A Report on the Draft EAGLES Encoding Standard for HPSG

Gregor Erbach Jochen Dörre Suresh Manandhar Wojciech Skut
Hans Uszkoreit*

Abstract

This report describes the progress made towards a declarative, language independent, high-level encoding format for HPSG grammars. The design of the encoding format ensures efficient implementability yet at the same time renders grammars to be highly compact thereby enhancing readability. Particular attention has been paid to the expressivity/efficiency trade-off. A clear upgrade path has also been provided for further extensions to the encoding format to accomodate future developments.

1 Introduction

The recent years have seen a growth in both the number of implemented HPSG grammars and the platforms that such grammars have been implemented upon. Although this is a very healthy trend from the perspective of promoting HPSG as a computationally viable grammatical theory, it also poses problems regarding the reusability of HPSG grammars. The specific problem of reusability for HPSG grammars is that grammars developed on one platform cannot be easily run on another platform. The second problem of reusability is that both the HPSG theory and implementation platforms are subject to improvements and changes. This means that grammars written sometime ago become obsolete and have to be recoded.

Apart from the reusability issue, there is also a serious need that linguists be able to test their grammars with as little programming effort as possible and that sample grammars found in texts can be tested and validated with minimum effort.

The EAGLES HPSG encoding format is an attempt to fulfill these needs by coming up with a declarative, extensible and yet efficiently implementable encoding format for HPSG grammars. The following goals were central in the design of the current draft standard:

1. it should be sufficiently expressive to state most if not all of the HPSG type hierarchy, principles, schemas and lexical rules as stated in HPSG-II [Pollard and Sag, 1994]. Hence, the formalism should permit the development of grammars of significant size and complexity.

2. it should not be tied or dependent in any way to an implementation language - Prolog, LISP or any (typed) feature formalism.

3. it should be restrictive enough to allow fully automatic compilation into several target typed feature formalisms which differ significantly from each other.

Goal (1) is the primary requirement of the encoding standard. Goal (2) is necessary to ensure portability across language platforms and to provide independence from specific implementation languages and can be thought as the reusability requirement. Goal (3) is a practical requirement to ensure the very survival of the standard and to demonstrate the implementability of the standard with existing technology.

The above goals conflict with each other. For instance, the requirement for high expressivity imposed by (1) conflicts with the requirement of translatability into current generation target feature formalisms as imposed by (3). HPSG for instance employs a range of descriptive machinery such as set-valued descriptions and linear precedence constraints which are unavailable in formalisms such as ALE [Carpenter and Penn, 1994], CUF [Dörre and Dorna, 1993] and ALEP [Meylemans, 1994]. For this reason, the aim of the current draft is to define a core standard that can be compiled into most of existing typed feature formalisms. The aim is then to incorporate more sophisticated aspects of HPSG in the

*Authors' affiliations: Gregor Erbach (erbach@dfki.uni-sb.de), Wojciech Skut (skut@coli.uni-sb.de) and Hans Uszkoreit (uszkoreit@dfki.uni-sb.de): Universität des Saarlandes, Saarbrücken; Jochen Dörre (jochen@ims.uni-stuttgart.de): Universität Stuttgart, Stuttgart; Suresh Manandhar (Suresh.Manandhar@ed.ac.uk): Language Technology Group, University of Edinburgh. The project was supported by the Commission of the European Communities EAGLES initiative. Thanks to Chris Brew and Jo Calder for providing useful comments.

near future. Of course, backwards compatibility will have to be maintained. The idea is to allow the standard to evolve in step with developments in HPSG and in implementation technologies - as should be the case with any practical standard.

For a grammar standard to have an impact there is need for enabling software to assist in the migration from the current development environment to the one in which the current grammar standard is employed. There are two issues that need to be addressed:

1. software needs to be developed to enable grammars written in the proposed standard to be automatically compiled into existing feature formalisms - the concrete realisation of goal (3) above.
2. porting of existing grammars into the standard format should be possible with minimal effort.

2 Overview of the draft standard

HPSG as a grammatical theory is composed of several components that are needed to specify a grammar of any significant size. These different components of HPSG are:

1. type hierarchy and appropriateness conditions
2. schemas
3. principles
4. lexicon
5. lexical rules

These distinctions are primarily motivated by the needs of the grammar writer. However, this distinction may not necessarily be preserved when a grammar is encoded within an feature formalism. Thus for instance, within a sufficiently expressive feature formalism the principles and the schemas can be stated as constraints on the types. This situation is analogous to the case when grammar specifications are encoded in a very expressive logic wherein all HPSG specifications are encoded as a set of axioms.

The approach taken within the EAGLES proposal is to keep grammar specifications as close to the linguists' notion as possible by maintaining the above distinctions in the grammar specification, while leaving open the manner in which such specifications are encoded within feature formalisms. At the same time, attention has been paid in making sure that conventional parsing techniques can still be employed (if desired) and efficiency is not lost. This is essential in order to make grammar specifications usable in a real environment.

The draft standard divides the HPSG specification language into several sub-specification languages. These consists of:

1. type specification
2. feature description
3. relational language
4. schema specification
5. principle specification
6. lexicon specification
7. lexical rule specification
8. morphophonological rule specification

At the time of writing, the lexicon, lexical rule and morphophonological rule specification languages are provisional, and not described in this document. In the following, we provide a brief overview of the rest of the specification sub-languages, providing sample encodings where appropriate.

3 Type system specification language

The primary concerns in coming up with a suitable type specification language for HPSG grammars were as follows:

1. choice of semantics for multiple inheritance
2. choice of semantics for appropriateness conditions
3. expressivity of the type system *w.r.t.*
   - defining HPSG non-lexical signs
   - defining hierarchical lexicons
4. compact minimal syntax

First of all, there are at least 3 different possibilities (and their combinations) in choosing a suitable multiple-inheritance semantics namely:

1. *open world*
   The *open world* interpretation of multiple inheritance is provided in systems such as ProFIT (mutidimensional inheritance) [Erbach, 1995], TDL (open-world mode) [Krieger and Schäfer, 1994] and CUF (subtype spec.) [Dörre and Dorna, 1993]. In essence, an open-world interpretation permits unification of any two types s and t as long as their conjunction $s\&t$ is consistent. It is the least restrictive one. One important advantage of the open-world interpretation is that it permits cross-classification of lexical types which is very important within a hierarchical lexicon.

   The big disadvantage of the purely open-world interpretation is that specifying HPSG non-lexical types becomes cumbersome since the grammar writer has then to specify, for instance, that the type *synsem* is incompatible with *loc* by employing a disjointness statement such as *disjoint(loc,synsem)*. This makes the grammar harder to read. Hence, the open-world interpretation alone is insufficient.

2. *closed world*
   The *closed world* interpretation is provided in systems such as TROLL [Gerdemann and Götz, 1994], TFS [Zajac, 1992] and CUF (disjunctive spec.). In a closed world interpretation, given the subtypes $[t_1, t_2, t_3]$ of a type s; the type s is interpreted as being equivalent to the disjunction $t_1 \vee t_2 \vee t_3$ *i.e.* $s = t_1 \vee t_2 \vee t_3$.

   One big advantage of a closed-world reasoning interpretation is *strong* type checking in that an underspecified description of type t must be consistent with at least one of its subtypes t_i.

   The disadvantage is that disjunctive expansion is computationally expensive and clever schemes need to be devised to make the process efficient.

3. *greatest lower bound*
   In systems such as LOGIN [Aït-Kaci and Nasr, 1986] and ALE [Carpenter and Penn, 1994] the inheritance hierarchy is required to be a semi-lattice. The conjunction $s\&t$ of types s and t is then interpreted as being equivalent to the *greatest lower bound* (*i.e. glb(s,t)*) of s and t. Thus in this approach the structure of the type hierarchy itself decides the mutual compatibility of types. Apart from this additional constraint, the *greatest lower bound* interpretation is similar to the *open world* interpretation.

   One advantage of the *greatest lower bound* interpretation in comparison to the *open world* interpretation is that incompatibility of types need not stated explicitly thereby preserving compactness and readability of the grammar. The disadvantage being that this interpretation becomes quite cumbersome for stating hierarchical lexicons since it requires *unique glb*s.

   The advantage of the *greatest lower bound* and the *open world* interpretation in comparison to the *closed world* interpretation is that they both permit lazy evaluation since implementations do not need to deal with disjunctive expansion. This can also be considered a disadvantage since it means "weaker" type checking.

The greatest lower bound interpretation for multiple inheritance was chosen as a sufficient and minimally expressive mechanism for stating HPSG non-lexical signs. For stating hierarchical lexicons, in addition an open-world interpretation is highly desirable. A combined system that provides both these interpretations is currently being investigated. One important point is that the *closed world* interpretation can be added on top (if needed in the future) without having to change the grammar.

Feature appropriateness constraints [Carpenter, 1992] [Manandhar, 1993] are modelled after the ALE formalism as it provides a simple but sufficient mechanism to model HPSG signs. Feature constraints can be attached to types. This is especially useful for specifying hierarchical lexicons. For space reasons, we do not discuss these here.

The parametric type $list(\alpha)$ is built-in to the standard allowing types such as $list(sign)$, $list(synsem)$, $list(quant)$ to be straightforwardly specified. Given below is a partial specification of the HPSG *sign* type.

```
  sign >   [word,non_word]
    approp [synsem:synsem,
            qstore:list(quant),
            qretr:list(quant)].

  cat approp [subcat:list(synsem),
              head:head,
              marking:marking].

   mod_synsem > [synsem,none].
    synsem approp [loc:loc,
                   non_loc:non_loc].

  loc approp [cat:cat,
              cont:sem_obj,
              conx:conx].

  conx approp [backgr:list(psoa),
               c_inds:c_inds].
```

Lexical hierarchies can be specified using feature constraints on types as illustrated in the example below:

```
%% The HPSG hierarchical lexicon
verb_sign >     [trans_verb_sign, intrans_verb_sign]
            def (synsem:loc:(cat:head:verb &
                             cont:quants:[] &
                             conx:backgr:[]) &
                 (@empty_non_loc) &
                 qstore:[]).

trans_verb_sign def
  (synsem:loc:cat:(head:(mod:none &
                         aux:minus &
                         inv:minus) &
                   subj:[synsem] &
                   comps:[synsem] &
                   spr:[] &
                   marking:unmarked)).
```

4 Schema specification language

The idea here is to state HPSG schemas close to the HPSG book notation but yet ensure that efficient compilation into PS-rules or relational dependencies is possible. Rudimentary support for free word-order is also provided.

For instance, head-complement schema (for English) can be stated as follows:

```
schema :: head_comp( syn:loc:head:subj:[_|_],
                     head_dtr:sign &
                     comp_dtrs:list(sign),
                    [head_dtr < comp_dtrs]
                    ).
```

```
%                                    HPSG Schemas
% ============================================================================
%                                     headed_struct
%                                          |
%                    ______________________|____________________________
%                   |              |                 |                  |
%             f_struct  head_mark_struct  head_adjn_struct  head_filler_struct
%                   |
%               ____|___________________________________
%              |                   |                    |
%         subj_struct         comp_struct         head_spec_struct
%              |                   |
%       _______|______________    _|__________________
%      |                      |  |                    |
% head_subj_struct    head_subj_comp_struct    head_comp_struct
% ============================================================================

headed_struct >   [f_struct,
                   head_mark_struct,head_adjn_struct,head_filler_struct]
           approp [head_dtr:sign].

   f_struct  >  [subj_struct,
                 comp_struct,
                 head_spec_struct].

       subj_struct >    [head_subj_struct,head_subj_comp_struct]
                 approp [subj_dtr:sign].

       comp_struct >    [head_subj_comp_struct,head_comp_struct].
                 approp [comp_dtrs:list(sign)].

       head_spec_struct approp [spec_dtr:sign].

   head_mark_struct approp [mark_dtr:sign].
   head_adjn_struct approp [adjn_dtr:sign].
   head_filler_struct approp [filler_dtr:sign].

schema :: head_subj(sign,
                    head_subj_struct,
                    [subj_dtr,head_dtr]
                   ).
schema :: head_comp(sign,
                    head_comp_struct,
                    [head_dtr,comp_dtrs]
                   ).
schema :: head_subj_comp(sign,
                         head_subj_comp_struct,
                         [head_dtr,subj_dtr,comp_dtrs]
                         ).
```

Figure 1: The EAGLES specification of HPSG schemas

Note that the schema definition contains 3 parts namely:

1. the specification of the `'MOTHER'` sign as the first argument of head_comp
2. the specification of the "daughters" sign as the second argument of head_comp
3. the specification of LP constraints as the third argument of head_comp

As we shall illustrate below, this seperation permits an implementation to compile out schema definitions into PS-rules enabling use of existing efficient parsing techniques.

The standard permits (and in fact requires) the "daughters" sign be a *type* defined within the type hierarchy. This means that appropriateness conditions and consistency checking apply to the "daughters" sign. This allows the "daughters" sign to be specified in a hierarchical fashion. For instance, the head_comp schema can be restated as given below:

```
headed_struct >   [head_comp_struct]
           approp [head_dtr:sign].

head_comp_struct >  []
             approp [comp_dtrs: list(sign)].

schema :: head_comp( syn:loc:head:subj:[_|_],
                      head_comp_struct,
                     [head_dtr < comp_dtrs]
                     ).
```

The notation `[head_dtr < comp_dtrs]` means that the head_dtr "precedes" the comp_dtrs list. On the other hand, the notation `[head_dtr,comp_dtrs]` means that the head_dtr "immediately precedes" the comp_dtrs. Given a local tree consisting of just the head_dtr and the comp_dtrs the two specifications are equivalent.

One important consideration during the design of the schema specification language was that compilation of the schema declarations into PS-rules was possible. Consider, for instance the head_comp schema given above. This can be compiled into the following PS-schema:

sign	$->$	*sign*,	*list*(*sign*)
'MOTHER'		head_dtr	comp_dtrs

In this particular case, a left-to-right rule instantiation would instantiate the comp_dtrs list (cf. ALE). In the general case when the position of the head is not fixed a standard head-driven approaches can be employed.

The almost complete specification of HPSG schemas including the different types of "daughters" signs is provided in fig. 1 which further refines the above specification.

The standard currently does not provide for mechanisms involving complex word order such as using the domain union mechanism [Reape, 1993] or LP constraints [Manandhar, 1995]. Future versions of the standard may add these.

5 HPSG principles encoding format

The principles encoding format allows HPSG principles to be stated independently of the schema encoding format. This is achieved by employing a principle interface specification that defines the principles that are applicable to a given schema. For instance, the head-feature principle, the marking principle and the subject condition would be stated as follows:

```
principles::
 [ head_feature_principle('MOTHER',head_dtr),

   ~head_mark_struct ->  marking_principle('MOTHER',head_dtr),
   marking_principle('MOTHER',mark_dtr)

   subject_condition(subj_dtr,comp_dtrs)
 ].
```

The specification of the head-feature principle specifies that it applies to every schema which contains a head_dtr. The `'MOTHER'` argument which by definition is present in every schema specification is passed as the first argument of head_feature_principle. Similarly, the subject_condition applies to every schema which contains a subj_dtr and a comp_dtrs. The marking_principle on the other hand, applies to every schema containing a head_dtr but excluding the head_mark_struct.

The above specifications only specify the interface between the schemas and the actual definition of the principles. The definition of the principles themselves are done using a simple relational language (in effect, restricted Prolog with an important extension). The relational language is discussed below.

6 Relational Language

The relational language is defined to be a pure Horn language (i.e. no cuts (!s) and no negation by failure) without any recursion. However, it provides a host of library routines to compensate for this simplicity. These involve guarded constraints, list manipulation, maplist predicate, list-based set operations etc. This restricted language guarantees termination. The library routines hide low-level operations and allow the grammars to be lot more abstract and portable across different programming language environments.

Guarded constraints provide the ideal mechanism to abstract over the manner in which *conditional* HPSG descriptions are encoded within current implementations. Consider the definition of the subject_condition predicate given below:

```
subject_condition(@inher_slash(SubjSlash),CompDtrs):-
    case(SubjSlash,
             [nelist => set_member(nelist, @accumulate(CompDtrs, inher_slash))]).
```

This specification highlights several important aspects of the relational language.

Firstly, the relational language permits functional annotation (alternatively a macro annotation) of a predicate call. The predicate inher_slash/2 would be defined as follows:

```
inher_slash(Slash, synsem:non_loc:slash:Slash).
```

The functional annotation permits the last argument of the a predicate call to be implicitly bound to the context. Thus a functional annotation `@p(A1, ... , An)` would require a predicate definition for `p/n+1` to exist.

The above definition of the subject_condition is equivalent to the following transformed definition where all the functional annotations have been eliminated:

```
subject_condition(Subj,CompDtrs):-
    inher_slash(SubjSlash, Subj),
    accumulate(CompDtrs, inher_slash, Slashes),
    case(SubjSlash,
             [nelist => set_member(nelist, Slashes)]).
```

Secondly, the library routines built-in to the relational language includes list iteration predicates such as accumulate/3 which in the above call extracts the inherited-slash values of the CompDtrs.

A summary of library predicates is provided below:

1. List manipulation routines - *append/3, select/3, reverse/2.*
2. Maplist routines for iterating over list structures - *maplist/3, accumulate/3.*
3. List based set operations - *set_member/2, set_difference/3, set_union/3, set_subset/2, set_disjoint/2.*

The primary aim of including the library predicates is to shield the grammar writer from having to write implementation language dependent low-level routines. At the same time, it enables the relational language itself to be kept simple and small since the more complex machinery is hidden inside the library routines.

This approach constrasts with an alternative solution which would include the full Prolog machinery so that all the library routines can be user defined. But clearly grammars written in this more expressive language would be harder to read and would require a Prolog interpreter/compiler. The library routines are expected to grow when the needs of the grammar writer are not met by the available set of routines.

Our current solution opens the possibility of efficient implementability in other languages and at the same time make its possible to write compact highly readable grammars.

Thirdly, the relational language provides guarded constraints as a declarative mechanism for specifying HPSG conditional descriptions such as the ones embodied in the subject_condition.

Guarded constraints provide support for *if-then-else* style programming in a declarative setting. Given a constraint of the form C & $if\ G\ then\ B\ else\ E$; C is known as the *context*, G is known as the *guard*; B, E are *actions*. The action B is added to the context whenever the context C *entails* the guard G. On the other hand, the action E is added to the context whenever C disentails G. Following Saraswat's work

```
principles ::
[ head_subj_struct -> valence_principle_subj('MOTHER',head_dtr,[subj_dtr]),
  head_comp_struct -> valence_principle_comps('MOTHER',head_dtr,comp_dtrs),
  valence_principle_subj_comps('MOTHER',head_dtr,[subj_dtr|comp_dtrs]),
  valence_principle_spec('MOTHER',head_dtr,[spec_dtr]),

  head_subj_struct ->  semantics_principle('MOTHER',head_dtr,[subj_dtr]),
  head_comp_struct ->  semantics_principle('MOTHER',head_dtr,comp_dtrs),
  semantics_principle('MOTHER',head_dtr,[subj_dtr|comp_dtrs]),
  semantics_principle('MOTHER',head_dtr,[mark_dtr]),

  head_subj_struct  -> nonlocal_feature_principle('MOTHER',head_dtr,[subj_dtr]),
  head_comp_struct  -> nonlocal_feature_principle('MOTHER',head_dtr,comp_dtrs),
  nonlocal_feature_principle('MOTHER',head_dtr,[subj_dtr|comp_dtrs]),
  nonlocal_feature_principle('MOTHER',head_dtr,[mark_dtr])
 ].
% valence_principle_xxx(Mother,Head-Daughter,Functional-Daughters)
%---------------------------------------------------------------
valence_principle_subj(@subj(MSub) & @comps(C) & @spr(Spr),
                       @subj(@append(MSub,FDtrSyns)) & @comps(C) & @spr(Spr),
                       @synsems(FDtrSyns)).
% semantics_principle(Mother,Semantic-Head,Other-Dtrs)
%---------------------------------------------------------
% Semantic-Head is adjunct-dtr otherwise head-dtr
semantics_principle(@qstore(MQStore) & @qretr(MRetr) & @cont(MCont),
                    SHead & @cont(SCont), ODtrs) :-
  accumulate([SHead|ODtrs], qstore, DQStore),
  case(SCont, [(psoa, ( MCont = ( @nucleus(Nucl) & @quants(MQuants) ),
                        SCont = ( @nucleus(Nucl) & @quants(SQuants) ),
                        set_difference(MQStore,DQStore,MRetr),
                        append(MRetr,SQuants,MQuants)
                       ))
               ],
                   SCont = MCont,
                   MRetr = [],
                   MQStore = DQStore
        ).
% nonlocal_feature_principle(Mother,Head-Dtr,Other-Dtrs)
%---------------------------------------------------------------
nonlocal_feature_principle(@inher_slash(MISlash) &
                           @inher_que(MIQue) & @inher_rel(MIRel),
                           HeadDtr & @to_bind_slash(HTSlash) &
                                     @to_bind_que(HTQue) &
                                     @to_bind_rel(HTRel),
                           ODtrs):-
  accumulate([HeadDtr|ODtrs], inher_slash, DISlash),
  accumulate([HeadDtr|ODtrs], inher_que, DIQue),
  accumulate([HeadDtr|ODtrs], inher_rel, DIRel),
  set_difference(HTSlash,DISlash,MISlash),
  set_difference(HTQue,DIQue,MIQue),
  set_difference(HTRel,DIRel,MIRel).
```

Figure 2: The EAGLES specification of HPSG principles

[Saraswat, 1993] guarded constraints forms the basis of concurrent constraint programming languages such as Oz [Smolka *et al.*, 1993].

Guarded constraints are specified using the case/2 and case/3 predicates which is a syntactic sugaring of the *if-then-else* style syntax. A description of the form `case(C, [ G1 => B1, G1 => B2 ], E)` is equivalent to:

$$\begin{array}{ll} C\ \& & if\ G1\ then\ B1\ else\ F1 = t\ \& \\ & if\ G2\ then\ B2\ else\ F2 = t\ \& \\ & if\ (F1 = t\ \&\ F2 = t)\ then\ E \end{array}$$

7 Discussion and Conclusions

In this short report, we have provided an overview of the efforts of the EAGLES working group on formalisms to provide a simple encoding format for HPSG grammars. This report discussed parts of the encoding format relating to HPSG schemas, type system and principles. The remaining parts of the draft standard deal with lexical entry formats, lexical rule specification and morphology. All parts of the encoding format are continually being refined and improved.

Three major concerns during the formulation of the encoding format was *simplicity*, *declarativity* and *efficient implementability*. One perhaps crude way to judge simplicity is simply by grammar size; the shorter the encoding of the same grammar the simpler it is. In this respect, the current format certainly provides a highly compact notation for HPSG grammars. Furthermore, full declarativity of grammars are retained. And finally, we are confident that the current design allows efficient implementation using existing techniques.

References

[Aït-Kaci and Nasr, 1986] Hassan Aït-Kaci and Roger Nasr. LOGIN : A logic programming language with built-in inheritance. *The Journal of Logic Programming*, 3:185–215, 1986. Also appeared as MCC report AI-068-85.

[Carpenter and Penn, 1994] Bob Carpenter and Gerald Penn. *The Attribute Logic Engine: User's Guide (Version 2.0.1)* . Carnegie-Mellon University, Pittsburgh, December 1994.

[Carpenter, 1992] Bob Carpenter. *The Logic of Typed Feature Structures.* Cambridge University Press, 1992.

[Dörre and Dorna, 1993] Jochen Dörre and Michael Dorna. CUF: A Formalism for Linguistic Knowledge Representation. Dyana-2 deliverable, IMS, Stuttgart, Germany, August 1993.

[Erbach, 1995] Gregor Erbach. ProFIT: Prolog with Features, Inheritance and Templates. In *Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL'95)*, Dublin, Ireland, March 1995.

[Gerdemann and Götz, 1994] Dale Gerdemann and Thilo Götz. Troll: Type resolution system: User's guide. Research report, Universität Tübingen, 1994.

[Krieger and Schäfer, 1994] Hans-Ulrich Krieger and Ulrich Schäfer. *TDL*—a type description language for constraint-based grammars. In *Proceedings of COLING 94*, Kyoto, 1994.

[Manandhar, 1993] Suresh Manandhar. CUF in Context. In Jochen Dörre, editor, *Computational Aspects of Constraint-Based Linguistic Description I.* DYANA-2, Deliverable R1.2.A, August 1993.

[Manandhar, 1995] Suresh Manandhar. Deterministic Consistency Checking of LP Constraints. In *Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL'95)*, pages 165–172, Dublin, Ireland, March 1995.

[Meylemans, 1994] Paul Meylemans. ALEP - Arriving at the next platform. *ELSNews*, 3(2):4–5, 1994.

[Pollard and Sag, 1994] Carl Pollard and Ivan Andrew Sag. *Head-driven Phrase Structure Grammar.* Chicago: University of Chicago Press and Stanford: CSLI Publications, 1994.

[Reape, 1993] Mike Reape. Getting Things in Order. In Wietske Sijtsma and Arthur van Horck, editors, *Discontinuous Constituency.* Berlin: Mouton de Gruyte, 1993.

[Saraswat, 1993] Vijay Saraswat. *Concurrent Constraint Programming.* MIT Press, 1993.

[Smolka *et al.*, 1993] Gert Smolka, Martin Henz, and Jörg Würtz. Object-Oriented Concurrent Constraint Programming in Oz. Research Report RR-93-16, Deutsches Forschungszentrum für Künstliche Intelligenz, Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany, April 1993.

[Zajac, 1992] Rémi Zajac. Inheritance and Constraint-Based Grammar Formalisms. *Computational Linguistics*, 18(2):159–182, 1992.