Implementing HPSG grammars

Part I: Background

Detmar Meurers
OSU, LING795K, Spring 2002

Why implement an HPSG theory?

The implementation of HPSG theories can be very valuable in terms of

a) providing feedback for a rigid and complete formalization of a linguistic theory, as well as

b) stimulating system development to enhance the link between theory and implementation and to improve performance.

But implementations can only be valuable in this way if some basic conditions are met ...

Conditions on useful grammar implementations

For implementations to be useful

- the implementation needs to be thoroughly documented. In particular also: all differences between the linguistic theory and the implementation should be documented and explained.
- the system used should support a clear, tractable, and formally meaningful way of implementing a grammar close to the linguistic theory.

HPSG grammars

1. From a linguistic perspective
2. From a formal perspective
3. From an implementation perspetive

HPSG grammars from a linguistic perspective

From a linguistic perspective, an HPSG grammar consists of

a) a lexicon
   licensing basic words
b) lexical rules
   licensing derived words
c) immediate dominance (ID) schemata
   licensing constituent structure
d) linear precedence (LP) statements
   constraining word order
e) a set of grammatical principles
   expressing generalizations about linguistic objects

HPSG grammars from a formal perspective (I)

From a formal perspective, an HPSG grammar consists of:

- The **signature** as declaration of the domain:
  - type hierarchy (which kind of objects exist)
  - appropriateness conditions (which objects have which properties)
- The **theory** constraining the domain
  - A theory is a set of description language statements, the constraints.
  - A linguistic object is admissible with respect to a theory iff it satisfies each of the descriptions in the theory and so does each of its substructures.

HPSG grammars from a formal perspective (II)

The theory (in a formal sense) is a set of description language statements, sometimes called "the constraints", which single out the grammatical objects from the ungrammatical ones.

- The description language statements consist of: type assignment, path equality, conjunction, disjunction, negation.
- Most of the theory—the lexicon, ID schemata, and principles—is already expressed using such statements.
- Other components can also be formalized using the same logical basis:
  - LP statements (Richter and Sailer 1995)
  - Lexical rules (Meurers 1995, 2001)

HPSG grammars from an implementation perspective

Desideratum:

The system should support a clear, tractable, and formally meaningful way of implementing close to the linguistic theory. Recoding a linguistic theory in terms of some unrelated or lower-level computer language makes it difficult to obtain meaningful feedback from the implementation for linguistics.

What can be expressed in the different systems?

a) constraints describing the domain directly
b) relational backbone relating elements in the domain
   i. relations
   ii. phrase structure

Computational systems built on constraints describing the domain directly

TFS: Typed Feature structure System
(Emele and Zajac 1990)

- A TFS theory is a set of implicational statements with type antecedents.
- Any description can be entered as a query and the system returns a description subsumed by the query, such that the descriptions satisfies each theory constraint and so does each of its parts.

TFS evaluation

Pro:

- The organization of constraints is the same as in the HPSG architecture.

Cons:

- Only implicational statements with type antecedent are allowed. No general treatment of negation.
- Very severe control problems: efficiency, termination

Computational systems using a relational backbone

CUF: Comprehensive Unification Formalism
(Dörre and Eisele 1991; Dörre and Dorna 1993)

- A theory is expressed using definite clauses as a relational extension of the description language.
- An HPSG grammar is implemented by rewriting it as a logic program. The recursive constraints are encoded on a different level (relations) than the linguistic data structure constraints (arguments to the relations).
- A query is a call to one of the relations and the system returns the instantiation of the arguments required by the relation.

Relational backbone – an example

```
well_formed_sign(Sign) :-
    well_formed_word(Sign);
    well_formed_phrase(Sign).

well_formed_word((phon:[john],synsem:loc:cat:(head:noun
                                              subcat:[]))).
...

well_formed_phrase(Phrase) :-
    id_schema(Phrase,HeadDtr,OtherDtrs),
    well_formed_sign(HeadDtr),
    well_formed_signs(OtherDtrs),
    ....
```

Relational backbone – example continued

```
id_schema((Mother, synsem:loc:cat:subcat: [Rest]
                   dtrs: (headed_structure,
                          head_dtr: HeadDtr,
                          comp_dtrs: CompDtrs)),
          (HeadDtr, word, synsem:loc:cat:subcat: [Rest|Comps]),
          CompDtrs) :-
   head_feature_principle(Mother,HeadDtr),
   ....

head_feature_principle(synsem:loc:cat:head:H,
                       synsem:loc:cat:head:H).
```


Relational backbone – evaluation

Pros:

- The grammar writer determines the order of execution of goals by the way in which (s)he encodes the grammar, i.e., the definite clause program.
- A definite clause encoding allows the efficient processing techniques of logic programming to be used, e.g., clause indexing, earley deduction, goal freezing.

Con:

- Since the organization and expression of constraints is completely different from an HPSG theory, a grammar with a relational backbone is only related to the original linguistic theory on an intuitive level.

Computational systems with phrase structure backbone

ALE (Carpenter 1992; Carpenter and Penn 1994), **TDL** (Krieger 1995), **LKB** (Copestake 1993), **Troll** (Gerdemann and Götz 1996)

A program is organized around a special kind of relation, phrase structure, sometimes permitting definite clause attachments.

Phrase structure is a relation

- demanding a fixed number of daughters (at runtime)
- having a designated phonology argument satisfying the condition: $\text{phon}(\text{Mother}) = \text{phon}(\text{Dtr}_1) \oplus \ldots \oplus \text{phon}(\text{Dtr}_n)$

$\Rightarrow$ very efficient algorithms for parsing due to indexing on input string

Phrase structure backbone – example

```
head_complement_schema rule
  (Mother, synsem:loc:cat:subcat: [Rest])
===>
  cat> (HeadDtr, word, synsem:loc:cat:subcat: [Rest|Comps]),
  cats> Comps,
  goal> (head_feature_principle(Mother,HeadDtr),
         ...).
```

Phrase structure backbone – evaluation

Pros:

- most important recursive structure singled out
- very efficient algorithms for parsing are available since indexing on the phonology can be used

Cons:

- As with the general relational encoding, the organization and expression of constraints is different from the linguistic theory.
- The additional restrictions on phrase structure (compared to general relations) can require the grammar writer to encode each constraint specifying constituent structure in several phrase structure rules.

Combining multiple worlds

A relational backbone is advantageous for processing reasons, but a relationally encoded grammar recodes instead of uses the elaborate typed feature structure domain defined by the signature.

Idea: Use of both relational background and implicational constraints allows for more modular, underspecified encoding of grammars.

- ConTroll (Götz and Meurers 1995, 1997): Combine a relational backbone with implicational constraints with complex antecedents.
- Trale (Milca Project): Combine a phrase structure backbone with implicational constraints with complex antecedents.

Example

Assume the relation `well_formed_phrase` defined to hold of all and only the grammatical phrases (with `r/1` being the relevant relations, cf. p. 12).

```
well_formed_phrase(Phrase) :- r(Phrase).
```

How can one add the head feature principle to this?

```
head_feature_principle(synsem:loc:cat:head:H,
                       synsem:loc:cat:head:H).
```

- has to be called in the relevant places
- requires encoding parts of the ontology in the program structure

Example (cont)

To call the relation `head`$_f$`eature`$_p$`rinciple` at the appropriate places, the relational encoding has to account for the ontological difference between phrases (those with DTRS of type *headed_phrase* vs. those of other types, i.e. *coordinate_struc*):

```
well_formed_phrase((X,dtrs: (headed_struc,head_dtr:HeadDtr))) :-
    head_feature_principle(X,HeadDtr),
    r(X).
well_formed_phrase((X,dtrs: coordinate_struc)) :-
    r(X).
```

Evaluation of example

- **Problem:** Such an encoding is neither compact nor efficient, since (without special indexing) its execution always leaves a choice-point behind.
- **Why?** In the relational encoding it is not possible to only refer to those phrases having DTRS of type *headed_phrase*, since every subcase has to be licensed.
- **Solution:** Universal principles. They are constraint based in the intuitive sense: Every structure that is not explicitly excludes is well-formed.

```
(phrase, dtrs:headed_struc)  *>
                  (synsem:loc:cat:head:H,
                   dtrs:head_dtr:synsem:loc:cat:head:H).
```


References

Carpenter, Bob (1992). *ALE – The Attribute Logic Engine, User's Guide*. Laboratory for Computational Linguistics Report CMU-LCL-92-1, Laboratory for Computational Linguistics, Carnegie Mellon University.

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

Copestake, Ann (1993). *The Compleat LKB*. Technical report 316, University of Cambridge Computer Laboratory.

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

Dörre, Jochen and Andreas Eisele (1991). *A Comprehensive Unification-Based grammar Formalism*. Tech. Rep. Deliverable R3.1.B, Dyana.

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

Gerdemann, Dale and Thilo Götz (1996). Troll manual. Ms., Universität Tübingen.

Götz, Thilo and Walt Detmar Meurers (1995). Compiling HPSG Type Constraints into Definite Clause Programs. In *Proceedings of the 33nd Annual Meeting of the Association for Computational Linguistics (ACL 95)*. Cambridge, MA: MIT, pp. 85–91. http://ling.osu.edu/~dm/papers/acl95.html.

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

Krieger, Hans-Ulrich (1995). *TDL*—A Type Description Language for Constraint-Based Grammars. Foundations, Implementation, and Applications. Ph.D. thesis, Universität des Saarlandes, Department of Computer Science.

Meurers, Walt Detmar (1995). Towards a Semantics for Lexical Rules as used in HPSG. In *Proceedings of the ACQUILEX II Workshop on the Formalisation and Use of Lexical Rules*. Cambridge, UK, pp. 1–20. Also: First Conference on Formal Grammar, Barcelona, 1995. http://ling.osu.edu/~dm/papers/dlrs.html.

Meurers, Walt Detmar (2001). On expressing lexical generalizations in HPSG. *Nordic Journal of Linguistics* 24(2), 161–217. Special issue on "The Lexicon in Linguistic Theory". http://ling.osu.edu/~dm/papers/lexical-_generalizations.html.

Richter, Frank and Manfred Sailer (1995). Remarks on Linearization: Reflections on the Treatment of LP-Rules in HPSG in a Typed Feature Logic. Master's thesis, Seminar für Sprachwissenschaft, Universität Tübingen. http://www.sfs.uni-_tuebingen.de/~fr/cards/thesis.html.