Magic compilation is applied only on parse types constraints. As a result, parse types are also used to avoid inefficiency resulting from tabling too many intermediate results. At run time parse types and the corresponding magic constraints are interpreted bottom-up, while the remaining constraints are interpreted top-down.
Parse type constraints (and the corresponding magic constraints derived using magic compilation) are interpreted using an extension of *semi-naive* interpretation (Ramakrishnan, Srivastava, and Sudarshan 1992). Semi-naive interpretation is a bottom-up

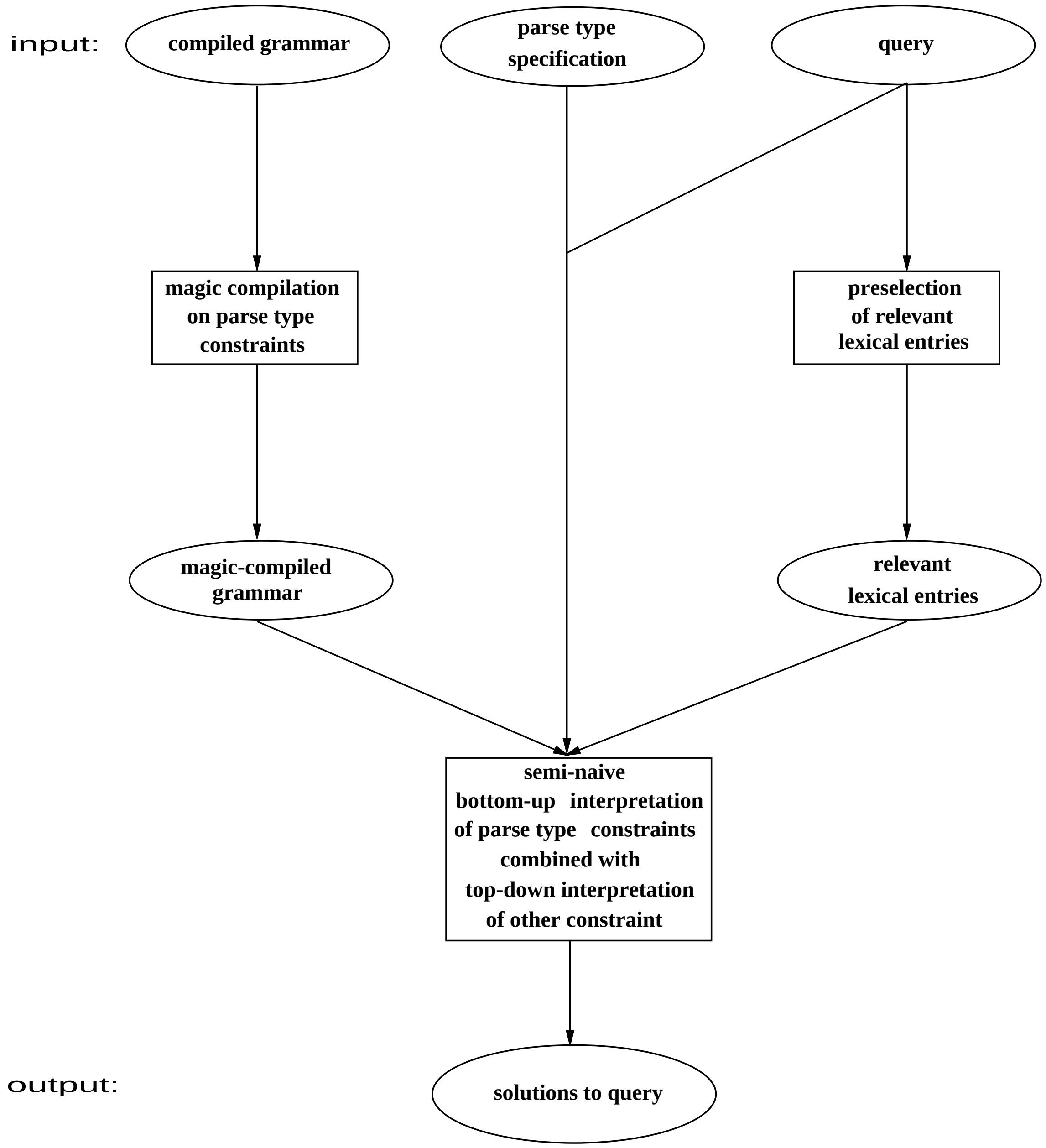

Abbildung 6: Overview of the ConTroll magic component

processing strategy which tables passive items, i.e., complete intermediate results.[3] Semi-naive interpretation blocks repeated derivation of items from the same earlier derived items as in of *naive* bottom-up interpretation (Bancilhon 1985).
Non-parse type constraints are interpreted top-down as discussed in Sec. 2.1. Given

[3]As shown in (Minnen in prep.) magic compilation does not depend on this particular type of bottom-up interpretation.

the delaying mechanism used for top-down interpretation, it is possible that not all sub-goals of a non-parse type goal can be interpreted locally, but remain *delayed* because they are not sufficiently instantiated when called. To deal successfully with these delayed goals, semi-naive interpretation is extended with the possibility to store passive items with delayed goals and a generalisation of the operation to combine these items similar to proposals in (Johnson and Dörre 1995).

In addition to magic compilation we increase the goal-directedness of the magic interpreter by means of using the input string to restrict the number of items that are added to the table during initialisation: Only those lexical entries that have a value for their phonology feature that appear in the input string are tabled initially.

As shown in (Minnen 1996) the explicit characterisation of filtering resulting from magic compilation offers other possibilities for further increasing processing efficiency. Using the program transformation technique of *unfolding*[4] (Tamaki and Sato 1984) one can eliminate redundant filtering steps thereby mimicking different more efficient processing strategies.

With respect to the termination problems referred to above, Minnen (1996) showed that it is possible to make filtering less specific by means of *off-line abstraction*, for example, by means of imposing a particular term-depth to the magic predicates in the transformed grammar. Just as in Earley deduction, abstraction (also called *restriction*) ensures termination for a large class of grammars. The compile-time application of an abstraction function allows the application of a subsumption check in a static, i.e., off-line, fashion, thereby avoiding the run-time overhead of such an expensive check on derived items before they are tabled.

Magic compilation has thus proven to be a very promising extension of bottom-up interpretation. It allows bottom-up filtering that achieves a goal-directedness which corresponds to dynamic top-down interpretation with abstraction and subsumption checking. Filtering is incorporated into the logic underlying a grammar as a result of its explicit characterisation as magic predicates. For a large class of grammars abstraction and subsumption checking can be performed off-line thereby allowing for more efficient processing. Furthermore, it enables a reduction of the number of edges that need to be stored through unfolding magic predicates.

Compared to top-down interpretation as discussed in Sec. 2.1 mixed bottom-up/top-down interpretation of magic-compiled ConTroll linearisation grammars sometimes performs significantly worse and sometimes significantly better. This is partly due to the fact that the intricate interplay of top-down and bottom-up interpretation restricts the number of possible orderings in which constraints in the grammar can be interpreted by presupposing a strict order between type constraints and relational constraints. This can possibly be improved upon by relaxing this restriction in a more sophisticated processor.

As shown in (Minnen, Gerdemann, and Götz 1995) and (Minnen, Gerdemann, and Hinrichs 1996) the ordering of constraints in a grammar can be of extreme import-

[4]This program transformation technique is sometimes also referred to as *partial execution*. See, for example, (Pereira and Shieber 1987).

ance and a more thorough comparison between the sketched bottom-up/top-down interpreter and the top-down interpreter would involve a refinement of the former as to the order in which constraints can be interpreted.

2.6 Extraction of phrase structure

The extraction of phrase structure turned out to be a theoretically and practically difficult problem (Aldag 1997). Consider the following constraint.

$$\begin{aligned} \textit{headed-phrase} \Rightarrow\ & \textsc{phon}{:}Z \wedge \\ & \textsc{head-dtr}{:}\textsc{phon}{:}X \wedge \\ & \textsc{comp-dtr}{:}\textsc{phon}{:}Y \wedge \\ & \mathrm{concat}(X, Y, Z) \end{aligned}$$

It could be said to encode a phrase structure rule, since it concatenates the PHON values of the two daughters to yield the PHON value of the mother. Some important assumptions are made here: we know where the string information of categories is to be found (i.e., under the PHON attribute), and we know that the concat-relation encodes string concatenation. That these assumptions are non-trivial is confirmed by the fact that it is undecidable if a given definite relation encodes string concatenation or not.

Aldag (1997) gave precise conditions under which the extraction of a phrase structure backbone is possible for a ConTroll constraint grammar. However, the fragment grammar developed in the B8 project did not have an inducible phrase structure backbone (see the B8 Ergebnisbericht). We thus needed to rely on top-down and bottom-up processing as described above.

2.7 Grammar writing environment

Our experience from the last phase had shown that the development of a large grammar is very difficult without some tools to support the grammar writer. This is particularly true if the grammar is not developed by an individual, but by a whole group of people. Since the B8 project was going to develop a large grammar in ConTroll, we decided early on to supplement our theoretical work with practical tools to aid the grammar writers.

2.71 The Xtroll graphical user interface

The Xtroll GUI programmed by Carsten Hess allows the user to interactively view AVMs, search attributes or values in those representations, compare two representations (e.g. multiple results to a query) and highlight the differences, etc. Fonts and Colours can be freely assigned to the attributes and types. The displayed structures (or any part of it) can be can be printed or saved as postscript file. The GUI

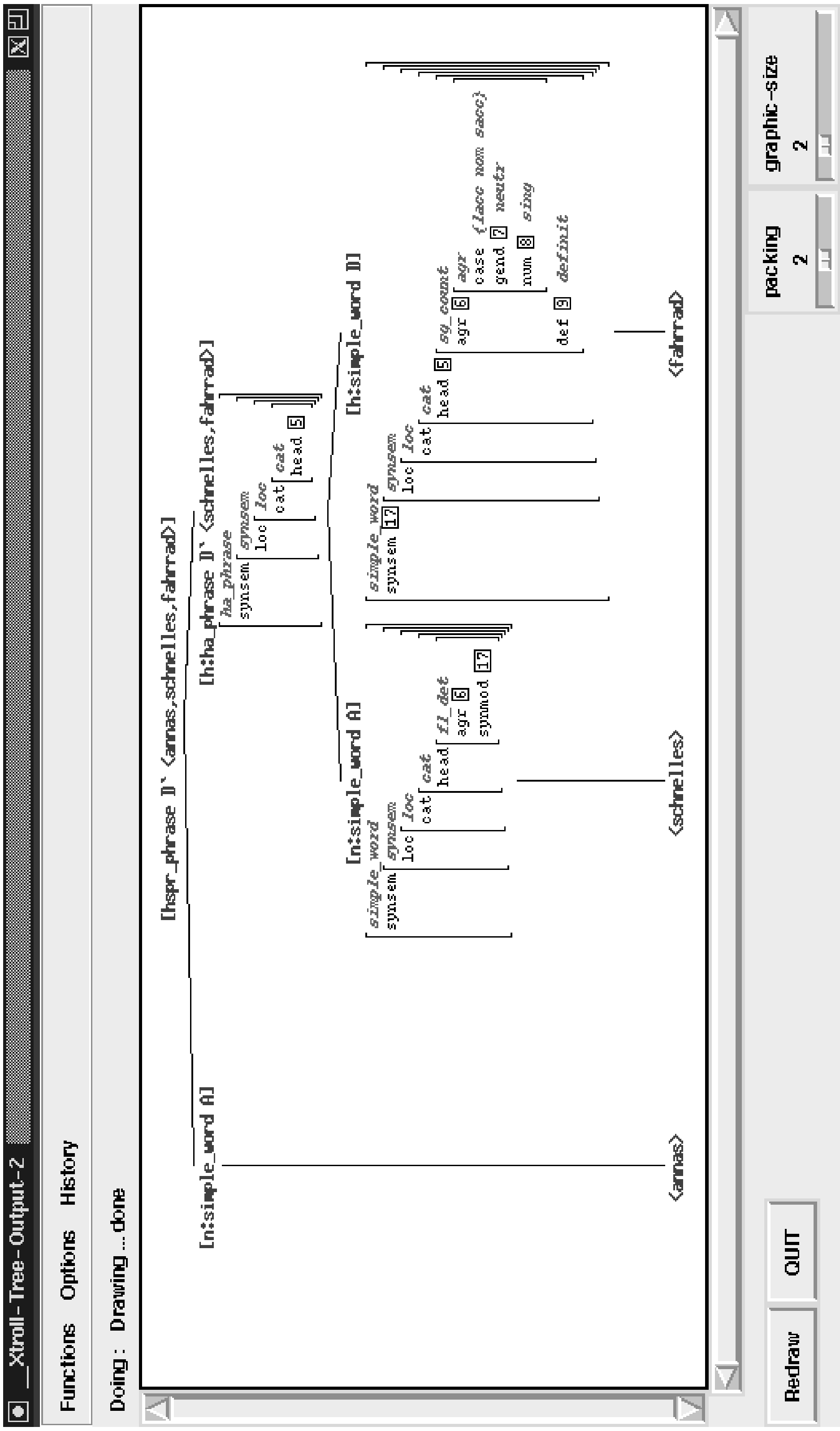

Abbildung 7: A screen shot of a window showing user configured tree output.

comes with a clean backend interface and has already been used as front end for other natural language applications, e.g., in the VERBMOBIL project.
A special feature of Xtroll is that it offers a mechanism for displaying feature structures as trees according to user specified patterns. Note that displaying trees is not an obvious display routine in ConTroll, since the system does not impose a phrase structure backbone but rather allows a direct implementation of HPSG grammars which usually encode the constituent structure under DTRS or some similar attribute. Since trees are a very compact representation allowing a good overview of the structure, Xtroll allows the user to specify that certain paths under a type are supposed to be displayed in a tree structure. As labels for the tree nodes, Xtroll can display a user definable selection of the following: the feature path to the node, the type of the structure, the phonology, and finally an abbreviation resulting from matching user specified feature structure patterns. An example for such a tree output is shown in Figure 7. Here the abbreviations were used to display category information in an X-bar fashion. Clicking on the labels displays the AVM associated with this node.
The use of the fully user definable, sophisticated display possibilities of Xtroll in our experience have turned out to be indispensable for developing large typed feature based grammars.

2.72 A graphical debugger

We provide a sophisticated tracing and debugging tool which was developed to allow stepwise inspection of the complex constraint resolution process.
The debugger displays the feature structure(s) to be checked for grammaticality and marks the nodes on which constraints still have to be checked. As a result of the determinacy check, each such node can also be marked as failed, delayed or deterministic. Similar to standard Prolog debuggers, the user can step, skip, or fail a constraint on a node, or request all deterministic processing to be undertaken. An interesting additional possibility for non-deterministic goals is that the user can inspect the matching defining clauses and chose which one the system should try.
For example in figure 8, the selected goal with tag $\boxed{1}$ is listed as delayed and is displayed at the bottom to have two matching defining clauses out of seven possible ones. Using the mouse, the user can chose to display the matching or all defining clauses in separate windows.
We believe that the availability of a sophisticated debugger like the one implemented for the ConTroll system is an important prerequisite for large scale grammar development.

2.73 Automatic macro detection

When writing a typed feature structure based grammar one usually wants to abbreviate often used feature paths or complex specifications. In ConTroll this can be done using the definite clause mechanism. However, from a processing point of view, it is

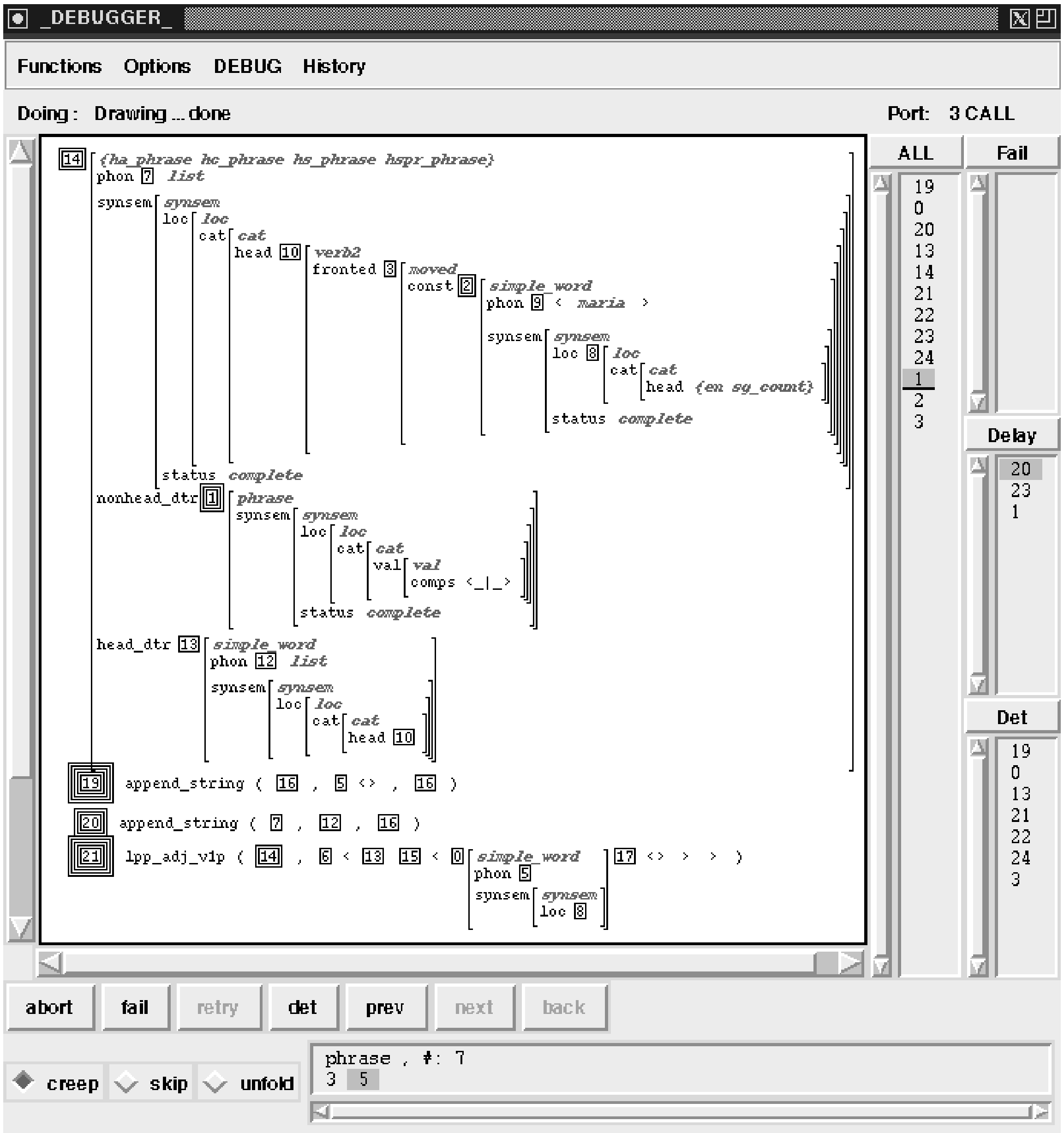

Abbildung 8: A screen shot of the graphical debugger in action

inefficient to treat macros in the same way as ordinary relations. We thus implement a fast, purely syntactic preprocessing step that finds the relations that can be treated as macros, i.e., unfolded at compile time. These macro relations are then compiled into an internal representation during a second preprocessing step. When the rest of the grammar is parsed, any macro goal will simply be replaced by its internal representation.
Obviously, making automatic macro detection a property of the compiler relieves the grammar developer from the burden of distinguishing between macros and relations,

thereby eliminating a potential source of errors.

2.74 Automatic macro generation

Since HPSG theories usually formulate constraints about different kind of objects, the grammar writer usually has to write a large number of macros to access the same attribute, or to make the same specification, namely one for each type of object which this macro is to apply to. For example, when formulating immediate dominance schemata, one wants to access the VFORM specification of a *sign*. When specifying the valence information one wants to access the VFORM specification of a *synsem* object. And when specifying something about non-local dependencies, one may want to refer to VFORM specifications of *local* objects.
ConTroll provides a mechanism which automatically derives definitions of relations describing one type of object on the basis of relations describing another type of object – as long as the linguist tells the system which path of attributes leads from the first type of object to the second.
For a large grammar, which usually specifies hundreds of macros, this mechanism can save a significant amount of work. It also provides a systematic rather than eclectic way of specifying abbreviations in a grammar, which is vital if several people are involved in grammar development.

3 Vergleiche mit Arbeiten außerhalb des Sonderforschungsbereichs und Reaktionen der wissenschaftlichen Öffentlichkeit auf die eigenen Arbeiten

There are quite a number of typed feature systems available today, among them ALE (Carpenter and Penn 1994), CUF (Dörre and Dorna 1993) and TFS (Emele and Zajac 1990; Emele 1994). What distinguishes ConTroll from all other systems is its type system, and the use of delaying and the Andorra principle to *interleave* the execution of relations and type constraints in a principled fashion. For large constraint grammars such as the B8 fragment grammar, these have proven to be necessary prerequisites for precise and efficient execution. These ideas can be added to any of these other systems, and would result in a similar improvement for them.
ConTroll's type system exactly fits the requirements of HPSG, as formalised by King (1989). Crucial to this system is its interpretation of *appropriateness*, a set of declarations which specify which features can have values for which types. ConTroll and ALE are the only systems that interpret appropriateness with *total well-typing*, which states that every feature structure must have values for all and only the features appropriate to its type. In addition to being better suited to HPSG, this leads to a variety of practical benefits, from the ability to perform strong inferencing on

types in user descriptions, to more efficient internal encodings of feature structures, to, in the case of lexical rules, providing the ability to overcome the frame problem and perform default reasoning on feature transfer.

ConTroll, unlike ALE, also enforces the exhaustive type discipline of King (1989), which states, in the case of HPSG's type hierarchy, that every non-maximal type's objects are partitioned by its immediate subtypes. This allows the user to encode HPSG type hierarchies directly in ConTroll without modifying the appropriateness conditions to account for feature co-occurrence restrictions implicit in the signature. No other feature-based system uses this interpretation except TFS; and unlike TFS, ConTroll uses a combination of compile-time expansion and lazy type inferencing to enforce this interpretation with a minimum of run-time overhead (and in practical cases, none).

TFS also offers type constraints and relations. It has some serious drawbacks, however. Unlike ConTroll, TFS does not allow universal principles with complex antecedents, but only type constraints. And the system does not include a delay mechanism, so that it is often impossible to ensure termination or efficient processing. CUF also does not offer universal principles. Otherwise, its definite clause component is very similar to ConTroll's, in that both use delay statements and preferred execution of determinate goals.

ALE does provide a simple form of type constraints, but their unfolding is neither lazy, nor can their execution be controlled by the user in any way. This leads to severe termination problems with recursive constraints. On the other hand, ALE's type constraints are an extension of the underlying phrase structure component, and were never designed to play the fundamental role that constraints play in ConTroll. ALE also has a definite clause language, similar to ConTroll's. Only ConTroll, however, provides both constraints and relations, plus a principled mechanism for interleaving execution of the two. Without this, the effective use of general implicational constraints, the language of HPSG's principles, is simply impossible.

ALE also provides inequational descriptions, extensional types and an abstract machine with features not found in Zahnert (1997), such as built-in chart parsing, all of which are quite useful for linguistic applications. In the next phase of the B4 project, we will integrate the complete functionality of ALE with ConTroll.

Our approach to bottom-up/top-down interpretation is highly innovative in that we use selective magic compilation for filtering in parsing with typed feature structure grammars. Johnson and Dörre (1995) provides a means of memoizing with coroutined constraints, which allows a mixed top-down/bottom-up approach; but to our knowledge, there is no other work on the use of magic compilation for the special case of natural language processing.

The discussion on the formal foundation and implementation of lexical rules was mainly advanced by the presentations given at the ACQUILEX II workshop entitled "The formalisation and use of lexical rules" (August 9–11, 1995). In particular, Carl Pollard and Mike Calcagno of the Ohio State University provided an interesting alternative conception of lexical rules. While we formalise lexical rules as descriptions relating *word objects* (Meurers 1995), Pollard and Calcagno propose a more traditional formalisation of lexical rules as as meta-descriptions relating *lexical entries* (Calcagno 1995). The consequent invitation of both researchers as guest scientists of the SFB 340 made it possible to compare the two formalisations and led to (Calcagno, Meurers, and Pollard in prep.), in which the different ontological assumptions and their consequences are discussed.

4 Offene Fragen

Further questions raised by the research in this phase can mainly be located in two areas:

- Linearisation grammars
- Low level compilation

The notion of linearisation grammars that evolved during this project phase (in the fragment project B8 and elsewhere) abandons the close connection between linear order and hierarchical structure still present in the ID/LP format. This puts new and strenuous demands on any processing strategy, since the parsing problem for such grammars is inherently difficult. It is an open problem at this point how such grammars can be handled most efficiently.
The work on low level, abstract-machine-based compilation for constraint grammars has only just started. The compilers for a subset of ConTroll developed in our own project and the one developed for a subset of ALE by Wintner are steps in the right direction, but many issues remain to be explored further:

- better type and appropriateness compilation
- better memory management, particular better maintenance of the parsing chart
- integrating normal logic programming control with parsing
- better indexing methods for lexicon and parsing chart
- parallel processing of abstract machine code

Both areas, linearisation and low-level compilation, will be addressed during the next phase of our project.

5 Veröffentlichungen, Vorträge und sonstige Aktivitäten

5.1 Veröffentlichungen

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

Alt, N. (1997). A typed feature logic with set-valued attributes as a foundation for LP rules. Master's thesis, Seminar für Sprachwissenschaft, University of Tübingen.

Baker, K. and G. Penn (1997). An ALE Implementation of a Head-Driven Phrase Structure Grammar for English. Language Technologies Institute Technical Report, Carnegie Mellon University.

Calcagno, M., D. Meurers, and C. Pollard (in preparation). Lexical Rules in HPSG. Ms., SFB 340, Universität Tübingen.

Carpenter, B. and G. Penn (1996). Compiling Typed Attribute-Value Logic Grammars. In H. Bunt and M. Tomita, eds., *Recent Advances in Parsing Technology.* Kluwer.

Carpenter, B., G. Penn, and O. Popescu (1997). The Attribute Logic Engine (ALE) 3.0 User's Guide (distributed with Prolog software, HPSG Grammar, and Syllabification Grammar). Language Technologies Institute Technical Report, Carnegie Mellon University.

Gerdemann, D. (1995a). Open and closed world types in NLP systems. In *Proceedings of 5. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft.*

Gerdemann, D. (1995b). 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, October). *Troll manual* (draft ed.). Seminar für Sprachwissenschaft, Universität Tübingen.

Götz, T. (1995). Compiling HPSG constraint grammars into logic programs. In *Proceedings of the joint ELSNET/COMPULOG-NET/EAGLES workshop on computational logic for natural language processing*, Edinburgh.

Götz, T. (in prep.). *Feature Constraint Grammars.* Ph. D. thesis, Universität Tübingen.

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

Götz, T. and W. D. Meurers (1996). The importance of being lazy - using lazy evaluation to process queries to HPSG grammars. In P. Blache (Ed.), *Actes de la troisième conférence anuelle sur le traitment automatique du langage naturel.*

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.

Hinrichs, E. and T. Nakazawa (1996a). Applying Lexical Rules under Subsumption. In *Proceedings of the* HPSG/TALN-*96 International Conference*, Marseille, France.

Hinrichs, E. and T. Nakazawa (1996b). Applying Lexical Rules under Subsumption. In *Proceedings of the 16th Conference on Computational Linguistics*, Denmark.

Meurers, D. and G. Minnen (1995a). A Computational Treatment of HPSG Lexical Rules as Covariation in Lexical Entries. In *Proceedings of the 5th International Workshop on Natural Language Understanding and Logic Programming*, Lisbon, Portugal.

Meurers, D. and G. Minnen (1995b). The Covariation Approach as Computational Treatment of HPSG Lexical Rules. In *Proceedings of the* AQUILEX *II Workshop on Lexical Rules.* Cambridge, UK.

Meurers, D. and G. Minnen (1996). Off-line Constraint Propagation for Efficient HPSG Processing. In *Proceedings of the* HPSG/TALN-*96 International Conference*, Marseille, France.

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

Meurers, W. D. (1994). On implementing an HPSG theory – aspects of the logical architecture, the formalisation, and the implementation of head-driven phrase structure grammars. In: Erhard W. Hinrichs, W. Detmar Meurers, and Tsuneko Nakazawa: *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation.* Arbeitspapiere des SFB 340 Nr. 58, Universität Tübingen.

Meurers, W. D. (1995). Towards a semantics for lexical rules as used in HPSG. In *Proceedings of the Conference on Formal Grammar*, Barcelona. Also in: Proceedings of the ACQUILEX II Workshop on Lexical Rules, 1995, Cambridge, UK.

Minnen, G. (1996). Magic for Filter Optimization in Dynamic Bottom-up Processing. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*, Santa Cruz, USA.

Minnen, G. (in preparation). *Off-line Compilation for Reversible Constraint-based Grammars.* Ph. D. thesis.

Minnen, G., D. Gerdemann, and T. Götz (1995). Off-line Optimization for Earley-style HPSG Processing. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*, Dublin, Ireland.

Minnen, G., D. Gerdemann, and E. Hinrichs (1996). Direct Automated Inversion of Logic Grammars. *New Generation Computing 14(2).*

Penn, G. (1996). On the Connection between Context-Free Grammars and Lambek Categorial Grammars. Laboratory for Computational Linguistics Technical Report, Carnegie Mellon University.

Penn, G. (1996). Multiple WH-Extraction and Clitic Placement in Serbo-Croatian. Linguistics Qualifying Paper, Carnegie Mellon University.

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

Penn, G. (in prep.). Animacy Contrasts in Slavic WH-Questions. To submit to *Linguistic Inquiry.*

Penn, G. (in prep.). Parametric Types for Typed Attribute-Value Logic. To submit to *Journal of Logic, Language, and Information.*

Penn, G. and O. Popescu (1997). Head-Driven Generation and Indexing in ALE. Language Technologies Institute Technical Report, Carnegie Mellon University.

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

5.2 Vorträge

10th March, 1995. Guido Minnen. Paper “Off-line Optimization for Earley-style HPSG Processing” (Co-authored by Dale Gerdemann and Thilo Götz). To the 7th Conference of the European Chapter of the Association of Computational Linguistics, Dublin, Ireland.

5th April, 1995. Thilo Götz. Paper “Compiling HPSG constraint grammars into logic programs”. To the Workshop on Computational Logic for Natural Language Processing, Edinburgh, Scotland.

23rd June, 1995. Detmar Meurers. Paper “A semantics for lexical rules as used in HPSG”. To the Int. HPSG Workshop. Tübingen.

28th June, 1995. Detmar Meurers. Paper “Compiling HPSG type constraints into definite clause programs” (Co-authored by Thilo Götz). To the 33rd Annual Meeting of the Association for Computational Linguistics, Cambridge MA, USA.

9th August, 1995. Guido Minnen. Paper “The Covariation Approach as Computational Treatment of HPSG Lexical Rules” (Co-authored by Detmar Meurers). To the ACQUILEX II Workshop on the Formalisation and Use of Lexical Rules, Cambridge, UK.

10th August, 1995. Detmar Meurers. Paper “A semantics for lexical rules as used in HPSG”. To the ACQUILEX II Workshop on the Formalisation and Use of Lexical Rules, Cambridge, UK.

13th August, 1995. Detmar Meurers. Paper “A semantics for lexical rules as used in HPSG”. To the First Formal Grammar Conference. Barcelona, Spanien.

March, 1996. Erhard Hinrichs. “When do Lexical Rules Apply.” Invited talk at the Center for Cognitive Science. Edinburgh, Scotland.

9th March, 1996. Gerald Penn. “Animacy Contrasts in Slavic WH-Questions.” To the Workshop on New Perspectives in HPSG, Carnegie Mellon University. Pittsburgh, USA.

22nd March, 1996. Dale Gerdemann. Talk “Term encoding of typed feature structure grammars”. Cognititve Science Seminar, Human Communications Research Center, Edinburgh University.

April, 1996. Erhard Hinrichs. “Applying Lexical Rules Under Subsumption.” Invited Talk at the Ohio State University. Columbus, USA.

May, 1996. Erhard Hinrichs. “Applying Lexical Rules Under Subsumption.” (Co-Author: Tsuneko Nakazawa). To the 3. Int. HPSG Conference. Marseille, France.

22nd May, 1996. Thilo Götz. Paper “The importance of being lazy – Using lazy evaluation to process queries to HPSG grammars” (Co-authored by Detmar Meurers). To the HPSG/TALN-96 International Conference, Marseille, France.

22nd May, 1996. Guido Minnen. Paper “Off-line Constraint Propagation for Efficient HPSG Processing” (Co-authored by Detmar Meurers). To the HPSG/TALN-96 International Conference, Marseille, France.

10th June, 1996. Gerald Penn. “Statistical Optimization in a Feature Structure Abstract Machine.” SFB 340 Kolloquium, Tübingen.

27th June, 1996. Guido Minnen. Paper “Magic for Filter Optimization in Dynamic Bottom-up Processing”. To the 34th Annual Meeting of the Association of Computational Linguistics, Santa Cruz, USA.

5th July, 1996. Detmar Meurers. Talk “Description-level Lexical Rules”. To the Int. Workshop “Lexical Rules in HPSG”

17th September, 1996. Gerald Penn. “The Attribute Logic Engine: Extending Prolog to Typed Feature Structures.” Computerlinguistik Seminar, Internationales Zentrum Sommerakademie, Tübingen.

August, 1996. Erhard Hinrichs. “Applying Lexical Rules Under Subsumption.” To the 16th International Conference on Computational Linguistics (COLING96). Copenhagen, Denmark.

September, 1996. Erhard Hinrichs and Paola Monachesi. “HPSG and the Syntax-Semantics Interface.” Tutorial at the East-West Conference on Computational Linguistics. Tübingen.

September, 1996. Erhard Hinrichs and Detmar Meurers (Organisers): “Issues of Grammar Formalization and Implementation.” Workshop at the East-West Conference on Computational Linguistics. Tübingen.

19th September, 1996. Gerald Penn. "Reconsidering the Multiply-Filled [Spec,CP] Hypothesis: Evidence from a Dialect of Serbo-Croatian." To the Workshop on Issues of Grammar Formalization and Implementation, Internationales Zentrum Sommerakademie, Tübingen.

20th September, 1996. Detmar Meurers. Talk "Implementing HPSG grammars". To "Issues of Grammar Formalization and Implementation" Workshop. East-West Conference. Tübingen.

October, 1996. Erhard Hinrichs and Paola Monachesi. "Head-Driven Phrase Structure Grammar. Tutorial at the KONVENS Conference, Bielefeld.

1st October 1996. Detmar Meurers. Demonstration of the Fragment Grammar in the ConTroll System. To the Int. Conference "Interfaces of Grammar". Tübingen.

1st October, 1996. Thilo Götz. "Logic and interpretation of ConTroll". To the Int. Conference "Interfaces of Grammar". Tübingen.

9th October, 1996. Frank Richter und Manfred Sailer. Demonstration of the Fragment Grammar in the ConTroll System. KONVENS 96. Bielefeld.

22nd October, 1996. Gerald Penn. "A Feature-based Extension to the Warren Abstract Machine." Invited talk at the Département d'Ingénierie Informatique, Université Catholique de Louvain. Louvain-la-Neuve, Belgium.

24th October, 1996. Gerald Penn. "Appropriateness and Type Inferencing for the Feature Structure Abstract Machine." Invited talk in the Programming Systems Colloquium, Universität des Saarlandes. Saarbrücken.

25th October, 1996. Gerald Penn. "An HPSG Analysis of Multiple WH-Extraction and Clitic Placement in Serbo-Croatian." Invited talk in the Computational Linguistics Colloquium, Universität des Saarlandes. Saarbrücken.

January, 1997. Erhard Hinrichs. "Applying Lexical Rules Under Subsumption." Invited talk at the Rijksuniversiteit Groningen. Groningen, The Netherlands.

January, 1997. Detmar Meurers. "Universelle Prinzipien und Relationale Constraints in einem getypten Featurelogiksystem." Invited talk in the Colloquium of the DYNALEX Project, SFB 282. Düsseldorf.

May, 1997. Gerald Penn. "Linearization and WH-Extraction in HPSG: Evidence from a Dialect of Serbo-Croatian." To the Slavic in HPSG Workshop, 30th Poznan Linguistic Meeting. Poznan, Poland.

June, 1997. Dale Gerdemann. "Introduction to Constraint-Based Natural Language Processing." To the Workshop on Topics in Constraint-Based Natural Language Processing. Summer School in Behavioral and Cognitive Neurosciences, University of Groningen, The Netherlands.

July, 1997. Gerald Penn. "Type and Appropriateness Compilation for Feature Structure Abstract Machines." To the Workshop on Topics in Constraint-Based Natural Language Processing. Summer School in Behavioral and Cognitive Neurosciences, University of Groningen, The Netherlands.

July, 1997. Guido Minnen. “Selective Magic HPSG parsing.” To the Workshop on Topics in Constraint-Based Natural Language Processing. Summer School in Behavioral and Cognitive Neurosciences, University of Groningen, The Netherlands.

July, 1997. Detmar Meurers. “ConTroll as a large grammar development platform.” To the Workshop on Topics in Constraint-Based Natural Language Processing. Summer School in Behavioral and Cognitive Neurosciences, University of Groningen, The Netherlands.

July, 1997. Thilo Götz. “Extending the Andorra-I precompiler to the typed feature structure case.” To the Workshop on Topics in Constraint-Based Natural Language Processing. Summer School in Behavioral and Cognitive Neurosciences, University of Groningen, The Netherlands.

July, 1997. Andreas Zahnert. “The Implementation of Freezing in a Run-time Environment for CLP(TFS) Programs.” To the Workshop on Topics in Constraint-Based Natural Language Processing. Summer School in Behavioral and Cognitive Neurosciences, University of Groningen, The Netherlands.

July, 1997. Thilo Götz. Paper “Interleaving universal principles and relational constraints over typed feature logic” (Co-authored by Detmar Meurers). To the 35th Annual Meeting of the Association for Computational Linguistics, Madrid, Spain.

August 1997. Erhard Hinrichs, Detmar Meurers und John Nerbonne. *Grammar Development in Constraint-Based Formalisms.* 2-week course at the ESSLLI Summer School. Aix-en-Provence. France.

5.3 Sonstige Aktivitäten

5.31 Projektübergreifende Aktivitäten

21st – 23rd June, 1995. Organisation of the Int. HPSG Workshop on the topics “Linearization and Complex Predicates, The Syntax–Semantics Interface, Issues of Computation and Formalization”. Tübingen.

10th – 13th March, 1996. Klausurtagung of the projects B4 and B8 “On the formalization of HPSG grammars and their implementation”. Oberjoch.

5th – 6th July, 1996. Organisation of the Int. Workshop on “Lexical Rules in HPSG”. Tübingen.

16th – 27th September, 1996. Organization of the East-West Conference on Computational Linguistics, incl. Workshop on “Issues of Grammar Formalization and Implementation”. Internationales Zentrum Programm, Universität Tübingen.

30th June – 4th July, 1997. Organisation of the Workshop on "Topics in Constraint-Based Natural Language Processing." Summer School in Behavioral and Cognitive Neurosciences, University of Groningen, The Netherlands.

5.32 Sonstige Aktivitäten von Projektmitgliedern

Erhard Hinrichs:

- Co-Chair, Program Committee (with Steve Abney), 7th Meeting of European Chapter of the Association of Computational Linguistics (EACL95), 1995.
- Member of the HPSG Standing Committee, since 1995.
- Head of the European Chapter of the Association for Computational Linguistics, 1995-1997.
- Vice-president of the Foundation for Language, Logic and Information (FOLLI), 1995-1997.
- President of the Foundation for Language, Logic and Information (FOLLI), since 1997.

Detmar Meurers:

- Member of the HPSG Standing Committee, since 1995.
- Editor of the "HPSG Gazette" Newsletter (with Adam Przepiórkowski), since 1996.

5.33 Gastwissenschaftler

Mike Calcagno. The Ohio State University, Columbus, USA.
15. June – 15. August 1996.

Carl Pollard. The Ohio State University, Columbus, USA.
15. June – 15. July 1996 und 15. June – 15. July 1997.

Literatur

Abney, S. (1991). Parsing by chunks. In R. Berwick, S. Abney, and C. Tenney (Eds.), *Principle-Based Parsing*. Kluwer Academic Publishers.

Aït-Kaci, H., A. Podelski, and S. C. Goldstein (1993). Order-sorted theory unification. Technical Report 32, Digital Equipment Corporation.

Bancilhon, F. (1985). Naive Evaluation of Recursively Defined Relations. In Brodie and Mylopoulos (Eds.), *On Knowledge Base Management Systems - Integrating Database and* AI *Systems*. Springer-Verlag.

Bresnan, J. (Ed.) (1982). *The Mental Representation of Grammatical Relations.* Cambridge, MA: MIT Press.

Calcagno, M. (1995). Interpreting lexical rules. In *Proceedings of the Conference on Formal Grammar*, Barcelona. Also in: Proceedings of the ACQUILEX II Workshop on Lexical Rules, 1995, Cambridge, UK.

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.

Copestake, A. (1992). *The Representation of Lexical Semantic Information.* Cognitive science research paper CSRP 280, University of Sussex.

Dörre, J. and M. Dorna (1993, August). CUF - a formalism for linguistic knowledge representation. In J. Dörre (Ed.), *Computational aspects of constraint based linguistic descriptions I*, pp. 1–22. Universität Stuttgart: DYANA-2 Deliverable R1.2.A.

Dörre, J. and A. Eisele (1991, January). A comprehensive unification based formalism. DYANA Deliverable R3.1.B, Universität Stuttgart.

Emele, M. C. (1994). The typed feature structure representation formalism. In *Proceedings of the International Workshop on Sharable Natural Language Resources*, Ikoma, Nara, Japan.

Emele, M. C. and R. Zajac (1990). Typed unification grammars. In *Proceedings of the* 13^{th} *International Conference on Computational Linguistics.*

Flickinger, D. (1987). *Lexical Rules in the Hierarchical Lexicon.* Ph. D. thesis, Stanford University, Stanford, USA.

Flickinger, D., C. Pollard, and T. Wasow (1985). Structure-sharing in lexical representation. In *Proceedings of the Twenty-Third Annual Meeting of the ACL*, Chicago, pp. 262–267. ACL.

Gerdemann, D. D. (1991). *Parsing and Generation of Unification Grammars.* Ph. D. thesis, University of Illinois.

Gerdemann, D. and P. J. King (1993). Typed feature structures for expressing and computationally implementing feature cooccurence restrictions. In *Proceedings of 4. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft*, pp. 33–39.

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. (1994). A normal form for typed feature structures. Arbeitspapiere des SFB 340, Universität Tübingen.

Haridi, S. and S. Janson (1990). Kernel Andorra Prolog and its computation model. In D. H. D. Warren and P. Szeredi (Eds.), *Proceedings of the seventh international conference on logic programming*, pp. 31–46. MIT Press.

Johnson, M. and J. Dörre (1995). Memoization of Coroutined Constraints. In *Proceedings of the 33rd Association for Computational Linguistics*, Cambridge, USA.

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.

le Provost, T. and M. Wallace (1993). Generalised Constraint Propagation over the CLP Scheme. *Journal of Logic Programming 10*.

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.

Marriott, K., L. Naish, and J.-L. Lassez (1988). Most Specific Logic Programs. In *Proceedings of 5th International Conference and Symposium on Logic Programmning*.

McCarthy, J. and P. Hayes (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In (Meltzer and Michie 1969).

Meltzer, B. and D. Michie (1969). *Machine Intelligence 4*. Edinburgh, Scotland: Edinburgh University Press.

Nilsson and Maluszynski (1995). *Logic Programming in Prolog* (Second ed.). Wiley.

Pereira, F. and S. Shieber (1987). *Prolog and Natural Language Analysis*. CSLI Lecture Notes, No. 10. Chicago, USA: Center for the Study of Language and Information.

Pereira, F. and D. H. D. Warren (1983). Parsing as Deduction. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Cambridge, *usa*.

Pollard, C. (1993). Lexical rules and metadescriptions. Handout for the International Conference of the SFB 340, October 5, 1993, Universität Stuttgart.

Pollard, C. and I. A. Sag (1987). *Information-based Syntax and Semantics, Vol. 1.* Number 13 in Lecture Notes. Stanford University: CSLI Publications. Distributed by University of Chicago Press.

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

Ramakrishnan, R., D. Srivastava, and S. Sudarshan (1992). Efficient Bottom-up Evaluation of Logic Programs. In J. Vandewalle (Ed.), *The State of the Art in Computer Systems and Software Engineering.* Kluwer Academic Publishers.

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.

Richter, F. and M. Sailer (1996). Word order and regions. Ausarbeitung zu einem Vortrag beim *International Center Workshop on Computational Linguistics*, 20.9.1996 Tübingen.

Sailer, M. (1997). Modifikation an einer unterspezifizierten Syntax-Semantik-Schnittstelle in HPSG. In (Meurers, Richter, Sailer, and Winhart 1997).

Tamaki, H. and T. Sato (1984). Unfold/Fold Transformation of Logic Programs. In *Proceedings of the 2nd International Conference on Logic Programming*, Uppsala, Sweden.