Towards a platform for linearization grammars

Frederik Fouvry and Detmar Meurers*

Seminar für Sprachwissenschaft
Universität Tübingen

1 The issue

A number of authors working in the HPSG paradigm (Reape, 1990, 1994, 1996; Richter and Sailer, 1995; Kathol, 1995; Müller, 1995, 1999a; Penn, 1999, and others) have argued on linguistic grounds that constituent structure should not be directly linked to word order properties. While the formal realizations of the proposals differ, all these so-called linearization approaches have in common that they allow constituents to be discontinuous, i.e., they do not cover a continuous stretch of phonological or graphemic material.[1] On the computational side, some algorithms for parsing grammars licensing such discontinuous constituents have been proposed (Johnson, 1985; Reape, 1991; van Noord, 1991). A computational encoding of an HPSG linearization grammar for German is discussed in Müller (1996). The BABEL system described implements the HPSG linearization theory of Müller (1999a) in Prolog and uses a basic parser for discontinuous constituents for processing with this grammar.

While the BABEL approach shows that an HPSG linearization theory can form the basis of a fairly wide-coverage and efficient implementation, the approach has the disadvantage of forcing the grammar writer to encode the typed feature logic HPSG theory directly at the low level of the Prolog first order terms. The system therefore is not well-suited for implementing, testing and comparing different HPSG linearization theories. On a related point, the BABEL system is tailored to Müller's German linearization grammar, which makes it difficult to compare its linearization grammar with a more traditional phrase structure based grammar. Finally, most of the work on BABEL has focussed on the linguistic side of the implemented theory, so that apart from the points discussed in Müller (1999b) the question which computational optimizations of parsing linearization grammars could be applied is essentially left open.

In this paper we want to report on our ongoing work to provide a computational platform for HPSG linearization grammars. We are extending a version of the phrase structure based ALE system (Carpenter and Penn, 1994) so that certain kinds of HPSG linearization grammars can be specified and processed as such.[2] More concretely, the ALE system is extended with a parser for discontinuous

*We would like to thank Stefan Müller, Gerald Penn and the anonymous reviewers for their valuable feedback on an earlier version of this paper. For comments, please contact us at {fouvry, dm}@sfs.nphil.uni-tuebingen.de.

[1]In this paper, we focus on HPSG related approaches to discontinuous constituency. An overview of various other approaches to discontinuous constituency can be found in Bunt and van Horck (1996).

[2]As discussed in Meurers (1994), there are a number of differences between the HPSG architecture of Pollard and Sag (1994) and ALE grammars, two of which should be mentioned here: First, the signature of an HPSG grammar in contrast to that of ALE grammars is interpreted based on the assumption that the most specific subtypes cover the domain. This is sometimes called the closed-world assumption (Gerdemann and King, 1994; Gerdemann, 1995) and leads to more inferences based on the signature. And second, HPSG theories generally consist of implicational principles, which cannot be directly

constituents which takes into account linearization statements such as precedence or isolation as explained below. Once such a special syntax for linearization statements has been introduced, one can exploit the information expressed in these statements to improve processing. Currently, we only use this to fold checking of LP statements into the parsing algorithm.

Apart from providing an implementation platform for the linearization theories as basis for dedicated computational treatments of such grammars, the platform also provides the opportunity to compare linearization and non-linearization grammars within the same system to evaluate their coverage and efficiency. While most authors have argued for linearization theories on linguistic grounds, we believe that such theories might also be advantageous computationally. Using a linearization approach, it is, for example, possible to eliminate the computationally costly empty categories for linguistically well-motivated cases of head-movement, such as verb-second in Germanic (cf., e.g., Müller, 1999a, sec. 11.5.2). This nicely complements the linguistic proposal of Bouma et al. (1998), which eliminates the need for argument or adjunct traces. It thus seems motivated to explore processing of linearization grammars both from a linguistic as well as from a computational perspective.

2 Background

In the GPSG paradigm (Gazdar et al., 1985), the notion of phrase structure (PS) was generalized to immediate dominance/linear precedence (ID/LP) grammars in order to make it possible to express generalizations over the linearization of constituents across rules. The domain in which the linear precedence rules apply are the local trees licensed by the immediate dominance rules. In a PS grammar, the string covered by a mother in a local tree is the concatenation of the strings covered by the daughters in that tree. For an ID/LP grammar, the only thing that changes is that the string covered by the mother is a permutation of the strings covered by the daughters in that local tree, and more specifically, a permutation which is consistent with all LP statements.

To use these grammars for parsing there are two options. Under the so-called *indirect* parsing approach, the ID/LP rules are expanded out into ordinary phrase structure rules and the resulting (strongly equivalent) phrase structure grammar is used for parsing.[3] Alternatively, Shieber (1984), Seiffert (1987), Blache (1992), and others proposed *direct* parsing algorithms for ID/LP grammars.

The data structure representing the linguistic categories at the nodes of the trees licensed by the ID rules has developed into a central aspect of the architecture of HPSG (Pollard and Sag, 1987, 1994). With a typed feature logic (King, 1989; Carpenter, 1992; and others) as underlying formalism, an HPSG theory is based on an explicitly declared ontology of linguistic objects, and a formal description language is used to express universal principles as constraints on these objects. The constituent structure is integrated into this linguistic ontology via the attributes DTRS and PHON. The priority of ID and LP constraints over other linguistic constraints is thus replaced by a uniform set of constraints. As a result, the string covered by the mother of a local tree is no longer per definition of the formalism the concatenation or permutation of the coverage of the daughters, which was the case for PS or ID/LP grammars. Instead, the relation between the phonology of the mother and the phonology of the daugh-

encoded in ALE, in particular when recursive data structures or principles with complex antecedents are involved.

The linearization parser described in this paper is integrated into an ALE version which was extended by Gerald Penn to include closed world inferencing and implicational constraints with complex antecedents. The resulting TRALE system thus is much closer to the HPSG architecture, but some differences remain, e.g., in the interpretation of the complex antecedents. The TRALE system can therefore be localized between a phrase structure based system like ALE and a direct HPSG implementation system like ConTroll (Götz and Meurers, 1995, 1997a,b, 1999).

[3] With unification-based ID/LP grammars, restrictions on the complexity of the categories are needed to ensure that the resulting PS grammar is finite.

ters has to be explicitly specified. This can either be done by specifying a constraint on phrasal objects with recursive relations checking the linear order of elements (Reape, 1990; Richter and Sailer, 1995; Kathol, 1995; Müller, 1995, and others), or by extending the feature logic so as to permit stating linear precedence constraints directly as proposed by Manandhar (1995).

To process HPSG grammars, a wide variety of computational systems has been developed. As with GPSG, there are *direct* approaches such as TFS (Emele and Zajac, 1990) or ConTroll (Götz and Meurers, 1997a,b) attempting to process the formalism as specified in linguistics. In these systems, an HPSG implementation simply consist of a signature declaring the domain of objects and a theory of usually implicational statements constraining this domain. While the advantage of such a direct approach is that the implemented grammar is essentially identical to the linguistic theory, direct processing with such a uniform constraint pool has the disadvantage of ignoring the domain-specific knowledge that some of these constraints, in particular the constraints on constituency as most important recursive data structure, are more important to guide processing than others. The majority of computational approaches to HPSG processing therefore require a manual recoding of an HPSG theory in order to introduce a relational backbone to guide processing. While systems such as CUF (Dörre and Dorna, 1993) rely on a general definite clause encoding for this purpose, the idea behind most systems is to require a grammar to be based on a special relational backbone, namely phrase structure. Equipped with such a phrase structure backbone, systems such as ALE (Carpenter and Penn, 1994), TDL (Krieger and Schäfer, 1994), ALEP (Alshawi et al., 1991), LKB (Copestake et al., 1993), and others can make use of well-known parsing and generation algorithms.[4] Generally speaking, the underlying idea of such systems is to separate the phrase structure relation from the general pool of linguistic constraints in order to provide a domain specific algorithmic solution which improves on the processing of the uniform constraint pool in terms of efficiency and termination.

As mentioned above, an HPSG theory does not require the phonology of a phrase to be the concatenation of that of the daughters, even if many HPSG theories define a "Constituent Ordering Principle" (Pollard and Sag, 1994) in a way which corresponds to the traditional concatenation or permutation operations which are built into the PS or ID/LP grammar formalisms. Over the past ten years, a strand of research in the HPSG paradigm has investigated the use of linearization domains which are larger than a local tree (Reape, 1990, 1994, 1996; Richter and Sailer, 1995; Kathol, 1995; Müller, 1995, 1999a; Penn, 1999, and others). The authors argue on linguistic grounds that phenomena such as clause union or verb-second in Germanic languages should be treated in terms of separating the linearization from the constituent structure to allow for discontinuous constituents. Since the linearization grammar approach severs the immediate correspondence between phonology and constituent structure, this link has to be replaced by other constraints relating the constituent structure to the string covered by that structure. As data structure for expressing this relation, most HPSG linearization approaches extend the representation of a sign with a so-called *word order domain* in addition to the constituent structure. The theory then includes constraints specifying how the word order domains are formed. Of particular interest is when the word order domains of the sub-constituents of a tree are *domain-unioned* (using Reape's terminology) to form a single domain. This permits the elements of the sub-constituents to linearize in the larger domain of the mother instead of within the local domain of each daughter. The LP statements apply to these word order domains which can, but do not have to, be larger than local trees.

When it comes to processing HPSG *linearization* proposals, one can pursue a *direct* approach and implement the proposal in a system such as TFS or ConTroll. On the other hand, Johnson (1985), Reape (1991), van Noord (1991) and Müller (1996) have proposed parsing algorithms for discontin-

[4] Of course, the downside of this is that instead of general constraint satisfaction working on any kind of query, processing is restricted to parsing and generation.

uous constituents. One can therefore follow an *indirect* approach and recode the grammar so that the immediate dominance relations form a relational backbone. The linearization parser then takes care of licensing every permutation of dominated elements.

An important difference to the direct ID/LP approaches is, however, that the construction of the word order domains and the LP constraints on the possible word orders remains on the level of the linguistic data structure, not the extracted relational backbone. Erbach et al. (1995), for example, base their constraint resolver for the linearization constraints on the feature logic extension described by Manandhar (1995) to incorporate linear precedence constraints in a formally precise way. Being more concerned with the processing aspects, Kasper et al. (1998) go as far as integrating bit vector encodings into the linguistic formalism to obtain a more efficient variant of the shuffle operation licensing the permutations of strings. In our eyes, their idea to shift such processing aspects into the linguistic formalism shows that a dedicated algorithmic treatment should be used for linearization constraints. We therefore propose to deal with linearization constraints on the level of the immediate dominance relations serving as backbone for the parser. The linguistic data structure is freed from the construction of the word order domains as well as the expression of linearization constraints. In essence, one could take our approach as following the remark of Reape (1991, pp. 67f) that "the question of specializing the permutation-complete algorithms still looms large. [...] Clearly, more work needs to be done on 'folding in' some of the low-level string combination operations into the parsers."

3 Introducing linearization statements as first class citizens in ALE

3.1 Specification of linearization constraints

3.1.1 Local statements

As a first step towards extending the ALE system with a linearization parser, one needs to fix a syntax for specifying the linearization constraints and define what the statements are intended to mean. We base our work on the "Linear Specification Language (LSL)" of Götz and Penn (in preparation), and more concretely on the subset of that language dealt with by Suhre (1999), who defines LSL grammars and their interpretation as an extension of context free grammars to discuss various formal and computational properties.

The rules of our ALE extension, which we will refer to as ID/LSL rules, are essentially built up like ordinary ALE phrase structure rules, except for the fact that the ordinary mother and daughter specification in the rule only encodes immediate dominance instead of phrase structure. When no linearization statements are specified, all orders of the terminals are permitted. Different from the ID/LP formalism this includes the possibility of interleaving subconstituents, which in a tree representation corresponds to crossing branches.

The new operator `lp>` is used to attach linearization constraints to a rule. The BNF syntax of ID/LSL rules is given in figure 1 on the following page. There are three types of linearization statements: isolation (`[`A`]`), weak linear precedence ($A < B$), and immediate linear precedence ($A << B$). The three can be combined to form complex statements using conjunction (`,`) or disjunction (`;`).

The descriptions A and B in these linearization statements are used to refer to nodes in the local tree licensed by a rule, i.e., they are Prolog variables (or contain one in a conjunction) which co-occur in the description of either the mother or a daughter of the rule. Isolation and the two linearization statements thus are expressed with respect to particular tree nodes.

The *weak linear precedence* requirement $A < B$ is satisfied if the rightmost terminal dominated by A

```
<rule> ::=  <rule_name> rule <desc> ===> <rule_body> .

<rule_body> ::=  <rule_clause>
              |  ( <rule_clause>, <rule_body> )

<rule_clause> ::=  cat> <desc>
                |  goal> <goal>
                |  lp> <lincons>

<lincons> ::=  [ <desc> ]
            |  <desc> < <desc>
            |  <desc> << <desc>
            |  ( <lincons> , <lincons> )
            |  ( <lincons> ; <lincons> )

<desc> ::= ordinary ALE description
<goal> ::= ordinary ALE goal
<rule_name> ::= ordinary ALE rule name
```

Figure 1: Syntax of the ID rules with LSL statements

is to the left of the leftmost terminal dominated by B. The *immediate linear precedence* statement A << B is interpreted alike, but enforces the additional requirement that the rightmost terminal under A is immediately to the left of the leftmost terminal under B. The *isolation* operator [A] is used to require a daughter or a mother node of an ID/LSL rule to dominate a continuous sequence of terminals, which corresponds to the notion of *bounding* in the terminology of Dowty (1989). As we will discuss in the following section, these isolated nodes also play a second role, namely as the domain in which global linearization statements are required to hold. Before we turn to this issue, let us illustrate the local linearization statements introduced so far with a simple example.

Figure 2 shows a rule intended to license a flat VP with a ditransitive verbal head and two NP complements which can occur in any order before the verb.[5]

```
ditrans_vp rule
   (VP, synsem:loc:cat: (head:verb,
                         subcat:[NN]))
   ===>
   cat> (D, synsem: (DD, loc:cat:head: (noun, case:dat)),
   cat> (A, synsem: (AA, loc:cat:head: (noun, case:acc)),
   cat> (V, word, synsem:loc:cat:(head:verb,
                                  subcat: [NN,AA,DD])),
   lp>  ([D],[A],[VP], D < V, A < V).
```

Figure 2: An example ID/LSL rule

Just like in ordinary ALE rules, the mother of the rule precedes the ===> operator. Here, the mother is described as a verbal projection subcategorizing for a single argument. The expressions starting with capital letters are variables, so that `VP` is a variable referring to the mother node of the local tree. Each daughter is described following the `cat>` operator, and again variables are introduced to be able to refer to the respective daughter nodes. The linearization constraints introduced by the `lp>` operator make reference to the variables introduced for the mother and the daughter nodes. The

[5]The examples provided in this paper are obviously intended for illustration purposes only.

three isolation statements express that the two complements and the resulting VP are required to be continuous constituents. And the two weak precedence statements ensure that both the dative and the accusative object have to precede the verbal head. Since all nodes in this example are required to be continuous (or are lexical), this ID/LSL rule corresponds to two ordinary phrase structure rules, one for each order of complements.

The three types of local linearization statements introduced above correspond to the LSL constraints discussed by Suhre (1999). While the ID/LSL format makes it possible to collapse several phrase structure rules into one, the HPSG linearization approaches in the literature clearly require additional machinery in order to be able to control word order and discontinuities across the local trees licensed by the rules. In the next section we therefore introduce global linearization statements for this purpose.

3.1.2 Global linearization statements

In the previous section, we introduced local linearization statements in order to express constraints on the order of the nodes in the local tree licensed by the ID/LSL rule which the local linearization statement is attached to. The linearization statements are thus local in two ways: They only apply to the single ID rule they are attached to, and they can only constrain the linearization of those tree nodes which are licensed by the particular ID rule.

With global linearization statements there are two corresponding facets as well: The traditional LP rules of ID/LP grammars are global in the sense that such LP rules express the same order constraints across all ID rules. Since this makes it possible to express generalizations across ID rules, such global linearization statements need to be included in our setup.[6] The second aspect in which linearization constraints can be global is related to the domain in which linearization is possible in the ID/LSL setup. The linearization domains in our ID/LSL setup can be larger than local trees (unless explicitly limited by an isolation statement). Therefore linearization statements which are independent of an ID rule licensing a local tree are needed in order to constrain the linearization of nodes which are not part of the same local tree.

Once one abandons local trees as linearization domain, the first question one needs to address is what kind of domains replace the local trees as word order domains. Reape (1994), Kathol (1995) and others proposed to explicitly introduce word order domains into the linguistic data structure, i.e., under a special attribute. In our implementation of linearization grammars, the word order domains are not represented in the linguistic data structure as such, but are part of the backbone that is singled out to be processed by dedicated algorithms. The word order domains thus need to be specified by means outside of the ordinary description language. To start with, the basic word order domain in which the global linearization constraints apply is taken to be the complete string that is parsed.[7] Every parsing input thus consists of at least one complete, continuous linearization domain. Additional word order domains can be introduced by the isolation statements specified as part of an ID/LSL rule. These isolation statements thus serve two purposes: Firstly, the isolation statements provide the domains in which the global linearization statements express their ordering constraints, i.e., a global linear precedence statement constrains the order of two nodes within the same isolation domain.[8]

[6] In GPSG this is the only way to express linearization constraints, which means that only grammars satisfying the Exhaustive Constant Partial Ordering (ECPO) property can be expressed (Gazdar et al., 1985, pp. 49).

[7] In this paper and the current implementation, we only deal with the question of parsing with linearization grammars. As far as we see, our setup could in principle be extended to include a generator.

[8] Currently, there only is one type of isolation operator, so all global linearization statements apply to the same domains. This setup could be extended by naming isolation statements and specifying in each global linearization statements which of the named domains they apply to.

And secondly, as discussed in the previous section, the isolation statements enforce that the terminals dominated by the isolated node are a continuous sequence. The phonology of the isolated constituent is included as one block in the word order domain of its mother, i.e., the internal composition of the phonology of an isolated constituent is not accessible for global linearization constraints operating in the higher word order domain.

Turning to the way in which global linearization statements are integrated into our setup, they are are introduced by the predicate `lp/2`, for which the BNF syntax is shown in figure 3.

```
<global_lp> ::= <rule_name> lp <lincons> .
```

Figure 3: Syntax of the global linearization constraints.

Just like with local linearization statements, there are two kinds of global precedence constraints, weak linear precedence (A < B) and immediate linear precedence (A << B). For the local linear precedence statements, A and B included variables to identify those tree nodes licensed by the ID schema which are ordered by the statement. This option is not available for global linear precedence statements, which are not restricted to apply to the nodes licensed by a particular schema. Instead, full descriptions are used to single out the nodes to be ordered. To illustrate this, take the global linearization statement shown in figure 4, which is intended to order inverted verbs before nominal elements.

```
inverted_verb_position lp
  synsem:loc:cat:head: (verb,inv:+) < synsem:loc:cat:head:noun.
```

Figure 4: A global linearization statement

To the left and right of the precedence operator <, one finds descriptions of signs possibly occurring in a domain (in contrast to the variables as names of nodes known to exist for the local statements). One thus needs to specify when such a global linearization statement applies. We assume that a statement of the form A < B or A << B applies in every domain containing a sign a described by A and a sign b that is described by B.[9]

While the syntax in figure 3 allows isolation as well as linear precedence statements to be included in such global constraints, currently we only treat global precedence constraints. Isolation thus can only be expressed as part of a local statement attached to a specific ID/LSL rule. Ultimately, we intend to make use of the full syntax, which will allow for global isolation statements such as shown in figure 5. The intended interpretation of this particular statement is that every node in the tree which is saturated

```
finite_s_isolation lp [ (synsem:loc:cat:subcat:[]) ] .
```

Figure 5: A global isolation statement.

is an isolated domain.

With respect to processing, a problem that can arise with respect to this application criterion is that at a particular point in processing, not enough information might be available to determine whether we have such signs a and b in the domain or not. For example, with respect to the rule in figure 4 it is clear what happens when there are no verbs or nouns in the domain (the constraint does not

[9]This includes the possibility that a and b are the same sign. As discussed below, we assume a universal interpretation for the descriptions in linear precedence constraints. Therefore only one element which is described by both descriptions A and B of a linearization statement can occur per a word order domain. Interestingly, such an interpretation of linearization constraints is actually used by Gazdar et al. (1985) and Kathol (1995).

apply), or when there is an inverted verb and a noun in the domain (it applies). But when the domain contains a noun and a verb for which processing has not yet determined the INV value, one either has to make a non-deterministic choice with respect to the INV value the verb might have and return to this point later to also try out the alternatives, or one can postpone the decision whether the linearization statement applies until the INV value is known so that no non-deterministic choice is required. To limit the search space, we decided to implement the second option and ball back on the first option only when processing failed to provide the information needed. The information needed to decide whether the linear precedence constraint applies can arise non-locally at any point during processing, be it from a ID/LSL rule, a lexical entry, or a universal principle. The issue is parallel to the problem with non-local instantiations of information relevant to an LP statement in an ID/LP grammar as discussed by Morawietz (1995). If at the end of processing, the relevant information has not been determined, we currently report the pending constraints to the user together with the potential solution. Ultimately, the fall-back strategy mentioned above should step in at this point to explore the search space by further instantiating the result non-deterministically according to the possibilities determined by the signature.

Once a global linearization statement is known to apply to a domain, the question becomes how the descriptions in the linear precedence requirement are quantified. It would be most general to introduce existential and universal quantifiers into the linearization constraint syntax, as proposed by Götz and Penn (in preparation). This way, the grammar writer could specify linearization constraints such as "for all domains including an inverted verb and a noun, every inverted verb must precede some noun." We currently settle for a less complex approach without explicit quantification and interpret all descriptions in linear precedence statements universally, i.e., for all signs in the domain which satisfy this description, the constraint must hold.

3.1.3 Limitations

It is important to mention at this point that the syntax for local and global linearization statements we provide in the extended ALE system and its interpretation sketched above is considerably less general than the general mechanism available in HPSG for obtaining the PHON value of a phrase. In fact, at least some of the HPSG linearization theories proposed in the literature clearly fall outside of the class of grammars for which we propose to deal with the linearization constraints as part of the backbone used by the linearization parser. Richter (1997), for example, proposes that the phonology of a verb in verb-second position is that of the phonology of the verb proper plus the phonological contribution of the topicalized constituent. And Kathol (1995) proposes that particle verbs lexically introduce two independent units for linearization, the verb itself and the particle.

However, the much larger part of linearization theories does not make such strong use of linearization. For these theories, our platform makes it possible to test the trade-off involved in different implementations of such grammars: from converting them to phrase structure grammars to encoding them in the version of linearization grammars which we intend to treat. Of course, certain techniques for efficient processing, such as indexing on the string position, are not applicable for parsing with linearization grammars. Still, we believe that creating a single platform where both approaches are possible at least provides a starting point on which linearization grammars can be directly compared to ordinary phrase structure grammars expressing the linearization with other methods such as head-movement.

After this sketch of the functionality of the linearization platform we develop, the second part of this paper discusses how this functionality is rendered computationally.

3.2 Compilation and indexation

3.2.1 Compilation of the linearization constraints

In the version of ALE that we build on, the phrase structure rules are compiled into an efficient clause representation. In our extension to this system, the linearization statements described in the first part of the paper are folded into the immediate dominance rules at compile time. This has the advantage that we can use the information provided about dominance and linearization at compile time to make processing more efficient. First we shall discuss how relevant information is encoded for the parser. That is followed by a description of the compilation of the local statements, and finally of the global statements. We end with a discussion of the indices generated by the compiler.

In the standard ALE parser, the edges are indexed on their left-most vertex. Given that the parser processes the daughters bottom-up from left to right, this allows for an efficient retrieval of edges from the chart. Once an edge has been found, the parser knows where it ends, and that vertex is also the leftmost (or first) vertex of the next edge to the right. In a linearization setup, the begin and end vertices alone cannot be used any longer to select edges: the constituents are not necessarily continuous. Following Johnson (1985) and Suhre (1999), we use bit-vectors to encode what part of the input an edge describes. Every bit stands for a word in the input. An empty category, for instance, is represented by the vector 0. The vectors are realized as integers, and we call that value the *coverage* of an edge.

Bit operations on the coverage are used to detect if the edges obey the ordering constraints. In the following, the functions rightmost and leftmost stand for respectively rightmost and leftmost bit of the coverage, which is the same as the respective word of the category:

immediate precedence A << B is satisfied if $\mathrm{rightmost}(A) = \mathrm{leftmost}(B) - 1$;

precedence A < B is satisfied if $\mathrm{rightmost}(A) < \mathrm{leftmost}(B)$;

continuity corresponds to a single sequence of 1-bits, i.e., if the bit-vector of an edge is 0 or if it starts with a block of only 1-bits, followed by a block of only 0-bits, the vector is continuous;

compatibility $A \wedge B$: the bitwise 'and' of the coverages must be zero, i.e., they should not overlap. For instance, $010 \wedge 101 = 000$, so they are compatible, but $010 \wedge 110 = 010$. It is obvious that this will be one of the first selection criteria;

new coverage $A \vee B$: the bitwise 'or' of the coverages, i.e., the new coverage consists of all the words covered by the daughter edges. $010 \vee 101 = 111$;

containment A vector B is contained in A if A xor $B = A - B$: all 1-bits in B are also set in A.

A central design decision was to keep the word order domain information out of the linguistic data structure, and compile it separately. This keeps the edges smaller since no (recursive) domain structure needs to be represented in the edges. Smaller edges require less copying and this results in faster unification.[10] We shall return to this in section 3.2.2.

As we have discussed in section 3.1.1, the local constraints must be structure-shared with a daughter category described by `cat>`. The sharing makes all information available which is needed to manipulate edge information at compile time such that costly searches for the edges being ordered can be

[10]To obtain a quantitative measure of the improvement involved, we intend to test a grammar with explicit domain representation against a grammar not including such a representation.

avoided during the analysis. More specifically, the parser does not need to look in the chart to find all edges with a compatible category. Instead, the compiler knows in which variable the coverage of the edges involved is stored, and these can be passed on directly to the predicate `linearize/2` that checks the compatibility with the local linearization statements. Consider the simple rules in figure 6.

```
one rule
s ===>
        cat> (A, n),
        cat> (B, vp),
        lp> (A<B).

first lp (vp < n).
```

Figure 6: A simple grammar

The compiled form of the rule `one` is given in figure 7. In that figure, `get_edge/8` retrieves edges from the chart, and `add_edge/9` adds a new result to the chart. We shall return to `get_edge/8` in the context of indexation. The integration of the local linearization statement is shown in the call to `linearize/2` in line 19 of figure 7. `Cov1` is the coverage of the first daughter (`n`), `Cov2` is the

```
    rule(FS1, pos(B,C,Cov1), E, F-(G-H), Chart) :-
       FS1='$r'(0,1,1,1,1,_),
3:     num(J),
       K is J-1,
       lpcons(K, Cov1, FS1, E, L, Chart, G-H, _),
6:     (   rule_body(one, sel(B,Cov1,_,C,C), B, L, Chart, H, G, Cov1,
                     C, F, Cov1)
       ;   leftwards(B, B1),
           nonvar(D1)when user:(Chart=D1,
9:                              arg(B1,D1,E1),
                                rule_body(one,sel(B1,Cov1,_,B-1,C),B1,
                                          C1,Chart,H,G,Cov1,C,F,Cov1)),
           assertz(user:linealppc(B1,D1,E1,C1)),
12:        fail
       ).

    rule_body(one, M, N, O, Chart, H, C, F, Cov1):-
15:    FS2='$r'(0,0,0,0,1,_),
       get_edge(M, Chart, R, Cov2, FS2, H-T, _, _),
       lpcons(R, Cov2, FS2, O, W, Chart, G-T, _),
18:    CovM is Cov1\/Cov2,
       linearize(Cov1<Cov2),
       leftmost_position(CovM, C, Z),
21:    add_edge(Z, C, CovM, W, '$r'(0,0,1,1,1,_), [F,R], G-T, one, Chart).
```

Figure 7: Compiled form of rule `one`[11]

coverage of the other daughter. `linearize/2` checks if neither is an empty category, if the coverage of one edge is not contained in the coverage of the other, and finally if the `n` edge is preceding the `vp` edge.

Global constraints cannot be integrated to the same extent, because the nodes to which they apply are not licensed by a single rule that is compiled and, more generally, it is not known at compile time to

[11]The compiled code shown here was modified slightly for presentation reasons.

which nodes and how many times the global constraint will apply. Therefore, they need to be checked at run time for every edge E that is added in which a node occurs that might be ordered by a global constraint. Before the edge for the next daughter is fetched from the chart, it is tested whether E is allowed by the global linearization constraints. This is shown in figure 7 on the preceding page in the lines 5 and 17 (`lpcons/8`), for the first and second daughters respectively. Other tests like coverage and feature structure compatibility are performed before this happens.

```
    lpcons(Id, Cov, FS, DS0, DS, Chart, L, first) :-
        ( arrays:aref(1, DS0, M) ->
3:          DS0=N
        ; arrays:aset(1, DS0, []<[], N),
          ([]<[])=M
6:      ),
        M=..[<,O,P],
        ( terms:subsumes_chk('$r'(0,0,0,0,1,_), FS) ->
9:          ordsets:ord_intersection(P, L, R),
            check_lp_constraint(R, _>Cov, Chart),
            ordsets:ord_add_element(O, Id, S)
12:     ; S=O % Constraint not applied
        ),
        ( terms:subsumes_chk('$r'(0,1,1,1,1,_), FS) ->
15:         ordsets:ord_intersection(O, L, U),
            check_lp_constraint(U, _<Cov, Chart),
            ordsets:ord_add_element(P, Id, V)
18:     ; V=P % Constraint not applied
        ),
        W=..[<,S,V],
21:     arrays:aset(1, N, W, DS).
```

Figure 8: Compiled form of global linearization statement `first`

The information about global constraints is stored in a separate data structure. This structure is manipulated by `lpcons/8`, see figure 8. When the global constraints are applied, it is checked, for every linearization statement in the constraint,[12] one needs to check if the feature structure of E is subsumed by the description in the constraint (line 8). If that is the case, the constraint will apply. To test the binary relations `<</2` and `</2` the second argument needs to be found. The data structure contains a set of pointers to all edges in the chart that satisfied the description for the second argument in the statement. This set is intersected with the set containing the pointers to the edges in the current domain, the scope of the global constraints (line 9). This yields the edges in the current domain to which a linearization constraint is attached. For each one of the edges in the intersection,[13] the linearization conditions are checked (`check_lp_constraint/3`; line 10). Finally, if this was successful, E is added to the elements that satisfy the constraint, which makes it available for the call to `check_lp_constraint/3` for the next edge (line 11). Then an analogous computation is done for the other argument of the linearization statement (lines 14–19).

Finally, there is the problem related to the order in which the edges are found. The standard ALE parser works, as already mentioned, from left to right in the rules, reading the input from right to left. This means that all edges that can be found to the right of an edge will have been found when they are needed. This makes active edges unnecessary. Unfortunately, this processing option is unavailable for a linearization parser: The sister edge can come be found in either direction. We have chosen

[12]In the example, there is only one.

[13]Here is where the global constraints are quantified universally over the word order domain.

to keep the current implementation strategy in ALE to avoid edge copying as much as possible and make use of co-routining for the cases in which an edge is not yet available. SICStus Prolog has some built-in predicates for co-routining, like `when/2`. This particular predicate suspends processing when certain conditions are not fulfilled, and proceeds as soon as they are. For instance, the goal `when(nonvar(X), write(X))` will write `X` only when it is not a variable. When it is a variable, it will wait until it becomes instantiated.

The use of co-routining is apparent in the figure 7 on page 162. First the edges to the right are exhausted (lines 6–7), and then the daughters to the left, which are still to be found, are used to apply a rule (lines 8–12). For the edges to the right, it is known when they were found (i.e., how far the parser had processed the input). The edges to the left are entirely unknown. The parser needs to prepare rules which are to be fired when edges are found, and this for for every of the remaining positions. These are calculated by `leftwards/2`. The rules are suspended until the missing edges are filled in. Because such conditions on variables are deleted on backtracking together with the variables concerned, the parser needs to assert a fact (`linealppc/4`), which maintains the link between the current code and the chart when the code is re-activated. Neither the chart itself, nor any edges need to be asserted: four variables suffice. One of the variables is associated with a condition that will trigger the call of the suspended goals, once it is fulfilled.

3.2.2 Internal representations: domains and indices

Two issues glossed over in the previous section deserve a more detailed discussion: domains and indexation. We have mentioned that the domain information, i.e., the set of edges in the domain, has to be available. When an edge is added to the chart, a pointer to it is also added to the domain set of that edge, and the domains of the daughter edges which are not isolated are unioned into this set. Updating the domain information is thus fully integrated in the parsing procedure.

More important is how the isolation works. Isolation is the constraint that hides the edges in the isolated domain from outside access, i.e., from this point upwards, the domain is only available as one block. It is a requirement that isolated domains are continuous. The continuity is checked (see p. 161) and the domain that is passed on is limited to the pointer to the isolating edge, instead of the entire set.

To allow edges that satisfy a series of criteria to be fetched efficiently, there are currently three indices in the chart, for each of the values that are often used as selection criteria. The indices group pointers to edges which are the same given a certain criterion. A good example is the coverage index: for every coverage that occurs during the analysis, all edges with that coverage are collected in one set. The other indices are for the edge IDs and the chart state. The selection criteria are passed on to `get_edge/8` in the compiled form of the rule as shown in figure 7 on page 162, which only fetches the edges from the chart that meet the requirements.

The global constraints and the domains also use indices, although those are stored separately from the chart indices: the domain index changes when a domain is isolated, and the global constraints are first looked up by constraint, not by edge. Indexation can be used here because the word order domain information is not stored in the linguistic structure. All the indices are updated when the edge is added to the chart.

4 Summary

In this paper, we have presented our ongoing project to extend the ALE system so that linearization grammars can be directly specified. The system has been extended with a parser for discontinuous constituents which takes into account linearization statements (precedence, isolation) which are specified separate from the HPSG theory as such. We have sketched the local and global linearization statements we introduced and how they are currently interpreted.

As motivation for this enterprise, we argued that from a processing perspective the linearization constraints assumed by HPSG-linearization approaches should be separately specified instead of including them on the level of the linguistic data structure. This makes it possible to keep the linguistic data structure small in order to speed up unification and to explore dedicated algorithms treating the linearization constraints as part of a parsing process. Finally, the system will make it possible to compare linearization and non-linearization grammars within the same setup to evaluate their coverage and efficiency.

References

Alshawi, Hiyan, Arnold, D. J., Backofen, R., Carter, D. M., Lindop, J., Netter, K., Pulman, S. G., Tsujii, J., and Uszkoreit, H. (1991). Eurotra ET6/1: Rule Formalism and Virtual Machine Design Study (Final Report). Final report, CEC.

Blache, Philippe (1992). Using Active Constraints to Parse GPSGs. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, pp. 81–86.

Bouma, Gosse, Malouf, Rob, and Sag, Ivan A. (1998). A Unified Theory of Complement, Adjunct, and Subject Extraction. In G. Bouma, G.-J. Kruijff, and R. Oehrle (Eds.), *Proceedings of the Joint Conference on Formal Grammar, Head-driven Phrase Structure Grammar, and Categorial Grammar (FHCG-98)*, pp. 83–97. Universität des Saarlandes, Saarbrücken.

Bunt, Harry and Horck, Arthur van (Eds.) (1996). *Discontinuous Constituency*, Volume 6 of *Natural Language Processing*. Berlin and New York, NY: Mouton de Gruyter.

Carpenter, Bob (1992). *The Logic of Typed Feature Structures – With Applications to Unification Grammars, Logic Programs and Constraint Resolution*, Volume 32 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge, UK: Cambridge University Press.

Carpenter, Bob and Penn, Gerald (1994). ALE – The Attribute Logic Engine, User's Guide, Version 2.0.1, December 1994. Technical report, Computational Linguistics Program, Philosopy Department, Carnegie Mellon University.

Copestake, Ann, Sanfilippo, Antonio, Briscoe, Ted, and Paiva, Valeria de (1993). The ACQUILEX LKB: An introduction. In T. Briscoe, A. Copestake, and V. De Paiva (Eds.), *Inheritance, Defaults and the Lexicon*, pp. 148–163. Cambridge, UK: Cambridge University Press.

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

Dowty, David (1989). On the Semantic Content of the Notion of "Thematic Role". In G. Chierchia, B. Partee, and R. Turner (Eds.), *Properties, Types, and Meanings*, Volume 2, pp. 69–129. Dordrecht: Kluwer Academic Publishers.

Emele, Martin and Zajac, Rémi (1990). Typed Unification Grammars. In H. Karlgreen (Ed.), *Proceedings of the 13th Conference on Computational Linguistics (COLING-90)*. Helsinki.

Erbach, Gregor, Kraan, Mark van der, Manandhar, Suresh, Ruessink, Herbert, Skut, Wojciech, and Thiersch, Craig (1995). Extending Unification Formalisms. In *Proceedings of the 2nd Language Engineering Convention*. London and New York, NY.

Gazdar, Gerald, Klein, Ewan, Pullum, Geoffrey K., and Sag, Ivan A. (1985). *Generalized Phrase Structure Grammar*. Cambridge, MA: Harvard University Press.

Gerdemann, Dale (1995). Open and Closed World Types in NLP Systems. In *Proceedings of the DGfS Fachtagung Computerlinguistik*. Düsseldorf.

Gerdemann, Dale and King, Paul (1994). The Correct and Efficient Implementation of Appropriateness Specifications for Typed Feature Structures. In *Proceedings of the 15th Conference on Computational Linguistics (COLING-94)*, pp. 956–960. Kyoto.

Götz, Thilo and Meurers, Walt Detmar (1995). Compiling HPSG Type Constraints into Definite Clause Programs. In *Proceedings of the 33rd Annual Meeting of the ACL*, pp. 85–91. Association for Computational Linguistics: MIT, Cambridge, MA.

Götz, Thilo and Meurers, Walt Detmar (1997a). The ConTroll System as Large Grammar Development Platform. In *Proceedings of the Workshop "Computational Environments for Grammar Development and Linguistic Engineering (ENVGRAM)" held in conjunction with the 35th Annual Meeting of the ACL and 8th Conference of the EACL*, pp. 38–45. Association for Computational Linguistics: Universidad Nacional de Educación a Distancia, Madrid.

Götz, Thilo and Meurers, Walt Detmar (1997b). Interleaving Universal Principles and Relational Constraints over Typed Feature Logic. In *Proceedings of the 35th Annual Meeting of the ACL and 8th Conference of the EACL*, pp. 1–8. Association for Computational Linguistics: Universidad Nacional de Educación a Distancia, Madrid.

Götz, Thilo and Meurers, Walt Detmar (1999). The importance of being lazy – Using lazy evaluation to process queries to HPSG grammars. In G. Webelhuth, J.-P. Koenig, and A. Kathol (Eds.), *Lexical and Constructional Aspects of Linguistic Explanation*, Studies in Constraint-Based Lexicalism, pp. 249–264. Stanford, CA: CSLI Publications. Published version of a paper presented at the 3rd Int. Conference on HPSG, May 1996, Marseille.

Götz, Thilo and Penn, Gerald (in preparation). A Linear Specification Language for Parsing with Freer Word-Order Languages. In G. Penn (Ed.), *An Introduction to LSL: Motivation, Semantics and Computation*, Arbeitspapiere des SFB 340. Tübingen: Universität Tübingen.

Johnson, Mark (1985). Parsing with discontinuous constituents. In *Proccedings of the 23rd Annual Meeting of the ACL*, pp. 127–132. Chicago, IL.

Kasper, Robert T., Calcagno, Mike, and Davis, Paul C. (1998). Know When to Hold 'Em: Shuffling Deterministically in a Parser for Nonconcatenative Grammars. In *Proceedings of the Thirty-Sixth Annual Meeting of the ACL*.

Kathol, Andreas (1995). *Linearization-Based German Syntax*. Ph. D. thesis, Ohio State University, Columbus, OH.

King, Paul John (1989). *A Logical Formalism for Head-Driven Phrase Structure Grammar*. Ph. D. thesis, University of Manchester, Manchester.

Krieger, Hans-Ulrich and Schäfer, Ulrich (1994, November). TDL—A type description language for HPSG. Part 1: Overview. Research report RR-94-37, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Saarbrücken, Germany.

Manandhar, Suresh (1995). Deterministic Consistency Checking of LP Constraints. In *Proceedings of the 7th Conference of the EACL*. Dublin. arXiv:cmp-lg/9502017 v2 16 Feb 95.

Meurers, Walt Detmar (1994). On Implementing an HPSG Theory – Aspects of the Logical Architecture, the Formalization, and the Implementation of Head-Driven Phrase Structure Grammars. In E. W. Hinrichs, W. D. Meurers, and T. Nakazawa (Eds.), *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation.*, Arbeitspapiere des SFB 340 Nr. 58, pp. 47–155. Tübingen: Universität Tübingen.

Morawietz, Frank (1995). Formalization and Parsing of Unification–Based ID/LP Grammars. Arbeitspapiere des SFB 340 68, Universität Tübingen.

Müller, Stefan (1995). Scrambling in German – Extraction into the *Mittelfeld*. In B. K. T'sou and T. B. Y. Lai (Eds.), *Proceedings of the tenth Pacific Asia Conference on Language, Information and Computation*, pp. 79–83. City University of Hong Kong.

Müller, Stefan (1996). The Babel-System–An HPSG Prolog Implementation. In *Proceedings of the Fourth International Conference on the Practical Application of Prolog*, pp. 263–277. London and New York, NY. http://www.dfki.de/˜stefan/Pub/babel.html.

Müller, Stefan (1999a). *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Number 394 in Linguistische Arbeiten. Tübingen: Max Niemeyer Verlag.

Müller, Stefan (1999b). Restricting Discontinuity. Verbmobil Report 237, Deutsches Forschungszentrum für Künstliche Intelligenz, Saarbrücken. This report is also published in the Proceedings of GLDV 99 (Frankfurt/Main). The initial part is contained in the proceedings of the 5th Natural Language Processing Pacific Rim Symposium 1999 (NLPRS'99). http://www.dfki.de/˜stefan/Pub/e_restricting.html.

Penn, Gerald (1999). Linearization and WH-extraction in HPSG: Evidence from Serbo-Croatian. In R. D. Borsley and A. Przepiórkowski (Eds.), *Slavic in HPSG*, pp. 149–182. Stanford, CA: CSLI Publications.

Pollard, Carl and Sag, Ivan A. (1987). *Information-based Syntax and Semantics, Vol. 1: Fundamentals*. Number 13 in CSLI Lecture Notes. Stanford, CA: CSLI Publications.

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

Reape, Mike (1990). A Theory of Word Order and Discontinuous Constituency in West Continental Germanic. In E. Engdahl and M. Reape (Eds.), *Parametric Variation in Germanic and Romance: Preliminary Investigations*, DYANA Deliverable R1.1.A, ESPRIT Basic Research Action BR 3175, pp. 25–39. Edinburgh: Centre for Cognitive Science, University of Edinburgh.

Reape, Mike (1991). Parsing Bounded Discontinuous Constituents: Generalisations of some common algorithms. In M. Reape (Ed.), *Word Order in Germanic and Parsing*, DYANA Deliverable R1.1.C, ESPRIT Basic Research Action BR 3175, pp. 41–70. Edinburgh: Centre for Cognitive Science, University of Edinburgh.

Reape, Mike (1994). Domain Union and Word Order Variation in German. In J. Nerbonne, K. Netter, and C. Pollard (Eds.), *German in Head-Driven Phrase Structure Grammar*, Number 46 in CSLI Lecture Notes, pp. 151–197. Stanford, CA: CSLI Publications.

Reape, Mike (1996). Getting things in order. See Bunt and van Horck (1996), pp. 209–253. Published version of a Ms. from 1990.

Richter, Frank (1997). Die Satzstruktur des Deutschen und die Behandlung langer Abhängigkeiten in einer Linearisierungsgrammatik. Formale Grundlagen und Implementierung in einem HPSG-Fragment. In E. W. Hinrichs, W. D. Meurers, F. Richter, M. Sailer, and H. Winhart (Eds.), *Ein HPSG-Fragment des Deutschen. Teil 1: Theorie*, Arbeitspapiere des SFB 340 Nr. 95, pp. 13–187. Tübingen: Universität Tübingen.

Richter, Frank and Sailer, Manfred (1995). Remarks on Linearization: Reflections on the Treatment of LP-Rules in HPSG in a Typed Feature Logic. Master's thesis, University of Tübingen.

Seiffert, Roland (1987). Chart-Parsing of Unification-Based Grammars with ID/LP-Rules. In E. Klein and J. van Benthem (Eds.), *Categories, Polymorphism and Unification*, pp. 335–54. CCS/ILLI, Edinburgh and Amsterdam.

Shieber, Stuart M. (1984). Direct Parsing of ID/LP Grammars. *Linguistics and Philosophy 7*, 135–154.

Suhre, Oliver (1999). Computational Aspects of a Grammar Formalism for Languages with Freer Word Order. Diplomarbeit, Department of Computer Science, University of Tübingen.

Noord, Gertjan van (1991). Head Corner Parsing for Discontinuous Constituency. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 114–121.