Appears in C. Brown and G. Koch (1991),
Natural Language Understanding and Logic Programming, III
Elsevier, Amsterdam

A Logic-Based Implementation of Head-Driven Phrase Structure Grammar

F. Popowich and C. Vogel

School of Computing Science, Simon Fraser University, Burnaby, British Columbia, V5A 1S6, CANADA

Abstract

Head-Driven Phrase Structure Grammar (HPSG), a unification-based formal language for describing linguistic phenomena, has a declarative semantics which makes it amenable to specification as a logic program. The HPSG formalism has undergone significant modification, becoming more declarative and incorporating greater lexicalization, since Proudian and Pollard first introduced a LISP-based HPSG chart parser in 1985. These theoretical developments have led us to the Prolog implementation of an HPSG interpreter which is described in this paper. We provide a brief introduction to the HPSG formalism, and then illustrate how a natural language grammar based on this formalism can be stated in a declarative notation that is directly interpretable by a Prolog chart-parser. This process leads us to refinements in both the specification of the formalism and the interpretation methodology.

1. INTRODUCTION

Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1987) is in the family of linguistic theories known as unification grammars (UGs) (Shieber, 1986). The construction of a particular UG is analogous to the implementation of a logic program since both formal systems are founded upon the use of unification as a principle operator. Indeed, UGs, presented in a language whose syntax is the structure of recursive attribute value matrices (AVMs), can be semantically grounded in formal logic (Kasper and Rounds, 1986). The construction of a UG can be seen as the axiomatization of a formal system[1]. Thus, it is not surprising that aspects of logic programming (i.e., its declarative semantics and capacity for rapid prototyping) are, similarly, aspects of UG construction. In this paper we make explicit the connection between UGs and logic programs by constructing a UG system for a specific unification grammar, HPSG, in which we rely on techniques fundamental to logic programming. We describe the construction of a Prolog-based HPSG analyzer which incorporates a chart parser. Our system is therefore a tool which allows us to analyze HPSG as an instance of a logic program.

There are numerous systems (cf., Shieber, 1985; Karttunen, 1986; Seiffert, 1987; Hirsh,

---

[1] This is a separate issue from the axiomatization's use in a parsing mechanism, though certainly an automated theorem prover can be used to implement a parser by doing proofs relative to the axiomatization.

1988) for parsing unification grammars in general, and Proudian and Pollard (1985) have provided a LISP-based system for HPSG in particular. But, HPSG has undergone quite a bit of development since the original implementation. The number of grammar rules has been reduced from sixteen (at least, this is the number included in Proudian and Pollard's 1985 implementation of the theory) to four (Pollard and Sag, 1987), and their theoretical foundations have been reformulated: no longer are they context free string rewrites augmented with unification of shallow feature structure matrices, now the rules themselves are feature structures which describe immediate dominance relationships between constituents, and separate principles constrain constituent ordering (cf., Gazdar et al., 1985). Keeping in mind the development of the theory towards the declarative statement of constraints on feature structures, through the new use of rules and universal principles, it is fruitful to reconsider its implementation.

2. HPSG

HPSG is a grammar formalism that incorporates aspects of traditional lexical grammar formalisms like *categorial grammar* (Oehrle, Bach and Wheeler, 1988) along with some aspects of contemporary linguistic theories like *generalized phrase structure grammar* (Gazdar et al, 1985). Before describing our characterization of HPSG as a logic program, we must first highlight some of the distinctive features of this unification-based grammar formalism.

2.1. Signs

The fundamental unit of discourse in HPSG is the *sign*, which is represented with attribute value matrices (AVMs). The attributes and values of a sign contain phonological, syntactic and semantic information of linguistic constituents. For our discussion, we need consider only a portion of these features.

The top level attributes of signs are PHON, SYN and SEM which indicate phonology, syntax, and semantics. Signs are partitioned into phrasal signs and lexical signs. The distinction is that phrasal signs also have a top level feature called DTRS which contains constituency information, while lexical signs do not have this feature. HPSG uses the DTRS feature of phrasal signs to represent the constituent structure of a phrasal sign in terms of its head daughter (HEAD-DTR), its complements (COMP-DTRS), and its adjuncts (ADJ-DTRS). There are other types of daughters which need not concern us here.

SYN values are classified into LOCAL and NONLOCAL features. Important LOCAL features are HEAD, SUBCAT and LEX. HEAD features record syntactic information usually associated with words, information like major category in linguistic classification, form (relative to category), case, aspects of agreement, and constraints on adjuncts. SUBCAT is a list of other signs, in decreasing order of obliqueness, with which a phrasal sign needs to combine in order to be saturated with respect to its classification in a subsumption hierarchy. LEX is a binary valued feature which correlates but does not coincide with the distinction between lexical and phrasal signs (Pollard and Sag, 1987, p.73).

Using these features, we may construct a sign to describe the lexical entry for "does" in Figure 1. Note that we have abbreviated this lexical sign by removing the NONLOCAL and the SEM attributes. The appearance of *VP[BSE]* and *NP[NOM]* in the SUBCAT list indicates that "does" must combine with a base-form verb phrase and a nominative noun phrase to produce a "complete" constituent. *VP[BSE]* and *NP[NOM]* are actually abbreviations for the signs introduced in Figure 2 (Pollard and Sag, 1987, p.69). Signs can be ordered by relative informedness into a lattice structure such that a given abbreviation stands for the class of signs represented by a node in this hierarchy (and the class of all

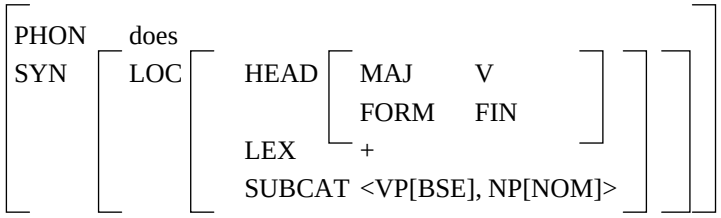
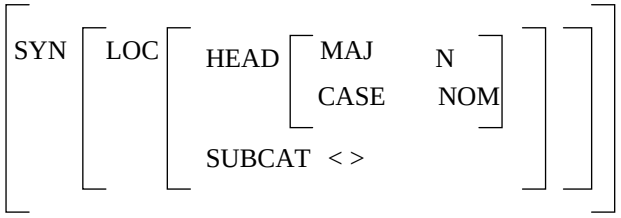
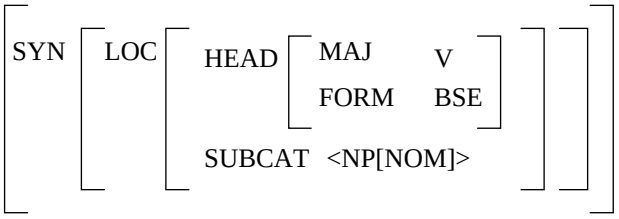


Figure 1. Lexical Entry for "does".



NP[NOM]



VP[BSE]

Figure 2. Sign Abbreviations.

subsumed signs). This ordering enables a formal specification of lexical types.

2.2. Lexical Types

A hierarchy of lexical types allows the specification of complex[2] types in terms of less complex types higher in the hierarchy. This eliminates considerable redundancy from the lexicon since individual lexical entries can be characterized by inheritance over elements from the lexical hierarchy rather than by their complete (explicit) specifications as signs.

Following Pollard and Sag (1987, §8), the lexical type *sign* is the root of lexical hierarchy. This type has two subtypes, *lexical-sign* and *phrasal-sign*. Lexical-signs are either *major-lexical-signs* or *minor-lexical-signs*, with the former being partitioned based on either

---

[2] We say that a sign *A* is more complex than a sign *B* if *A* is subsumed by *B* (i.e., *A* is more informed than *B*).

the value of their HEAD feature (i.e., *noun*, *adjective*, *verb* or *preposition*) or on the value of their SUBCAT feature (i.e., *saturated* or *unsaturated*). One can define a subtype like *v-trans* in terms of the types *verb* and *unsaturated* which would result in its inheritance of all the features specified in those two types (and all their supertypes). A particular lexical entry like *walk* could then be defined in terms of *v-trans*, also inheriting supertype information. Inheritance is facilitated by two types of path based reasoning (cf., Touretzky, Horty and Thomason, 1987) over this multiple inheritance hierarchy, *normal* (shortest path reasoning) and *complete* (fully skeptical reasoning) (Shieber, 1986, Flickinger, 1987).

Lexical types are not the only means of structuring information in the lexicon. Dependencies between two lexical entries (or lexical types) can also be stated with the use of *lexical (redundancy) rules*. We will not be concerned with these rules here, as there is some question of their necessity (Flickinger, personal communication).

2.3. Grammar Rules and Principles

HPSG, as a linguistic theory, specifies constraints on the combination of signs. A basic tenet of HPSG is that the HEAD-DTR of a phrasal sign is itself a sign. Syntactic and semantic information is shared between those two signs, as well as among the other complements and adjuncts of the HEAD-DTR, according to the constraints imposed by the few grammar rules and the universal and language specific principles. The rules and principles are expressed as information structures—they are stated in terms of signs just as are lexical entries.

The grammatical principles independently constrain *information* contained in signs. The Head Feature Principle restricts the sharing of HEAD information between a mother sign and its HEAD-DTR. The Subcategorization Principle mediates SUBCAT information between a mother sign and its HEAD-DTR in terms of the mother sign's COMP-DTRS. The Semantics Principle defines the sharing of SEM information among a mother sign and all of its daughters, as does the English Constituent Ordering Principle for PHON information.

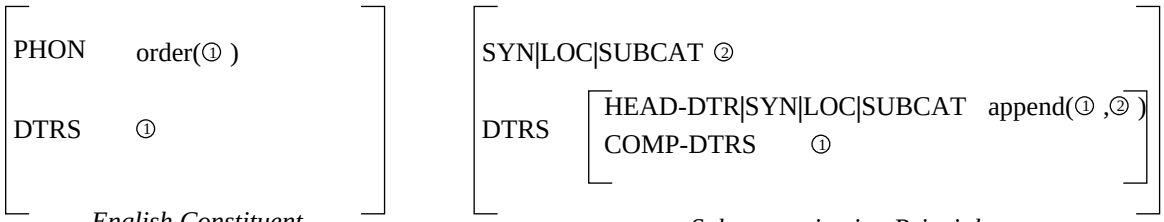


Figure 3. Constituent Ordering and Subcategorization Principles.

As the Subcategorization and Constituent Ordering Principles are essential to our discussion, they are illustrated in Figure 3. In these principles, re-entrancy (i.e., the sharing of a value by two features) is denoted by a common index. These signs follow the abbreviatory convention adopted by Pollard and Sag (1987) which allows reference to values inside nested AVMs by writing the paths of attribute names (separated by vertical bars) which lead to these values. These principles are assumed to apply in conjunction with any grammar rule. In the theory, the principles are stated using the relative pseudocomplement operator. The details of this operator need not concern us here, but the net effect is that the

principles all unify with each other into a single relation between a vacuous headed structure antecedent and a more fully specified consequent (Pollard and Sag, 1987). Applied to these two information structures, the relative pseudocomplement operator yields another information structure which contains all of the constraints imposed by each of the principles. Furthermore, any sign unifiable with the information structure for a rule must also be unifiable with this resultant structure to be considered valid. We take advantage of the mapping provided by the operator in our implementation of the system.

The Subcategorization Principle enforces the following constraint on phrasal signs: the first elements (the first elements are the most oblique elements) of the SUBCAT list of the HEAD-DTR of the phrasal sign are also the COMP-DTRS of the phrasal sign; the SUBCAT list of the phrasal sign takes its value from everything else on the SUBCAT list of the HEAD-DTR (the least oblique elements). Let A, B, C, and D be signs. Let Σ be a phrasal sign that unifies with the Subcategorization Principle whose SUBCAT list has the value <C, D> and whose COMP-DTRS list has the value <A, B>, then Σ will have a HEAD-DTR whose SUBCAT list has the value <A, B, C, D>. The Constituent Ordering Principle uses the function *order* to articulate the constraints between constituent structures of a phrasal sign and the phonology of the phrasal structure. Informally, the phonology of a phrasal sign can be defined as the combination of the phonologies of the constituent daughters. If the HEAD-DTR is lexical, the phonology of the HEAD-DTR will precede the phonologies of the COMP-DTRS in the combined phonology of the mother sign, just as "kissed" precedes "the cat" in the sentence "Mary kissed the cat." If the HEAD-DTR is not lexical, then the phonologies of the COMP-DTRS will precede the phonology of the HEAD-DTR, as "Mary" precedes the phrase, "kissed the cat." In the case of adjuncts, the phonology of lexical ADJ-DTRS precede the phonology of the HEAD-DTR as in "blue book", and non-lexical ADJ-DTRS follow the head as in "car in the shop". Some controversy surrounds the exact statement of the principle within the formal structure of HPSG, but that debate is outside the scope of this paper.

The "rules" of HPSG, as shown in Figure 4, constrain the *structure* of constituency. The first rule is responsible for combining heads with their final complement (i.e., their "subject"). The second rule is for combining lexical heads with all but their final complements. The third rule handles lexical heads in inverted constructions where they combine with all of their complements including the subject. Various versions of a fourth rule for treating adjuncts are tentatively put forward in (Pollard and Sag, 1987). The version that we have adopted is responsible for combining a head with a single adjunct — multiple adjuncts are handled by multiple applications of this single rule. Each of the grammar rules is a sign that is also subject to the informational constraints imposed by the grammatical principles. Since all objects of discourse expressed using the formalism (phrase structures, lexical entries, rules, principles) are expressed as signs, a natural way to specify the relationships among these objects and to describe the accumulation and satisfaction of constraints imposed by each object is to arrange them as nodes in an inheritance hierarchy beyond the lexical hierarchy (Vogel and Popowich, 1990).

3. IMPLEMENTATION

Given the above description of HPSG, our goal now is to convert an HPSG grammar (ie., the lexical entries, the lexical types, the principles and the grammar rules) into an equivalent Horn logic description (using Prolog notation) that can be analyzed by a Prolog chart-parser.

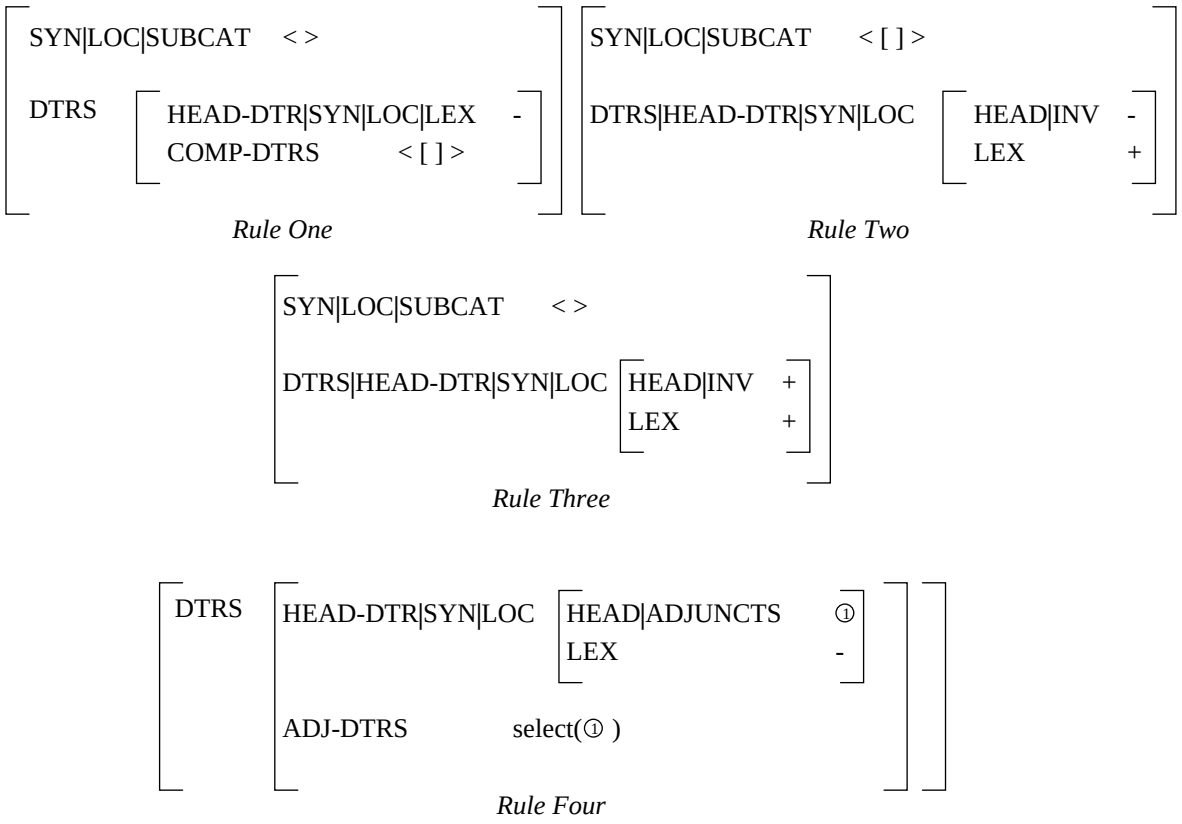


Figure 4. Grammar Rules.

3.1. The Grammar

In our conversion of an HPSG grammar into a set of Prolog clauses, we convert the AVMs appearing in signs into fixed-arity lists, not into directed acyclic graphs (DAGs) as described in Pollard and Sag (1987, §2). Although this means that the different possible features must be declared ahead of time (cf., Alshawi et al, 1988; Hirsh, 1988; Moens et al, 1989; McFetridge and Cercone, 1990) (either explicitly or implicitly) and must have different list positions associated with them (cf., Shieber, 1986) fixed arity term representation does have the advantage of more efficient unification during parsing.[3] Alternatively, one could base the implementation of the grammar on DAGs and have a general purpose unification framework like PATR (Shieber et al, 1983, Karttunen, 1986, Hirsh, 1988) underlying the implementation. Unfortunately, PATR would require some extensions (e.g., for handling functions) before it could capture all the different aspects of HPSG. Prolog possesses all the facilities needed for a concise characterization of HPSG.

A sign is represented as a list of attribute-value pairs. We call this list representation of a sign an *i-sign* (implementation-sign). An attribute-value pair is encoded as a two element

---

[3] AVMs can be implemented in Prolog as partially specified lists over which an AVM unification procedure operates (Gazdar and Mellish, 1989, §7). With fixed arity lists, we can use Prolog's term unification directly.

list. Values for certain attributes in HPSG like SUBCAT which are defined as lists are also implemented as Prolog lists.[4] There are some notational differences between signs and i-signs. For instance, in an HPSG sign an attribute may have a function as its value. A corresponding i-sign has the same attribute with a variable as its value that is later instantiated to the value of the function. Additionally, we specify that lexical i-signs have a null-valued *dtrs* attribute rather than no DTRS attribute at all.

In implementing the lexical hierarchy and the inheritance mechanism over that multiple inheritance hierarchy, we chose to implement only the complete mode of inheritance, since shortest path reasoning has been demonstrated by Touretzky (1986) to be unsound. In our system complete inheritance—fully skeptical reasoning—is implemented by compiling the hierarchy of type specifications into full information structures in search of unification failure (type conflicts). Where type conflict exists for nodes in the hierarchy, those nodes are thrown out as ambiguous.

In our definition of lexical types, we will also make use of what we will call *attribute-value (AV)* types. Our definitions of AV types play a fundamental rule in declaring the structure of an i-sign. The definitions of lexical types and of AV types collectively describe the list structures that implement signs and attribute-value pairs. The form of the definitions is reminiscent of that used by Mellish (1988) and by Moens et al. (1989). Circular type definitions are not allowed, nor are forward references to types (a type definition can reference only those types defined before it).

Figure 5 shows a selection of type definitions that describe a simplified version of an HPSG sign. First note that these type declarations are just concerned with some aspects of the *syntax* attribute of the sign. For many of the AV types, especially those that take atomic values, type restrictions are not placed on the values. This is because the primary purpose of our AV type definitions is to define the different *attributes* in an AVM. Other *values* are not typed in order to avoid recursive type definitions. For example, the value of *subcat* should be specified to be a list of signs, each of which would contain a *subcat* feature.[5] Note that the list structures corresponding to AV types can be distinguished from those for lexical types since the first element of the former is an atom. Thus, all of the definitions shown in Figure 5 are for AV types except *sign*, *psign* (phrasal sign), *lsign* (lexical sign), *csign* (complement sign) and *asign* (adjunct sign). Wherever an expression of the form "*@type*" appears in the right-hand side of a declaration, the expression is replaced by the appropriate definition for *type*. Thus, "*@type*" is used to indicate a node in the hierarchy from which information is to be inherited during the compilation process. This inheritance is carried out before any parsing is attempted. The type definitions for the lexical types *psign*, *lsign*, *csign* and *asign* contain PATR-like path equations for specifying values within the i-signs being defined. These path equations may make use of any of the features defined by the type definitions.

Our type declarations reflect some minor modifications to the HPSG sign structure. For instance, instead of *slash* taking a sign (or list of signs) as its value, it takes a feature structure containing semantic and local syntactic information, thus avoiding a recursive type

---

[4] Our initial prototype used difference lists as the values for the SUBCAT and COMP-DTRS features because difference lists allow easy partitioning, which is useful for verifying that a sign satisfies the Subcategorization Principle (cf. Figure 3). However, the use of difference lists complicates the sign unification procedure with an occurs check, decreasing efficiency. The existing system does not use difference lists.

[5] A more sophisticated approach would involve adopting the lazy type expansion strategy of STUF (Uszkoreit, 1987, Bouma, König and Uszkoreit, 1988).

```
phon     type [phon, _].
sem      type [sem, _].
maj      type [maj, _].
form     type [form, _].
inv      type [inv, _].
aux      type [aux, _].
case     type [case, _].
adjuncts type [adjuncts, _].
head     type [head,
          [@maj, @form, @inv, @aux, @case, @adjuncts]].
subcat   type [subcat, _].
lex      type [lex, _].
loc      type [loc, [@head, @subcat, @lex]].

syns1    type [syn, [@loc]].
slash    type [slash, [@syns1, @sem]].
rel      type [rel, _].
que      type [que, _].

inher    type [inher, [@slash, @rel, @que]].
to_bind  type [to_bind, [@slash, @rel, @que]].
non_loc  type [bind, [@inher, @to_bind]].

syn      type [syn, [@loc, @non_loc]].
dtrs     type [dtrs, _].

sign     type [@dtrs, @phon, @syn, @sem].

head_dtr  type [head_dtr, @sign].
comp_dtrs type [comp_dtrs, _].
adj_dtrs  type [adj_dtrs, _].
fil_dtrs  type [fil_dtrs, []].   % fillers not yet implemented

psign  type @sign with dtrs =
       [@head_dtr, @comp_dtrs, @adj_dtrs, @fil_dtrs].
lsign  type @sign with dtrs = [].

% Complement Sign (no adjuncts)
csign  type @psign with dtrs:adj_dtrs = [].

% Adjunct Sign (no complements)
asign  type @psign with dtrs:comp_dtrs = [].
```

Figure 5. Type Declarations.

definition.

A selection of lexical type declarations are introduced in Figure 6. Observe that the type *sat* (saturated) is defined to be a *@sign* with an empty subcategorization list. This type, along with the one for *noun* is used in the definition of the *parameterized* type for an *np*. Our definition of a parameterized type (cf., Bouma, 1990) introduces a variable *Case* that corresponds to some value within the i-sign. This parameterized type is used in the declaration of the *intrans* and *trans* types in describing the structure of a subcategorization list. The definition of *trans* uses the *np* template twice, once to introduce a nominative case *np*, and again to describe an objective case *np*. The lexical types from Figure 6 are used in the lexical entries introduced in Figure 7. Observe that most of the information about a lexical entry is contained in the hierarchy of types referenced by the definition.

```
sat     type @sign with syn:loc:subcat = [].
unsat   type @sign with syn:loc:subcat = [_|_].
no_adjn type @sign with syn:loc:head:adjuncts = [].

det     type @sign & @sat & @no_adjn with syn:loc:head:maj = d.
prep    type @sign & @unsat & @no_adjn with syn:loc:head:maj = p.
verb    type @sign with syn:loc:head:maj = v.
noun    type @sign with
   syn:loc:head:form = norm with
   syn:loc:head:maj = n.

np(Case) type @noun & @sat with syn:loc:head:case = Case.
common  type @noun & @unsat with
   syn:loc:subcat = [@det] with
   syn:loc:head:adjuncts = [@adj,@pp(on)].

intrans  type @unsat with syn:loc:subcat = [@np(nom)].
trans    type @unsat with syn:loc:subcat = [@np(obj),@np(nom)].

mainv(Form) type @verb with
   syn:loc:head:adjuncts = [@pp(on)] with
   syn:loc:head:aux = minus with
   syn:loc:head:inv = minus with
   syn:loc:head:form = Form.

word(Phon) type @lsign with
   phon = [Phon] with
   syn:loc:lex = plus with
   syn:loc:head:inv = minus.

det(Phon)            type @word(Phon) & @det.
transv(Phon,Form)    type @word(Phon) & @trans & @mainv(Form).
intransv(Phon,Form)  type @word(Phon) & @intrans & @mainv(Form).
```

Figure 6. Lexical Type Declarations.

```
entry @adj(good).
entry @adj(small).

entry @cn(cat).
entry @cn(cookie).
entry @cn(table).

entry @det(the).

entry @pn(john).
entry @pn(kim).
entry @pn(mary).

entry @preposition(on).

entry @auxv(does).

entry @intransv(walks,fin).
entry @intransv(walk,bse).
entry @transv(eats,fin).
entry @transv(eat,bse).
entry @transv(loves,fin).
entry @transv(love,bse).
entry @ditransv(gives,fin).
entry @ditransv(give,bse).
entry @icontrolv(to,inf,bse).
entry @icontrolv(seems,fin,inf).
entry @icontrolv(seem,bse,inf).
entry @icontrolv(tries,fin,inf).
entry @icontrolv(try,bse,inf).
entry @tcontrolv(persuades,fin,inf).
entry @tcontrolv(persuade,bse,inf).
entry @tcontrolv(believes,fin,inf).
entry @tcontrolv(believe,bse,inf).
```

Figure 7. Lexical Entries.

Just as types are used in the specification of the lexical entries, they can also be used in the formulation of the universal principles and grammar rules of HPSG. As mentioned in § 2.3,

```
hfp     type @psign with              % Head Feature Principle
   syn:loc:head = Sign with
   dtrs:head_dtr:syn:loc:head = Sign.

subp    type @psign with              % Subcategorization Principle
   syn:loc:subcat = Rest with
   syn:loc:lex = minus with           % modification to SUBCAT principle
   dtrs:comp_dtrs = Comps with
   dtrs:head_dtr:syn:loc:subcat = append(Comps,Rest).

cop     type @psign with              % Constituent Ordering Principle
   dtrs = Dtrs with
   phon = order(Dtrs).

principles type @hfp & @subp & @cop. % Unification of Principles
```

Figure 8. Universal Principles.

we will not be stating the principles in terms of the relative pseudocomplement operator. We can get the same effect by introducing the principles as constraints associated with the grammar rules: the principles are phrasal signs which are unified with each of the grammar rules (which are also phrasal signs). Adopting this approach, the Subcategorization Principle (subp), the Head Feature Principle (hfp), and the Constituent Ordering Principle (cop) can be defined in terms of the types introduced in Figure 8. We also define the *principles* type as the unification of all the grammar principles.

Observe that the value of the *phon* attribute in the Constituent Ordering Principle is a function, as is the value of the head daughter's *subcat* attribute in the Subcategorization Principle. In § 3.2 we shall see how such functions are actually processed.

The Subcategorization Principle as introduced in Figure 8 differs from the version introduced in Figure 3 in that it specifies *lex* to be *minus*. Informally, it captures the intuition that the constituent that results from combining a head with some (or all) of its complements is nonlexical.[6] Without this modification, there is nothing to prevent repeated application of Rule Two to a phrasal sign constructed from Rule Two: since neither Rule Two nor the principles specifies a value for the LEX feature of the phrasal sign, and since Rule Two may be invoked with an empty COMP-DTRS list, derivations containing an unbounded sequence of Rule Two invocations are allowed. Similarly, a phrasal sign resulting from the invocation of Rule One could be subject to repeated applications of Rule Three. There would also be related problems with applications of Rule Four. These problems are all prevented by our modification to the Subcategorization Principle.

The definitions of the grammar rules are included in Figure 9. Each grammar rule references the type *principles* and thus inherits the constraints imposed by all of the grammar principles[7]. Observe that the value of the *subcat* attribute is specified to be the empty list for

---

[6] In a larger grammar, we may want to associate the LEX feature with the grammar rule instead of the Subcategorization Principle. Aspects of the LEX feature are discussed in (Pollard and Sag, 1987, §3.3).

[7] While the syntax of declarations for rules, types, and lexical entries is different for each as can be seen from Figures 7, 8, and 9, this is just notational sugar to allow for a more efficient system: all could have been declared using the same syntax as that used to construct the lexical hierarchy.

```
1       rule    @principles & @csign with
                dtrs:head_dtr:syn:loc:lex = minus with
                syn:loc:subcat = [] with
                dtrs:comp_dtrs = [_].

2       rule    @principles & @csign with
                dtrs:head_dtr:syn:loc:lex =  plus with
                dtrs:head_dtr:syn:loc:head:inv = minus with
                syn:loc:subcat = [_].

3       rule    @principles & @csign with
                dtrs:head_dtr:syn:loc:lex =  plus with
                dtrs:head_dtr:syn:loc:head:inv = plus with
                syn:loc:subcat = [].

4       rule    @principles  & @asign with
                dtrs:head_dtr:syn:loc:lex = minus with
                dtrs:head_dtr:syn:loc:head:adjuncts = A with
                dtrs:adj_dtrs = select(A).
```

Figure 9. Grammar Rules.

rules 1 and 3, while it is required to contain a single element in rule 2. The *comp_dtrs* attributed of rule 1 is also specified as having a single element. The *adj_dtrs* attribute of rule 4 is assigned a list containing a single sign which is *selected* from the head daughter's *adjuncts* list. Note that this selection process, through an invocation of the *select* function, results in the creation of a copy of the selected sign, thus avoiding a destructive modification to the head daughter's adjunct list. Rules 1, 2 and 3 are specified to be of the *csign* type: these rules are applicable to heads taking only complement daughters and thus result in a phrasal sign with an empty *adj_dtrs* attribute. Rule 4 is of type *asign*: the phrasal sign takes an adjunct daughter and has an empty *comp_dtrs* attribute. The rule definitions are expanded into i-signs that can be used directly by the HPSG chart parser.

3.2. Chart Parsing HPSG

We have implemented a parser (Popowich and Vogel, 1990) using modifications to the chart parsing paradigm due originally to Kay (1973) and Kaplan (1973), and more clearly described by Thompson (1981) and by Gazdar and Mellish (1989). The chart parser is written in Prolog, incorporating results from a similar system (Popowich, 1989a) for a unification grammar formalism related to HPSG, called Tree Unification Grammar (Popowich, 1989b).

The algorithm represents a sentence with a chart, where nodes in the chart represent positions between words in the sentence being parsed. Edges between nodes represent analyses of well-formed substrings of the sentence. An edge is marked with the i-sign (also called the edge's *category*) for the analyses the edge represents. The endpoints of an edge indicate its position in the chart and the span of its analysis. New edges representing analyses of larger constituent structures of the sentence are introduced to the chart through a waiting list (the *agenda*) as the product of one of two processes. The Predictor step determines that the category associated with some edge satisfies the *head-dtr* of some grammar rule and creates new edges for each rule satisfied in this fashion. These new edges are placed on an agenda of edges to be processed. The new edges each have an associated list of expectations, i-signs that the new edge needs to combine with. An edge that has a non-empty expectations list is an *active* edge. The expectations list of the new edge is taken from the *comp-dtrs* and *adj-dtrs* features of the i-sign that marks the category of the new edge. The Completer

compares the next edge on the agenda (also called the current edge) with the edges in the chart, looking for edges that "meet" the current edge satisfying some of the expectations of one of the edges. A new composite edge with some or all of its expectations satisfied is created and added to the agenda. When an edge is created that spans the chart and has all of its expectations satisfied a successful parse has been created.

The procedure follows:

1. Take the next edge from the agenda, this is the *current* edge. Determine whether the current edge spans the chart.

2. Apply the Predictor Step to the current edge using each grammar rule. At most one edge will be generated from each grammar rule, and each new edge will satisfy the grammar principles as well. New edges are added to the agenda for future processing.

3. Apply the Completer Step to the current edge and the chart. The current edge may meet an edge in the chart from either the right or the left. If the two edges meet, one of them active and the other inactive, with some or all of the expectations of the active edge satisfied, a new edge is created and added to the agenda with the same category and remaining expectations of the active edge. The new edge spans the two constituent edges.

4. Add the current edge to the chart.

Note that in the version of HPSG implemented by Proudian and Pollard, ordering information was encoded implicitly in the string rewriting notation of phrase structure rules. However, the most recent version of HPSG has an explicit Constituent Ordering Principle which has been abstracted away from the rules. In this version of the theory the current expectation (complement) need not be adjacent to (i.e., meet) the head at all. This occurs in inverted constructions like "Did Mary walk?" in which the current expectation, corresponding to the most oblique complement using the ordering of the SUBCAT list, "walk," is separated from the head, "Did," by another word. Our approach, specific to the HPSG analysis of English, is to generate expectations at an edge from the SUBCAT list so that the current expectation is always textually adjacent to the head. That is, we create an expectations list which is the reverse of the subcategorization list. In the general case, the parser requires a less restricted definition of "meets" with additional constraint checking to verify the correspondence between analyzed substrings and the original string.

Incorporation of i-signs into edges can proceed either explicitly, storing on any given edge the mother i-sign which contains all information present at the given level of analysis, or implicitly, where *dtrs* information is constructed after the parse in accordance with the rules and principles of HPSG. While the latter approach is more space efficient, the former approach eliminates a step in processing (Seiffert, 1987). Our prototype system adopts the latter approach. Thus, at each edge we store the i-sign which corresponds in informativeness to the sign for the sentence at that particular level of analysis; this is the edge's category. But we do not record the constituent structure contained in the *dtrs* attribute. Instead, the constituent structure can be retrieved upon the identification of a spanning edge by tracing back through the edge numbers of those used to create the spanning edge. The system includes a toggle which can be set to force the parse to record complete phrasal i-signs explicitly, which is useful during debugging. However, as we shall see shortly, the parser is significantly more efficient when it is set to record i-signs implicitly. The system can also be directed to reconstruct the complete phrasal i-signs for successful parses from the implicit representation.

As is clear from the presentation of signs for the various principles of HPSG, functions are

included in the language to specify constraints on values. I-signs also allow this, even though Prolog does not support functions, simply by specifying a value as a variable to be instantiated when the relational constraints in which the variable participates are evaluated (Pereira and Shieber, 1987). These relations are recorded in edges as a list of pending restrictions as highlighted in Figure 10. When a new edge is created through either the

```
edge([ [dtrs, DTRS],
       [phon, VALUE],...],
     CONSTITUENT_LIST,
     EXPECTATIONS_LIST,
     [order(DTRS, VALUE), ...],
     LEFT_POS,
     RIGHT_POS,
     EDGE_ID).
```

Figure 10. An Active Edge with a non-Empty List of Relational Restrictions.

Completer or Predictor steps, the pending restrictions that have accumulated from its component edges are examined. Any restrictions that can be evaluated are then evaluated, with the remaining restrictions being associated with the new edge.

4. EXAMPLE

Consider the following analysis of the question, "Does John love Mary?" We begin by initializing the agenda with four edges (edges pass through agenda before being added to the chart (cf., Thompson, 1981)). The category of each corresponds to the i-sign associated with the lexical entry for each word in the sentence. None of the initial edges have expectations associated with them.

A portion of the output from our system has been included in Figure 11 to demonstrate the sequence of edges considered. It indicates information contained at edges as they are constructed and placed in the agenda for processing, not as they are actually inserted into the chart. Thus, we see that edge (4) whose category is the lexical i-sign for "Mary" is entered into the agenda, a stack, but we do not see any of its further processing and insertion into the chart. The next edge to be processed is edge (3), marked with the lexical i-sign for "love". Notice that the i-sign that marks the category of edge (5) has the same phonology of the i-sign on edge (3), but we see that the category of edge (5) is a phrasal i-sign obtained during the predictor step by applying the lexical i-sign for "love" that marks edge (3) as the *head-dtr* of Rule Two. Since this also implies that the grammar principles are satisfied, the *comp-dtrs* feature on the i-sign for edge (5) is a non-empty list encoded as the expectations list for the edge as discussed above. No adjuncts are included on the expectations list because Rule Four has not applied, and according to the hierarchy of types given in Figure 5 the *adj-dtrs* of Rule Two is an empty list. Constraints associated with the various principles (i.e., constituent ordering) carried in the i-sign for Rule Three are also annotated on the new active edge. In the predictor step during later processing, edge (1) is used as the basis of two additional edges (10 and 11), through the satisfaction of two grammar rules. Edge (11) is also active, containing two elements on its expectations list in increasing order of obliqueness (thus, it expects to meet its subject first, and its VP second; cf. Figure 1).

During the completer step, this new edge (11) will meet the inactive edge (2) associated with "John." The i-sign category of the inactive edge will unify with the least oblique

```
| ?- parse.
|: does john love mary.
initializing: stack inactive edge 1 ([does]), built from entry does.
initializing: stack inactive edge 2 ([john]), built from entry john.
initializing: stack inactive edge 3 ([love]), built from entry love.
initializing: stack inactive edge 4 ([mary]), built from entry mary.
predictor: stack active edge 5 ([love]), built from edge(3) and rule(2).
completer: stack inactive edge 6 ([love,mary]), built from edge(5) and
   edge(4).
predictor: stack active edge 7 ([love,mary]), built from edge(6) and
   rule(1).
predictor: stack active edge 8 ([love,mary]), built from edge(6) and
   rule(4).
completer: stack inactive edge 9 ([john,love,mary]), built from edge(2) and
   edge(7).
predictor: stack active edge 10 ([does]), built from edge(1) and rule(2).
predictor: stack active edge 11 ([does]), built from edge(1) and rule(3).
completer: stack active edge 12 ([does,john]), built from edge(11) and
   edge(2).
completer: stack inactive edge 13 ([does,john,love,mary]), built from
   edge(12) and edge(6).
Parse Found at 0.767 secs.

Done at 0.917 secs.
```

Figure 11. Some Output from the Chart Parser.

expectation from the active edge, resulting in a new, combined edge (12) which contains the remaining expectations. Note that if the expectations list corresponded in order directly to SUBCAT, the subject which is the least oblique complement, would be the last expectation, and without a generalization to the definition of "meets," the two edges would not meet. The new active edge (12) which still has the expectation for a VP complement will later meet the inactive edge (6) associated with "love Mary." The i-sign of the inactive edge will unify with this remaining expectation to yield an inactive edge (13) that spans the entire sentence. Constraints are evaluated as their arguments become instantiated. That the accumulated relational restrictions were met is implicit in the addition of the new edge to the stack. The new edge will have a null restrictions list.

Other edges will also be constructed, although they do not ultimately combine into a spanning edge. For instance, an analysis of the substring "John love Mary" is encoded in edge (9) since further processing of the inactive edge (6) for "love Mary," mentioned above, will create another active edge (7) in conjunction with Rule One during the predictor step. This edge will record the expectation of a least oblique complement which will be met by the edge corresponding to "John."

5. ANALYSIS

Based on the grammar and lexicon outlined in §3.1, we have constructed the following selection of test sentences:

1. Does Mary love John?
2. The cat walks on the table.
3. John eats the small cookie on the table.

4. John gives Mary the small cookie.

5. John persuades Mary to eat the good small cookie.

These sentences span the coverage intended by the rules and principles of Pollard and Sag (1987) with the exception of the Binding Inheritance Principle, which is discussed at length in Pollard and Sag (forthcoming), and the Semantics Principle. The first sentence illustrates inversion, requiring Rules Two and Three. Sentence 2 requires the application of Rules One, Two and Four. It is an ambiguous sentence since the adjunct *on the table* can either modify the verb phrase *walks* or the sentence *The cat walks*. The third sentence illustrates even greater ambiguity, having five different analyses: the adjunct can modify any constituent ending with the word *cookie*. Sentence 4 illustrates the use of a ditransitive verb. The last sentence introduces the transitive control (equi) verb *persuades*, and contains a construction in which two adjectives modify a noun, requiring two applications of Rule Four.

Table 1. Results from Test Runs.

Sentence	Grammar One			Grammar Two		
	Explicit	Implicit	Reconstructed	Explicit	Implicit	Reconstructed
1	2.2	1.0	1.4	1.8	0.9	1.2
2	21.6	4.8	8.5	10.9	3.0	4.5
3	204.0	22.2	54.5	92.4	11.7	25.7
4	10.6	3.1	4.3	7.1	2.4	3.5
5	67.9	7.4	10.8	52.5	6.1	9.2

Table 1 shows the amount of CPU time in seconds required to find all possible parses of our test sentences using Quintus Prolog (Quintus, 1990) on a SUN SPARCstation1. Each sentence is processed first with the explicit phrasal sign representation (Explicit), then with the implicit representation (Implicit) and finally with the reconstruction of complete phrasal signs from their implicit representations (Reconstructed). The sentences are parsed according to two different grammars which differ in their specification of Rule Four for adjuncts. The original grammar (One) contains the specification of Rule Four introduced in Figure 9. A modified grammar (Two) is also tested in which Rule Four is specified as an unsaturated phrasal sign whose SUBCAT list contains a single element. This limits the signs that can be modified by adjuncts to signs representing incomplete constituents, which excludes constructions involving modifiers of noun phrases and sentences (e.g. "Mary with the bright green eyes...."). Since certain signs cannot be sanctioned by the rule, they do not incur expenses associated with creating a sign, such as verifying the constraints stored on each edge, and certain ambiguous analyses are ruled out. Under grammar two, only sentence 3 is ambiguous: it has three analyses as opposed to the five analyses under grammar one.

The differences among timings are stark for the third sentence, which involves Rule Four. Since this is an ambiguous sentence and signs for numerous parses must be maintained, the storage of signs is quite significant, and there is about an eightfold difference in timings for parses that used explicit as opposed to implicit signs (using either grammar). In processing sentence 3, the modified grammar caused a fifty percent increase in efficency over the grammar with Rule Four unchanged. Results for the fifth sentence, which was the most complex unambiguous sentence, demonstrated similar behavior relative to the implicit/explicit distinction, and the modified grammar gave about a twenty percent increase in efficiency for both implicit and explict representations. In the first and fourth sentences, in

which ambiguity was not a problem, the modified grammar still performs significantly better than the unmodified grammar though not the fifty percent increase that emerged for both explicit and implicit signs when parsing the third sentence. Using implicit representation of signs for either grammar, the modified grammar gives better results. Implicit representation of signs offers a fifty percent savings in cpu time for sentence recognition and no less than a thirty-three percent savings if the complete phrasal sign description of a sentence is reconstructed from a parse. Statistics for implicit representation with sign reconstruction are included primarily for a clearer comparison with explicit representations. For many applications of the parser (e.g. a natural language interface to a database) the essential semantic information is contained in the root of the implicit representation on the spanning edge: with an implicit representation it is not necessary to reconstruct the full sign, although it is useful to reconstruct that sign when debugging a grammar.

6. DISCUSSION

We began by distilling HPSG grammar into an equivalent notation of i-signs in Prolog (essentially, we articulated HPSG as a logic program). This includes an implementation of a highly structured lexicon defined with reference to the multiple inheritance lexical hierarchy inherent in HPSG. In practical terms, implementing these levels of abstraction facilitates the system's use to experiment with lexicons, since any particular level is not cluttered with detail. Thus, the lexicon can be modified through inheritance from refinements to subtypes or by direct specification in lexical entries. Through this facility we have a tool for further exploration of HPSG. We were able to use it to recognize the need for minor modification to the grammar rules and principles, particularly with respect to the LEX feature.

The distillation also included the restructuring of the grammar principles as individual phrasal signs. We expressed the grammar rules such that they each inherit the information and constraints imposed by all of the principles. Additionally, we adopted a natural treatment of functions (for those functions stated in the HPSG articulation) to allow delayed evaluation as required by Prolog. While we described this approach only for the *order* function of the constituent ordering principle, but it also handles *append* and *select*. It will also cover the functions used in other principles, like the semantics principle when they are incorporated.

It was also gratifying to notice ways to structure the input to our interpreter (specifically, the structure of information recorded on edges in the chart parser) for more efficient processing of HPSG. We were able to avoid writing the generalized "meets" requirement of HPSG into the chart parser by structuring the expectations list as the reverse of the SUBCAT list. We were also able to save considerable space at run time by storing the constituent structure of signs implicitly in the chart. In addition to saving space, this made the system run more efficiently because of the resultingly limited size of structures to be processed during the unification of signs with expectations lists of edges in the Completer step of processing. Moreover, a consistent fixed-arity representation of signs allows the system to use built-in Prolog unification instead of a more complex AVM unification procedure. Additional tuning to the grammar itself gave further speed up.

Our ongoing work involves further investigation of HPSG as a formal language and encoding a larger portion of the HPSG grammar. We have incorporated a simple version of the semantics principle in which the SEM value of a phrasal sign is the unification of the SEM values of each of its daughters. Currently we are developing a more sophisticated version of the semantics principle, as well as extending our treatment of adjuncts. In the near future we will incorporate an additional rule and associated principle (Pollard and Sag, forthcoming) which together cover topicalized phrase structures. Concurrently, the results

from our logic-based implementation of HPSG are being incorporated into a LISP-based parsing system (McFetridge and Cercone, 1990) that forms the basis for a natural language interface to a database.

ACKNOWLEDGEMENTS

We would like to thank Dan Fass and the anonymous referees for their helpful comments on earlier drafts of this paper. We are also grateful to Mark Gawron and Paul McFetridge for numerous discussions about material contained herein.

The research presented in this paper is supported by a fellowship from the Advanced Systems Institute of British Columbia, and by the Natural Sciences and Engineering Research Council of Canada under Grant Nos. OGP0041910 and A4309.

REFERENCES

Alshawi, Hiyan, David Carter, Jan van Eijck, Robert Moore, Doug Moran and Steve Pulman. (1988). Overview of the Core Language Engine. In Institute for New Generation Computer Technology (Ed.), *Fifth Generation Computer Systems 1988, Proceedings of the International Conference on Fifth Generation Computer Systems 1988*. Springer-Verlag, New York, NY.

Bouma, Gosse. (1990). Non-Monotonic Inheritance and Unification. In W. Daelemans and G. Gazdar (Eds.), *Inheritance in Natural Language Processing Workshop Proceedings*. Institute for Language Technology and Artificial Intelligence, Tilburg University, Holland.

Bouma, Gosse, Ester König, and Hans Uszkoreit. (1988). A Flexible Graph-Unification Formalism and its Application to Natural Language Processing. *IBM Journal of Research and Development, 32*, 170-184.

Flickinger, Dan. (1987). *Lexical Rules in the Hierarchical Lexicon*. Doctoral dissertation, Stanford University, CA.

Gazdar, Gerald, and Chris Mellish. (1989). *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*. Addison-Wesley Publishing Company, Menlo Park, CA.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum, and Ivan Sag. (1985). *Generalized Phrase Structure Grammar*. Basil Blackwell, London, England.

Hirsh, Susan. (1988). P-PATR: A Compiler for Unification-based Grammars. In V. Dahl and P. Saint-Dizier (Eds.), *Natural Language Understanding and Logic Programming, II*. Elsevier Science Publishers B.V., North-Holland.

Kaplan, Ron. (1973). A General Syntactic Processor. In R. Rustin (Ed.), *Natural Language Processing*. Algorithmics Press, New York, NY.

Karttunen, Lauri. (1986). D-PATR: A Development Environment for Unification-Based Grammars. In: *Proceedings of the 11th International Conference on Computational Linguistics*. Institut fuer Kommunikationsforschung und Phonetik, Bonn University, FRG.

Kasper, Robert, and William Rounds. (1986). A Logical Semantics for Feature Structures. In: *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*. Columbia University, New York, NY.

Kay, Martin. (1973). The MIND System. In R. Rustin (Ed.), *Natural Language Processing*. Algorithmics Press, New York, NY.

McFetridge, Paul, and Nick Cercone. (1990). The Evolution of a Natural Language Interface: Replacing a Parser. In: *Proceedings of Computational Intelligence 90*. Università di Milano, Milan, Italy.

Mellish, Chris. (1988). Implementing Systemic Classification by Unification. *Computational Linguistics*, *14*(1), 40-51.

Moens, Marc, Jo Calder, Ewan Klein, Mike Reape, and Henk Zeevat. (1989). Expressing Generalizations in Unification-Based Grammar Formalisms. In: *Proceedings of the 4th Conference of the European Chapter of the Association for Computational Linguistics*. Manchester, England.

Oehrle, Richard, Emmon Bach, and Deirdre Wheeler. (1988). *Categorial Grammars and Natural Language Structures*. D. Reidel, Dordrecht, Holland.

Pereira, Fernando, and Stuart Shieber. (1987). *Prolog and Natural-Language Analysis*. Centre for the Study of Language and Information, Stanford University, CA.

Pollard, Carl, and Ivan Sag. (1987). *Information-Based Syntax and Semantics, Volume 1: Fundamentals*. Centre for the Study of Language and Information, Stanford University, CA.

Pollard, Carl, and Ivan Sag. (forthcoming). *Information-Based Syntax and Semantics, Volume 2: Topics in Binding and Control*. Centre for the Study of Language and Information, Stanford University, CA.

Popowich, Fred. (1989). *A Tree Unification Grammar-Based Natural Language Processor* (CSS-IS TR 89-08). Simon Fraser University, Burnaby, B.C.

Popowich, Fred. (1989). Tree Unification Grammar. In: *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*. Vancouver, B.C.

Popowich, Fred and Carl Vogel. (1990). *Chart Parsing Head-Driven Phrase Structure Grammar* (CSS-IS TR 90-01, CMPT TR 90-01). Simon Fraser University, Burnaby, B.C.

Proudian, Derek and Carl Pollard. (1985). Parsing Head-Driven Phrase Structure Grammar. In: *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. University of Chicago, Chicago, IL.

Quintus Computer Systems, Inc. (1990). *Quintus Prolog Manuals, Release 3.0 (UNIX)*. Mountain View, CA.

Seiffert, Roland. (1987). *Chart-Parsing of Unification-Based Grammars with ID/LP-Rules* (LILOG-REPORT 22). IBM Deutschland GmbH, LILOG, Stuttgart, FRG.

Shieber, Stuart. (1985). Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In: *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*. University of Chicago, Chicago, IL.

Shieber, Stuart. (1986). *An Introduction to Unification-Based Approaches to Grammar*. The University of Chicago Press, Chicago, IL.

Shieber, Stuart, Hans Uszkoreit, Fernando Pereira, Jane Robinson, and M. Tyson. (1983). The Formalism and Implementation of PATR-II. In B. Grosz and M. Stickel (Eds.), *Research on Interactive Acquisition and Use of Knowledge*. SRI International, Menlo Park, CA.

Thompson, Henry. (1981). Chart Parsing and Rule Schemata in PSG. In: *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*. Stanford University, Stanford, CA.

Touretzky, David. (1986). *The Mathematics of Inheritance Systems*. Morgan Kaufmann, Los Altos, CA.

Touretzky, David, John Horty and Richmond Thomason. (1987). A Clash of Intuitions: The Current State of Non-Monotonic Multiple Inheritance Systems. In: *Proceedings of the 10th International Joint Conference on Artificial Intelligence*. Milano, Italy.

Uszkoreit, Hans. (1987). A Flexible Type-Unification-Based Representation Formalism. In: *Proceedings of the Alvey/SERC Workshop on Natural Language Processing, Unification and Grammar Formalisms*. Centre for Cognitive Science, University of Edinburgh.

Vogel, Carl and Fred Popowich. (1990). Head-Driven Phrase Structure Grammar as an Inheritance Hierarchy. In W. Daelemans and G. Gazdar (Eds.), *Inheritance in Natural Language Processing Workshop Proceedings*. Institute for Language Technology and Artificial Intelligence, Tilburg University, Holland. Also appeared as CSS/LCCR TR 90-03, Centre for Systems Science, Simon Fraser University, Burnaby, B.C.

Table of Contents