CLP based HPSG Parsing

Johannes Matiasek
Austrian Research Institute Artificial Intelligence
Schottengasse 3,
A-1010 Vienna, Austria
john@ai.univie.ac.at

Abstract

We describe a system for principle based parsing of HPSG employing constraint logic programming techniques. Typed features structures are implemented as constraints on PROLOG variables and are instantiated in a lazy fashion. Grammar principles as well as relational constraints are stated in a declarative way by means of conditional constraints on feature structures. The procedural interpretation given to these conditional constraints together with the data driven delay mechanism implemented yields efficient parsing behavior.

1 Introduction

In this paper we describe a principle based parser for HPSG. Recent developments in theoretical and computational linguistics have moved from grammars defined over tree structures to principle based ones. For these grammars the general, well understood parsing algorithms developed for use with context free phrase structure grammar are of little use. On the other hand, generate and test strategies using the grammar principles to eliminate the ill-formed structures suffer from the huge search space of possible structures. Therefore all the specialized principle based parsers written for GB (Berwick 1991) employ a covering phrase structure grammar to restrict the search space. Although this strategy proved to be viable, methods without the need to use theory-external devices would be preferable.

HPSG is better to handle in this respect, since the system of sorted feature structures itself restricts the space of possible structures. The principles of grammar, formulated as constraints over these feature structures apply also to structures not yet fully instantiated, and therefore may also be used to guide the instantiation process. Nevertheless, to obtain an efficient system behavior and to avoid infinite recursion it is inevitable to provide some means to control the way in which instantiation and principle application interact procedurally. Since we wanted to preserve as much as possible from the declarative spirit of HPSG without running into efficiency problems, in the implementation described here this procedural guidance is achieved by a schema allowing for lazy principle application.

2 The HPSG formalism

HPSG (Pollard and Sag 1987, Pollard and Sag in press) differs from other, more traditional grammar formalisms in various important aspects. Unlike *rule-based* formalisms like LFG (Kaplan and Bresnan 1982) or GPSG (Gazdar et al. 1985) it doesn't employ phrase structure rules to generate the well-formed structures of language but rather uses universal and language specific *principles* to eliminate ill-formed structures. This principle based approach to language originated with GB-Theory (Chomsky 1981). However, HPSG departs from GB in not using any derivational notions such as movement and in employing feature structures instead of phrase structure trees as the primary device for linguistic representation. Immediate dominance and linear precedence conditions (in almost all other theories handled by phrase structure rules) are expressed in HPSG in the same way as all the other grammatical principles, i.e., in the form of constraints on feature structures.

Consequently, there is very little left to do for traditional phrase structure grammar based parsers/generators. Instead, parsing and generation can be viewed as constraint solving. An utterance to be parsed corresponds to a partially specified feature structure (only the phonetic value being specified), the parsing process amounts to checking the satisfiability of this partial structure in conjunction with the constraints originating from the principles of grammar and from the lexical entries.

Since HPSG has been introduced in Pollard and Sag 1987, some changes in the theory as well as in the requirements on the formal basis of it have been made. I will shortly sketch the main aspects of the formalism underlying HPSG in the version of Pollard and Sag in press (more formally described in Carpenter et al. 1991) which has been the basis of the implementation described here.

Linguistic objects in HPSG are modeled by *typed* (or *sorted* in the terminology of Smolka 1988) feature structures. Every node of a feature structure is labelled by a type symbol. The types themselves are partially ordered forming a lattice. All features appearing at a node of a feature structure must be *appropriate* for the type of that node. Furthermore, the values that a feature may take are type restricted. If a feature is appropriate for some type, it is also appropriate for all of its subtypes. Feature structures are *well-typed* if every feature that appears is appropriate and takes an appropriate value. Well-typedness can be ensured by *type inference* and *well-typed unification* can be defined (see Carpenter et al. 1991).

The principles of grammar impose further restrictions on linguistic objects. To be well-formed, a feature structure has to satisfy all principles of grammar (applied to all nodes of it), i.e, every node meeting the preconditions associated with a principle has to satisfy the constraints imposed by it. Often these preconditions are specified by a certain type, so Carpenter et al. 1991 require these constraints simply to be associated with types. But the way grammatical principles are stated in Pollard and Sag in press suggests a formal specification by means of conditional feature structures as this has been the general practice in Pollard and Sag 1987. The implementation described below allows for the latter case.

The constraints imposed by the principles range from simple ones, e.g., requiring only the identity of two substructures as the *Head Feature Principle* to rather complex relational

constraints[1] as in the *Subcategorization Principle* which constrains the SUBCAT-value of the head daughter (of a headed phrase) to be the result of appending the list of the SYNSEM-values of the complement daughters to the SUBCAT-value of the phrase.

3 About CLP

Constraint logic programming has been developed during the past few years to circumvent some difficulties arising from from the fact that unification in logic programming languages such as PROLOG is defined syntactically over Herbrand-terms. This prevents for example syntactically distinct but (in the sense of an underlying theory) semantically equivalent terms to be unified. Various extensions to syntactic unification have been proposed, one of them being so-called *semantic unification*. The basis for the work described here is an extended PROLOG implementation[2] employing *attributed variables* as mechanism for semantic unification.

Attributed variables are an additional data-type allowing logical variables to be directly qualified by arbitrary user-defined attributes. This way constraints on variables can be easily specified by attaching appropriate attributes. Unification of two attributed variables or of an attributed variable with a term is handled in a special way. For either case the user has to supply a predicate, which explicitly specifies how the attributes interact and how they have to be interpreted with respect to unification. Syntactic unification succeeds only if this combination—or verification—of the attributes involved is successful. These constraint solving clauses may be specified by conditional rewrite rules for attributes[3], providing a means to integrate constraint solvers into PROLOG in a declarative way.

A most natural way to integrate typed feature structures into PROLOG is to view these structures as constraints and implement feature structure unification as attribute rewriting. A further advantage with this approach is the possiblity of implementing delay mechanims without having to use metainterpreters and thus gaining efficiency. Therefore this method has been adopted for implementing the system described here.

4 A principle based HPSG parser

Two goals have been pursued when developing the system described here:

- The syntax for specifying the grammar should be as close to standard HPSG notation as possible.
- The amount of extra-grammatical specification required to make effective use of the grammar in a parser should be kept at a minimum. While there may be possibilities

[1] Such recursively defined relational constraints may lead to problems with infinite recursion if no hints on how to process them are provided.

[2] DMCAI CLP 2.1(Holzbaur 1992) is a enhanced version of SICStus Prolog providing extensible unification.

[3] in the enhanced version of DMCAI CLP, see Pfahringer and Matiasek 1992

to process a principle based grammar in a rule based fashion by transforming the principles in a particular way or using extra devices such as covering phrase structure grammars, the approach taken here employs the principles directly.

Despite these demands the system is required to operate with reasonable efficiency. How these goals have been achieved will be demonstrated below by describing the definition language for the type scheme of feature structures and how unification of feature structures is implemented as constraint solving. A schema of lazy principle application is given which allows to employ the principles directly in the parsing process without running the risk of getting trapped in infinite loops or loose efficiency due to too early a commitment to a particular choice.

4.1 The type scheme

The signature specifying the type scheme of sorted feature structures has to be defined before any feature structures can be used. The lattice of types is defined via the operator `..>/2`, part of which is shown below.

```
object ..> sign.
           sign ..> word.
           sign ..> phrase.
                    phrase ..> headed_phrase.
```

Appropriateness conditions and sortal restrictions on feature values are defined via the operator `==>/2`. For each type introducing a feature or further restricting the value of an inherited feature this definition has to be made. For non-atomic types that only inherit the appropriateness conditions from their supertypes an empty list has to be given.

```
sign          ==> [phon:   phon,
                   synsem: synsem].
word          ==> [].                      % inherit only
phrase        ==> [dtrs:   const_struc].   % add feature dtrs
headed_phrase ==> [dtrs:   headed_struc].  % value restrict dtrs
```

These two operators are sufficient for defining the signature. The predicates necessary to compute the transitive closure of `..>`, the set of supertypes of a given type etc. are defined in the usual way independent of the particular grammar[4].

4.2 Feature structures as constraints on variables

The implementation of typed feature structures in our system makes use of the CLP facilities provided by the enhanced PROLOG system described above. Feature structures are implemented by the attribute `fs(Type,Dag)` where `Dag` is either a well-typed list of

[4]For efficiency reasons, these type hierarchy traversing predicates are compiled into PROLOG facts. Together with clause indexing this enables processing of the type hierarchy in constant time.

feature-value pairs, the values being constrained variables, or the atom `uninstantiated`. Instantiation of feature structures can be done in a *lazy* fashion which saves considerable amounts of space and time in cases where unification fails due to incompatible types.

Unification of two feature structures is performed when PROLOG tries to unify two attributed variables constrained by these feature structures, and thus defined by means of rewrite rules for attributes. A preliminary version of these rules[5] not yet accounting for delayed principle application is:

```
fs(T1,uninstantiated),fs(T2,uninstantiated) => fs(T3,uninstantiated):-
   glb(T1,T2,T3).
fs(T1,uninstantiated),fs(T2,Dag2) => fs(T3,Dag3) :-
   glb(T1,T2,T3),
   unify_dags(T3,[],Dag2,Dag3).
fs(T1,Dag1),fs(T2,Dag2) => fs(T3,Dag3) :-
   glb(T1,T2,T3),
   unify_dags(T3,Dag1,Dag2,Dag3).
```

`unify_dags/4` merges the two input lists unifying the values of features occuring in both lists. Since these values are constrained variables, recursion is handled automagically by the rewrite rules above. The appropriateness conditions for the resulting type are checked on the fly to assure well-typedness of the resulting dag[6].

Given the possibility of having uninstantiated feature structures there must also be means to explicitly instantiate them. Instantiation is triggered by referring to a substructure within an uninstantiated feature structure. For this purpose appropriate operators have been defined. For example, `X::synsem:loc:cat:head===noun` enforces a subtyping of the syntactic head of `X` to type `noun`, in a similar way structure sharing of substructures can be enforced by using simple PROLOG variables as coreference tags in path equations. Since type inference applies during instantiation only well-typed feature structures are produced. These path expressions are used to specify lexical entries and to state the grammatical principles, to which we will turn below.

4.3 Principles of Grammar

Grammatical principles in our implementation of HPSG are formulated as conditional constraints and apply to all nodes of a feature structure. For defining these conditional constraints, the operator `===>/2` separating antecedent and consequent has been defined. This operator is expanded at read time in the way described below. As an example we

[5] using the conditional rewrite rule syntax described in Pfahringer and Matiasek 1992.

[6] this corresponds to the subfunction of *TypInf* of Carpenter et al. 1991 responsible for appropriately restricting the types of the feature values.

show how the *Head Feature Principle* is represented in our system:

```
head_feature_principle(X) :-
      X::=headed_phrase
  ===>
      X::synsem:loc:cat:head===H,
      X::dtrs:head_dtr:synsem:loc:cat:head===H.
```

A procedural interpretation of these conditional constraints is, that

- if a node satisfies the antecedent of the conditional then the consequent has to be enforced,
- if a node and the antecedent of the conditional fail to unify, then the consequent simply does not apply.

The case not made explicit above is the one which prevents conditional feature structures to be processed straightforwardely, i.e., the case when a node is compatible with the antecedent of a conditional but is not subsumed by it. In precisely that case principle application has to be delayed.

Delayed Application of Grammatical Principles

The idea behind implementing this kind of principle application blocking is to annotate the variables that are "responsible" for the unabiliy to decide on the antecedent of the principle that has to be applied[7]. We can now introduce `fs(Type,Dag,Goals)` as the final representation for feature structures in our system replacing `fs/2`. `Goals` is a list of goals which have to be invoked in case either `Type` or `Dag` get restricted due to unification with another feature structure.

The rewrite rules have to be augmented accordingly by calls to `start_goals/1` which calls all goals in the list, e.g.

```
fs(T1,Dag1,Goals1),fs(T2,Dag2,Goals2) => fs(T3,Dag3,[]) :-
   ...
   start_goals(Goals1), start_goals(Goals2).
```

Note the empty `Goals` in the resulting attribute description. Principles that cannot be applied due to insufficient specification of the resulting feature structure reinsert themselves into that list.

This insertion of delayed goals is triggered by the special treatment of the conditions in the antecedent of the principles, which are restricted to be path equations. These are translated appropriately at read time via `term_expansion/2`, embedding them into a predicate implementing the following behavior: If during descending the path of the

[7] This is a generalization of the `block` declaration of SICStus Prolog which only allows to check, whether an argument is instantiated or not.

path equation a substructure is uninstantiated or the type found at the path target is a supertype of the type specified in the condition, the goal is delayed by attaching it to the goals list of that variable.

Implementing application of grammatical principles that way no additional procedural devices have to be introduced to prevent too early a commitment to a particular choice or infinite loops due to blind instantiation. The declarative specification of the principles as they are suffices.

Recursive Relations

Some principles of HPSG involve relational constraints (such as *append*) expressible only by recursive definitions[8]. The conditional syntax with application blocking used in defining the grammatical principles is useful also to define these recursive relations. The antecedent part of these definitions is used for defining the sortal constraints on the arguments and for specifying the blocking conditions for the relation, as can be seen in the following example.

```
fs_append(X,Y,Z) :-
       X::=list,Y::=list,Z::=list
  ===> fs_empty_append(X,Y,Z),fs_nonempty_append(X,Y,Z).
fs_empty_append(X,Y,Z) :-
       X::=elist,Y::=list,Z::=list
  ===> Y = Z.
fs_nonempty_append(X,Y,Z) :-
       X::=nelist,Y::=list,Z::=list
  ===> X::first===First,Z::first===First,X::rest===XRest,Z::rest===ZRest,
       fs_append(XRest,Y,ZRest).
```

The advantages of this conditional definition are twofold. First, the disjunctive relation *append* can now be written as conjunctively applying the two specialized cases. Second, infinite loops due to uninstantiated variables can never occur.

4.4 Parsing as constraint interaction

In order to effectively produce a parse of an utterance the system must not be too lazy in instantiating structures and applying principles. One source of fully instantiated structures is the lexicon. The other possibility to enforce instantiation of structures is to use principles as generators. This can be done by keeping the preconditions of the principle to an absolute minimum (e.g. specifying only the type of the structures to which it applies). That way no delay in the application of the principle occurs and the constraints enforced by it are instantiated immediately.

Thus the conditional principle syntax and lazy application scheme makes it possible to specify declaratively within the grammar which degree of instantiation is required before a principle applies—making it either act as a filter for or a generator of structures.

[8]Lists are represented using the usual first/rest notation as in Shieber 1986.

5 Comparison with other approaches

Recently some other systems especially designed for HPSG parsing have been developed. Comparing our approach to the implementation of HPSG described in Balari et al. 1990, our approach benefits from what could be called indexing. Every variable is related to exactly those constraints that are relevant for this variable and to no other constraint whatsoever. So rewriting can be done just at the right point in time, namely when the variable is augmented by an additional constraint or instantiated during unification, and rewriting need only consider the relevant, *small* subset of all constraints. CLG(2) just augments predicates with two arguments for the list of constraints at clause entry and exit, and "applies a rewriting process to the whole list from time to time".

The system most directly comparable with ours is the one by Franz 1990. It implements the whole apparatus of HPSG including well-typedness constraints and parsing is performed via satisfiablity checking. The drawback of the system is its rather slow performance. We had no possibility of benchmarking both systems on the same platform but rough estimates indicate that our system performs faster by a factor of at least 100. The main reason for this efficiency gain in our system is the reduction of the search space by reformulating disjunctive constraints as simultaneously applying conditional constraints being applied lazily (as the `fs_append` example above shows).

6 Conclusion and Further Work

We have described a system for principle based parsing of HPSG employing CLP techniques. Typed features structures are implemented as constraints on PROLOG variables and are instantiated in a lazy fashion. Grammar principles as well as relational constraints are stated in a declarative way by means of conditional constraints on feature structures. The procedural interpretation given to these conditional constraints together with the data driven delay mechanism implemented yields efficient parsing behavior.

The system is fully implemented and has been sucessfully tested with an HPSG grammar covering a substantial fragment of German (on details of the grammar see Heinz and Matiasek to appear).

There remain of course possiblities to improve performance further. Handling of disjunctions is a crucial point in efficiently processing natural language and has therefore attracted a lot of attention (Trost 1992). Disjunctions originating from grammar principles often can be guised as conjunctively applying conditional constraints, enumeration of the remaining relatively few choices via backtracking leads to acceptable runtimes. Handling disjunctions stemming from the lexicon the same way usually leads to combinatorial explosion of search space and consequently also of runtime as serious lexicons tend to contain an abundance of such disjunctions. Simple disjunctions, i.e., disjunctions of atomic values can be handled easily in the CLP framework by introducing domain attributes as additional constraints and providing the appropriate attribute rewrite rules. Unfortunately, such simple disjunctions are rather rare. Currently we are investigating a schema of repre-

senting more general disjunctions locally and postponing decision as long as there is some other deterministic computation to be done. This strategy has the advantage of either reducing the choices remaining or at least reducing the amount of computation involved in backtracking. The existing delay mechanism will be used to implement this behavior.

Acknowledgements

This research has been sponsored by the Austrian *Fonds zur Förderung der wissenschaftlichen Forschung*, Grant No. P7986-PHY. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian *Ministry of Science and Research*. I would like to thank Bernhard Pfahringer for fruitful discussions and comments, Christian Holzbaur for providing DMCAI CLP, Harald Trost for helpful comments and for pushing me to write this paper and Prof. R. Trappl for his continuing support.

References

Balari, S., G. B. Varile, L. Damas, and N. Moreira. 1990. CLG(n): Constraint Logic Grammars. In *Proceedings of the 13th COLING*, 7–12. Helsinki.

Berwick, R. 1991. Principles of Principle-Based Parsing. In *Principle-Based Parsing*, ed. R. Berwick, S. Abney, and C. Tenny. Dordrecht: Kluwer.

Carpenter, B., C. Pollard, and A. Franz. 1991. The Specification and Implementation of Constraint-Based Unification Grammars. In *Proceedings of the Second International Workshop on Parsing Technology*, 143–153. Cancun, Mexico.

Chomsky, N. 1981. *Lectures on Government and Binding*. Dordrecht: Foris.

Franz, A. 1990. A Parser for HPSG. Technical Report CMU-LCL-90-3, Carnegie Mellon University, Pittsburg, PA.

Gazdar, G., G. P. E. Klein, and I.Sag. 1985. *Generalized Phrase Structure Grammar*. Cambridge, Mass.: Harvard University Press.

Heinz, W., and J. Matiasek. to appear. Argument Structure and Case Assignment in German. In *HPSG for German*, ed. J. Nerbonne, K. Netter, and C. Pollard. Stanford: CSLI Publications.

Holzbaur, C. 1992. DMCAI CLP Reference Manual. Technical Report TR-92-24, Austrian Research Institute for Artificial Intelligence, Vienna.

Kaplan, R., and J. Bresnan. 1982. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In *The Mental Representation of Grammatical Relations*, ed. J. Bresnan. Cambridge, Mass.: MIT Press.

Pfahringer, B., and J. Matiasek. 1992. A CLP Schema to Integrate Specialized Solvers and its Application to Natural Language Processing. Technical Report TR-92-37, Austrian Research Institute for Artificial Intelligence, Vienna.

Pollard, C., and I. Sag. 1987. *Information-Based Syntax and Semantics, Vol. 1: Fundamentals.* CSLI Lecture Notes 13. Stanford, CA: CSLI.

Pollard, C., and I. Sag. in press. *Head-Driven Phrase Structure Grammar.* To be published by University of Chicago Press and CSLI Publications.

Shieber, S. 1986. *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes 4. Stanford, CA: CSLI.

Smolka, G. 1988. A Feature Logic with Subsorts. Technical Report LILOG-Report 33, IBM-Germany, Stuttgart.

Trost, H. (ed.). 1992. *Coping with Linguistic Ambiguity in Typed Feature Formalisms.* Vienna. Proceedings of a workshop held at ECAI'92.