APPROACHES TO SURFACE REALIZATION WITH HPSG

Graham Wilcock
Centre for Computational Linguistics
University of Manchester Institute of Science and Technology
PO Box 88, Manchester M60 1QD, United Kingdom
graham@ccl.umist.ac.uk

Abstract

HPSG is widely used in theoretical and computational linguistics, but rarely in natural language generation. The paper describes some approaches to surface realization in which HPSG can be used. The implementation of all the approaches combines generation algorithms in Prolog and HPSG grammars in ProFIT. It is natural to combine a head-driven HPSG grammar with a head-driven generation algorithm. We show how a simple head-driven generator can easily be adapted for use with HPSG. This works well with simplified semantics, but if we implement the full HPSG textbook semantics this approach does not work. In a second approach to head-driven generation, we implement some recent revisions of HPSG, and show that head-driven generation with HPSG is in fact possible. We then switch to non-head-driven approaches. We show how a bag generation algorithm, developed for use with categorial grammar and indexed QLF, can be used with HPSG and Minimal Recursion Semantics. We describe an approach to incremental generation with HPSG, noting a difficulty for highly incremental generation with HPSG and proposing a solution. Finally we briefly mention a few other plausible approaches.

1 Introduction

In work on natural language generation, the most influential linguistic framework has probably been Systemic Functional Grammar (SFG). However, in other areas of computational linguistics the most widely used grammatical framework appears to be Head-driven Phrase Structure Grammar (HPSG). Why is it, then, that using HPSG for generation has been almost as unpopular as using SFG for parsing?

Without making any claim that HPSG is better than SFG for generation, we will review some plausible approaches to surface realization with HPSG. We will show that there are indeed some fundamental difficulties in using HPSG for generation, but also that there are some solutions to these difficulties.

The first approach to mention is the radical one of converting HPSG into something else before generation, such as Tree Adjoining Grammar (Kasper et al., 1995). Though this seems to support the view that HPSG is unsuitable for generation, it is in fact a valuable contribution to work on compiling HPSG grammars for efficient processing, whether for parsing (Torisawa and Tsujii, 1996) or for generation.

However, we will not be concerned with efficiency, but with more basic problems in the relations between HPSG and generation algorithms. The question is, can existing algorithms be used with HPSG grammars at all? For clarity, we use the simplest versions of the algorithms, which were originally developed for use with definite clause (DCG) grammars and categorial grammar. For uniformity, the algorithms are implemented in Prolog and the grammars are implemented in ProFIT (Erbach, 1995).

1.1 Generation from what?

A basic problem in using HPSG for generation is the question "Generation from what?" Various different semantic representations have been used with HPSG grammars, partly due to differences in semantic theories and partly due to differences in the requirements of particular applications, such as database interfaces, machine translation, or interactive dialogues.

The semantic theory which has been particularly associated with HPSG is Situation Semantics. As feature structures became central to linguistic description, and unification became central to linguistic processing, the standard (Pollard and Sag, 1994) semantic representation in HPSG has been a feature structure version of Situation Semantics, and semantic composition has been implemented by unification of the semantic features of the components. So one answer to what generation should start from is to generate from Situation Semantics.

The distinctive characteristic of HPSG is its emphasis on a head-driven organization of grammar. So it is natural to try using a head-driven generation algorithm, and this is compatible with the feature structure version of Situation Semantics. Head-driven approaches to generation with HPSG are described in detail by Wilcock and Matsumoto (1998). That work is summarised here, in Section 2 where a simple approach runs into fundamental difficulties and in Section 3 where a more sophisticated approach offers a solution.

In machine translation, "head-switching" between languages (when the syntactic or semantic head of a source language structure does not naturally transfer to the head of a translationally equivalent structure in the target language) means that a strongly head-driven approach to semantics is undesirable. The problem of logical form equivalence is also crucial for generation in machine translation. A flat, list-based semantic representation is therefore more suitable. Minimal Recursion Semantics (Copestake et al., 1997) has been developed specifically to provide such a flat representation for HPSG.

For generation from flat lists, we need non-head-driven approaches. In Section 4 we show how an existing bag generation algorithm, developed for use with categorial grammar and indexed logical form, can also be used with HPSG and Minimal Recursion Semantics implemented in ProFIT.

In interactive dialogues, generation needs to start from an incomplete bag of semantic terms, and continue incrementally as more terms are added. In Section 5 we describe an approach to incremental generation with HPSG. In contrast to categorial grammar, HPSG has some fundamental difficulties with highly incremental generation. However, we suggest a chart-based solution to the problem.

In conclusion, some other approaches are briefly mentioned in Section 6. However, before discussing the different approaches to generation, we introduce the ProFIT system used for the implementations.

1.2 HPSG in ProFIT

ProFIT (Erbach, 1995) extends Prolog with typed feature structures. A type hierarchy declaration defines the subtypes and appropriate features of every type. Typed feature terms, which can be mixed with ordinary terms in Prolog procedures, are compiled into normal terms by ProFIT before the procedures are passed to the normal Prolog compiler. An idea of how HPSG can be implemented in ProFIT is shown in Figure 1, from (Wilcock and Matsumoto, 1998), where further details are explained. We note here only that the Semantics Principle 'SemP' is defined by a template (:=) which says that the CONTENT of the mother is the same as the CONTENT of the head daughter, and this principle is imposed on all head-nexus-phrases (non-adjunct phrases) by invoking the template by @'SemP' within the template for hd_nexus_ph.

```
'HFP'  := synsem!loc!cat!head!HF &
   hd_dtr!synsem!loc!cat!head!HF.
'SemP' := synsem!loc!cont!Cont &
   hd_dtr!synsem!loc!cont!Cont.

hd_ph := <hd_ph & @'HFP' &
    synsem!loc!cat!val!comps![].
hd_nexus_ph := <hd_nexus_ph & @hd_ph &
    @'SemP'.
hd_subj_ph := <hd_subj_ph & @hd_nexus_ph &
    @'VALP'(spr) & @'VALP'(comps) &
    synsem!loc!cat!val!subj![].
hd_comp_ph := <hd_comp_ph & @hd_nexus_ph &
    @'VALP'(subj) & @'VALP'(spr).
```

```
@hd_subj_ph & phon!P0-PN &
    hd_dtr!(Head &
            synsem!loc!cat!val!subj![S]) &
    subj_dtr!(Subj & synsem!S)
---> [Head & <phrase & phon!P1-PN,
      Subj & <phrase & phon!P0-P1].

@hd_comp_ph & phon!P0-PN &
    hd_dtr!(Head &
            synsem!loc!cat!val!comps![C]) &
    comp_dtrs![Comp & synsem!C]
---> [Head & <word & phon!P0-P1,
      Comp & <phrase & phon!P1-PN].
```

Figure 1: Principles, Phrase Types, Schemata

2 Head-Driven Generation (I)

Head-driven generation algorithms (van Noord, 1990) are based on systematic sharing of logical form between the mother and one of the daughters, the semantic head daughter, in grammar rules. Given a suitable grammar with such systematic sharing, these algorithms are very efficient, especially when implemented with a chart (Haruno et al., 1996; Wilcock and Matsumoto, 1996). The question is, are HPSG grammars suitable in this sense?

This question is addressed by Wilcock and Matsumoto (1998), who point out that semantic head-driven generation with HPSG should be easy to implement because semantic heads are very clearly defined in HPSG (in head-adjunct phrases, the adjunct daughter is the semantic head; in other headed phrases, the syntactic head daughter is the semantic head) and in both cases, the HPSG Semantics Principle requires the semantic content of the semantic head to be identical to the semantic content of the mother. Therefore, taking this semantic content to be the logical form, HPSG appears to be extremely suitable for semantic head-driven generation. The Semantics Principle means that, apart from coordinate structures, all grammar rules must include the sharing of logical form required for head-driven generation.

2.1 BUG1 and HPSG

The simplest version of a head-driven generation algorithm was specified by van Noord (1990) as the BUG1 generator in Prolog. Figure 2, from (Wilcock and Matsumoto, 1998), shows how BUG1 can easily be used with a ProFIT HPSG grammar of the type illustrated in Figure 1.

```
hf(HF) := synsem!loc!cat!head!HF.
lf(LF) := synsem!loc!cont!LF.

% Head Feature Principle replaces link.
predict_word( @lf(LF) & @hf(HF), Word ) :-
  lex( Word & @lf(LF) & @hf(HF) ).
predict_rule(Head,Mother,Others,@hf(HF)) :-
  ( Mother & @hf(HF) ---> [Head|Others] ).

generate(LF, Sign, String) :-
  bug1( Sign & phon!String-[] & @lf(LF) ).
```

```
/* BUG1 generator: van Noord 1990 */
bug1(Node) :-
  predict_word(Node, Small),
  connect(Small, Node).
connect(Node, Node).
connect(Small, Big) :-
  predict_rule(Small,Middle,Others,Big),
  gen_ds(Others),
  connect(Middle, Big).
gen_ds([]).
gen_ds([Node|Nodes]) :-
  bug1(Node),
  gen_ds(Nodes).
```

Figure 2: ProFIT/HPSG Interface for BUG1

2.2 Problems with Quantifiers and Context

This very simple approach to head-driven generation with HPSG works successfully, if all the information required for generation is supplied as part of the initial logical form, and if this logical form can be identified with semantic content in the HPSG grammar. In particular, the initial logical form must be unifiable with the CONTENT feature of some appropriate lexical item which can serve as the pivot for the generation algorithm. However, this over-simplifies the way semantic information is represented in HPSG.

Wilcock and Matsumoto (1998) point out two severe difficulties for head-driven generation with HPSG. The first problem is how to handle quantifier scoping. Unscoped quantifiers are stored in the QSTORE feature, which is not part of CONTENT. At some point in a syntactic derivation, a quantifier is retrieved from the store and moved to the QUANTS feature, which is inside CONTENT. In the grammar rule which licenses this part of the derivation, the CONTENT of the mother includes the quantifier, but the CONTENT of the semantic head daughter does not, so the daughter is not the semantic head as required by the generation algorithm.

The second problem is how to handle contextual background conditions, such as the assumption that the person referred to by *she* is female. In HPSG, these conditions are specified in BACKGR, which is part of CONTEXT and is not part of CONTENT at all. If the conditions are included in the initial logical form, it will not be unifiable with the CONTENT feature of the semantic head, whose CONTENT does not include

CONTEXT features. As Wilcock and Matsumoto (1998) point out, even a simple sentence such as *She saw Kim* cannot be generated with this approach to head-driven generation with HPSG.

3 Head-Driven Generation (II)

In order to use a semantic head-driven generation algorithm with HPSG, while including unscoped quantifiers and contextual backgrounds, the role of semantic heads in the grammar needs to be consolidated, as proposed by Wilcock (1997). The problem is that semantic information in standard HPSG (Pollard and Sag, 1994) is fragmented into quantificational content, nuclear content, and context. Only nuclear content is consistently shared between the mother and the semantic head daughter, but nuclear content contains no more than referential indices attached to semantic roles. Quantificational content may or may not be shared, depending on whether quantifiers are retrieved. Contextual background is gathered from all daughters, with no special role for the semantic head.

The problem is in the mechanism for collecting together the quantifiers and background conditions of a phrase. In standard HPSG, a phrase's set of unscoped quantifiers (QSTORE) must be the union of the QSTOREs of the phrase's daughters (minus any quantifiers which are retrieved), and a phrase's set of contextual background conditions (BACKGR) must be the union of the BACKGRs of the phrase's daughters. This is known as *phrasal amalgamation*.

In contrast to this, head-driven generation requires all semantic information - nuclear, quantificational and contextual - to be shared between the mother and the semantic head, with semantic heads thereby playing a key role in the grammar. This requires the lexicalization of quantifier scoping, and the lexicalization of context, as described by Wilcock (1997). Basically, this means that a word's QSTORE must be the union of the QSTOREs of the word's arguments, and a word's BACKGR must be the union of the BACKGRs of the word's arguments. These requirements take the form of lexical constraints in HPSG theory, and this mechanism is known as *lexical amalgamation*.

3.1 Lexical amalgamation in ProFIT

A ProFIT implementation of lexical amalgamation is shown in Figure 3, from (Wilcock and Matsumoto, 1998). In the lexical entry for the verb *saw*, QSTORE sets and BACKGR sets are Prolog difference lists. The subject's BACKGR set B0-B1 and the object's BACKGR set B1-BN are amalgamated in the verb's BACKGR set B0-BN. The subject and object QSTORE sets, Q0-Q1 and Q1-QN, are similarly amalgamated in the verb's QSTORE Q0-QN.

```
lex( phon![saw|X]-X & @verb &
  synsem!loc!(
    cat!(head!<verb &
         val!(subj![@np &
              loc!(cat!head!case!<nom &
                   cont!index!Subj &
                   conx!backgr!B0-B1 &
                   qstore!Q0-Q1)] &
              comps![@np &
              loc!(cat!head!case!<acc &
                   cont!index!Obj &
                   conx!backgr!B1-BN &
                   qstore!Q1-QN)])) &
    cont!nuc!(seer!Subj & seen!Obj) &
    conx!backgr!B0-BN &
    qstore!Q0-QN) ).
```

```
'SHIP'    := synsem!loc!cont!Cont &
                 hd_dtr!synsem!loc!cont!Cont.
'QUIP'    := synsem!loc!qstore!QS &
                 hd_dtr!synsem!loc!qstore!QS.
'CHIP'    := synsem!loc!conx!Conx &
                 hd_dtr!synsem!loc!conx!Conx.

hd_nexus_ph := <hd_nexus_ph & @hd_ph &
     @'SHIP' & @'QUIP' & @'CHIP'.
```

Figure 3: Lexical Amalgamation and Logical Form Inheritance

The basic Semantics Principle, for semantic content only, was implemented by the template 'SemP' shown in Figure 1. In order to ensure the required sharing of unscoped quantifiers and background conditions between a phrase and its semantic head, the Semantics Principle is extended, as proposed by Wilcock (1997), to three principles: Semantic Head Inheritance Principle (SHIP), Quantifier Inheritance Principle (QUIP), and Contextual Head Inheritance Principle (CHIP). These are implemented by templates as shown in Figure 3, and the three principles are included in the grammar by the modified template for **hd_nexus_ph**, which replaces the earlier template in Figure 1. With these revisions, it is possible to include unscoped quantifiers and background conditions in the starting logical form, and perform head-driven generation successfully using the BUG1 generator.

4 Bag Generation

We now switch to non-head-driven approaches. Phillips (1993) proposed a bottom-up chart generation algorithm for use with indexed logical forms and categorial grammar in machine translation. An important property of the algorithm is that the order of terms in the logical form is not significant. The name *bag generation* is adopted from related work on shake-and-bake machine translation.

We now show how Phillips' algorithm can be used with HPSG grammar, with a bag of MRS relations instead of a bag of indexed logical terms. Though he presents the algorithm as a generator for categorial grammar, Phillips suggests that it can be adapted for use with phrase structure grammar (PSG), provided an indexed logical form is used and the indices are included in the syntactic categories. HPSG uses indices for agreement, and therefore includes the indices inside syntactic categories. By implementing HPSG as a PSG extended with typed feature structures (Section 1.2), and by implementing MRS as a similarly extended indexed QLF (Section 4.2), we can adapt Phillips' algorithm for use with HPSG.

4.1 A Simple Bag Generator

The adapted algorithm is shown in Figure 4. We use simple chart processing from the bottom-up chart parser of Gazdar and Mellish (1989). The main work is done by **start_gen**, which looks up the next term in the list of semantic terms, adds appropriate edges to the chart, and calls itself recursively on the remaining terms. Generation finishes when all edges have been built, and is successful if an inactive edge "spans" the whole input semantics, i.e. if an inactive edge's semantics are a *permutation* of the input semantics. `permutation/2` is a library predicate.

```
generate(Semantics,Category,String) :-
    abolish(edge,2),
    start_gen(Semantics),
    clause(edge(Category,[]),true),
    semantics(Category,EdgeSem),
    permutation(Semantics,EdgeSem),
    string(Category,String).

start_gen([]).
start_gen([Term|Terms]) :-
    foreach(lookup_term(Term,Category),
      add_edge(Category,[])),
    start_gen(Terms).

add_edge(Cat,Cats) :-
    clause(edge(Cat,Cats),true), !.
add_edge(Cat1,[]) :-
    asserta(edge(Cat1,[])),
    foreach(rule(Cat2,[Cat1|Cats]),
      add_edge(Cat2,[Cat1|Cats])),
    foreach(edge(Cat2,[Cat1|Cats]),
      add_edge(Cat2,Cats)).
add_edge(Cat1,[Cat2|Cats]) :-
    asserta(edge(Cat1,[Cat2|Cats])),
    foreach(edge(Cat2,[]),
      add_edge(Cat1,Cats)).

rule(Mother, Daughters) :-
    (Mother ---> Daughters).
```

Figure 4: A Simple Bag Generator

The algorithm assumes that Categories include surface strings and semantics as well as syntactic information (HPSG has Category *sign*, with string in PHON and semantics in CONTENT). For a particular grammar, the predicates `string` and `semantics` extract the string and semantics from the Category, giving a clean interface between the algorithm and the grammar. For a particular lexicon, `lookup_term` returns the Category of a lexical entry which includes the given semantic predicate, giving an interface between the algorithm and the lexicon.

4.2 MRS and QLF

MRS (Copestake et al., 1997) is a new semantic representation for use with HPSG. Like the indexed QLF of Phillips (1993), MRS was motivated by the needs of machine translation, where "flat" representations are preferred over strongly head-driven representations, as the head in one language may not correspond to the head in another language. Like the QLF, MRS depends on the use of indices to represent dependencies between the terms in the flat list. HPSG previously used indices only for entities of type *nominal_object*, to assign them to semantic roles as participants in ***states of affairs*** and to carry agreement features. In MRS, indices are also used for events, as in the QLF.

A major difference between MRS and the QLF is that MRS uses typed feature structures instead of ordinary logical terms. Each element in the list is an HPSG typed feature structure of type *relation*. This facilitates the integration of MRS into HPSG. While the QLF logical terms could be represented in Prolog, we need ProFIT to extend the terms with typed feature structures for MRS. We thus implement MRS as an extension of QLF in the same way that we implement HPSG as an extension of PSG.

Another major difference, which makes MRS a significant improvement over the QLF, is that MRS supports the representation of quantifier scope (either fully resolved or underspecified). This is done by including ***handles*** which label each term in the list. Scope can be represented by means of the handles, while maintaining the flat list representation, without the nesting required when operators are used to represent scope. As a musical joke about semantic *composition*, the handle feature is named HANDEL and the list feature is named LISZT.

4.3 Non-Head-Driven Semantics

In the semantics of Pollard and Sag (1994), which was derived from Situation Semantics, semantic composition is performed by recursive unification of semantic feature structures, to produce a single complex semantic feature structure in the semantic head. This semantic structure is structure-shared between a phrase and its semantic head daughter, by the Semantics Principle. This form of semantic representation is therefore suitable for semantic head-driven generation.

By contrast, in the flat MRS representation, semantic composition is performed by concatenation of the LISZTs of a phrase's daughters to give the LISZT of the mother. The LISZT of the semantic head daughter will not be the same as the LISZT of the mother. MRS is therefore suitable for bag generation, but not for head-driven generation[1]. The Semantics Principle must be scrapped or redefined. We show a simple revision of 'SemP' in Figure 5, in which INDEX and HANDEL (but not LISZT) are shared between mother and semantic head, in contrast to 'SemP' in Figure 1 which shares the whole CONTENT structure.

```
index(I)  := synsem!loc!cont!index!I.
handel(H) := synsem!loc!cont!handel!H.
liszt(L)  := synsem!loc!cont!liszt!L.
```

```
'SemP'    := @handel(H) & @index(I) &
     hd_dtr!(@handel(H) & @index(I)).
```

```
@hd_subj_ph & phon!P0-PN & @liszt!L0-LN &
    hd_dtr!(Head & @liszt!L1-LN) &
            synsem!loc!cat!val!subj![S]) &
    subj_dtr!(Subj & @liszt!L0-L1 &
              synsem!S)
---> [Head & <phrase & phon!P1-PN,
      Subj & <phrase & phon!P0-P1].
```

```
@hd_comp_ph & phon!P0-PN & @liszt!L0-LN &
    hd_dtr!(Head & @liszt!L0-L1 &
            synsem!loc!cat!val!comps![C]) &
    comp_dtrs![Comp & @liszt!L1-LN &
               synsem!C]
---> [Head & <word & phon!P0-P1,
      Comp & <phrase & phon!P1-PN].
```

Figure 5: Semantics Principle, LISZT Composition

We use Prolog difference lists to implement the LISZT feature for efficient concatenation, as we did with PHON. In the case of LISZT, the predicate `semantics` hides this representation from the algorithm. LISZT concatenation is added to the PSG rules for HPSG schemata, as shown in Figure 5.

A full MRS representation includes a top-level handle and a top-level index, which are specified separately from the flat list of terms. A top-level index is also mentioned by Phillips (1993). We have ignored these in the bag generation algorithm, to simplify adaptation to incremental generation in Section 5. However, the top-level index specifies what the semantics is *about*. For example, a simple QLF representation: `[man(m), walk(e,m)]` could mean either "A man walked" or "A man who walked". These are only distinguished by the top-level index (e or m respectively). In this respect, the top-level index specifies the *topic*, part of information structure, which we will come back to in Section 5.3.

[1]Head-driven generation with MRS would require the lexical amalgamation of LISZT.

5 Incremental Generation

A variant of Phillips' algorithm was used in PLUS (a Pragmatics-based Language Understanding System) for surface generation of dialogue responses from an indexed QLF using a categorial grammar. Jokinen (1996) gives an overview of the PLUS system, and describes the planning of dialogue responses to be passed to the surface generator as indexed QLF representations. Lager and Black (1994), discussing the PLUS surface generator, suggest that the algorithm and the QLF are suitable for incremental generation.

We now show how the bag generation algorithm can be modified for incremental generation with HPSG and MRS, and discuss differences between categorial grammar and HPSG which incremental generation emphasises. From an incomplete bag and a partial utterance, the generator attempts to *continue* the utterance as further semantic terms are added. We include a simple form of repair when the generator cannot find a way to continue the utterance. We ignore the issue of real-time processing here, and deal only with order of inputs and outputs. Though the order of terms in the bag is not itself significant, the order in which terms are added influences the utterance very strongly.

5.1 An Incremental Algorithm

The basic incremental algorithm is shown in Figure 6. `incremental_gen` is initialized with an empty bag of semantic terms and an empty list of strings uttered so far. The procedure inputs a new semantic term, looks it up in the lexicon, and adds a new edge for each word found, thereby triggering construction of further edges. When all edges have been built, the procedure calls `utter` and then recursively calls itself with the augmented bag of terms and the augmented list of strings uttered.

```
incremental_generation :-
    abolish(edge,2),
    incremental_gen([],[]).
```

```
incremental_gen(Sem0,Phon0) :-
    input_term(Term),
    ( Term = end_of_file
    ; foreach(lookup_term(Term,Category),
        add_edge(Category,[])),
      utter([Term|Sem0],Phon0,Phon1),
      incremental_gen([Term|Sem0],Phon1)
    ).
```

Figure 6: Incremental Generation

A set of utterance rules is shown in Figure 7, numbered from 1 to 4. Each rule is attempted, in the order given, until an utterance is produced, or the default rule 4 is reached, which simply utters nothing and allows `incremental_gen` to input another semantic term.

Utterance Rule 1 succeeds if it finds an inactive edge (a complete syntactic constituent) which spans the bag of semantic terms, and whose PHON list is a continuation of the list uttered so far. If so, the new part of PHON is output by `utter_continue`. (An edge "spans" a bag of terms if its own semantics is a permutation of the terms in the bag).

Utterance Rule 2 performs a simple form of repair, when there is a complete syntactic constituent which spans the semantics, but its PHON does not continue the utterance which has been started. In this case, `utter_repair` finds the minimum backtracking change needed to effect the repair, and utters the new part only, preceded by "Er,...".

If there is no complete syntactic constituent which spans the bag of terms, we could wait (Rule 4), or, if we want the generator to be more "talkative", we can use *active* edges from the chart. Utterance Rule 3 continues the utterance with any active edge whose semantics matches the new semantic term. The best set of rules needs to be found by further work.

5.2 Categorial Grammar and HPSG

Previous work on incremental generation has usually assumed that utterances must correspond to syntactic constituents. For example, discussing the possible adaptation of Phillips' algorithm to incremental generation, Lager and Black (1994) point out that some versions of Categorial Grammar (CG) would make the generator more talkative, by giving rise to "a more generous notion of constituency". However, in order to use HPSG for incremental generation, we must challenge the underlying assumption.

The basic approach to combining a head (verb) with its arguments (subject and complements) is significant here. Whereas in CG a head may be combined with its arguments one by one, giving a series of unsaturated

```
% 1. Inactive edge: Continue
utter(Sem1,Phon0,Phon1) :-
    span_inactive(Sem1,Cat),
    string(Cat,Phon1),
    utter_continue(Phon0,Phon1),!.
% 2. Inactive edge: Repair
utter(Sem1,Phon0,Phon1) :-
    span_inactive(Sem1,Cat),
    string(Cat,Phon1),
    utter_repair(Phon0,Phon1),!.
% 3. Active edge: Continue
utter([Term|_],Phon0,Phon1) :-
    span_active([Term],Cat),
    string(Cat,Phon1),
    utter_continue(Phon0,Phon1),!.
% 4. Active Edge: Wait
utter(_,Phon,Phon).
```

```
span_inactive(Sem, Cat) :-
    clause(edge(Cat,[]),true),
    semantics(Cat,EdgeSem),
    permutation(Sem,EdgeSem).
span_active(Sem, Cat) :-
    clause(edge(Cat,[_|_]),true),
    semantics(Cat,EdgeSem),
    permutation(Sem,EdgeSem).
utter_continue(Phon0,Phon1) :-
    append(Phon0,NewPhon,Phon1),
    write('    ... '),write(NewPhon),nl.
utter_repair(Phon0,Phon1) :-
    repair(Phon0,Phon1,Repair),
    write('    Er, ... '),write(Repair),nl.
repair([Word|Words],[Word|Words1],Repair) :-
    repair(Words,Words1,Repair).
repair(_,Words1,Words1).
```

Figure 7: Utterance Rules

intermediate constituents until a saturated one is completed, in HPSG a head (a verb) is usually combined first with all of its complements in one constituent (a VP), and then this is combined with the subject.

In incremental generation of English, after the subject has been generated, further semantic input may enable the verb to be generated next. In this case, CG may allow the subject and verb to be combined into a valid syntactic constituent, but HPSG will not recognise this as a constituent. If an incremental generator only utters valid constituents, an HPSG-based generator must wait, after uttering the subject, until all the complements of the verb have been identified, before uttering the verb as part of the complete verb phrase. This is a significant problem in using HPSG for incremental generation.

However, in deciding what units should be uttered it is *information structure* which should be decisive, not syntactic constituency. From this point of view, which we will discuss in the next section, the problem with HPSG's notion of constituency can be reduced to a computational matter. As the verb is put into the chart as an active edge before its complements are identified, it can also be uttered, if desired, without waiting for the VP edge to be completed. This may be computationally slightly awkward, but it is feasible.

5.3 Information Structure

The incremental system described so far does not take information structure into account, but is driven purely by the order in which semantic terms are supplied. In a "shallow" system (for example, a simultaneous interpretation system) it may be necessary to rely on the order to *implicitly* realize information structure, but if the system includes "deep" processing, with logical inferencing, then it is not satisfactory to rely on the order in which an inference component happens to produce its results. Such a system needs *explicit* management of information structure.

Engdahl and Vallduví (1996) argue that information structure is a distinct dimension, and locate INFO-STRUCT in the HPSG CONTEXT feature rather than CONTENT. However, the representation they propose is purely syntactic: Link (topic) and Focus are equated with syntactic constituents (NPs and VPs) which realize the topic concept and the focus information. In a footnote, they accept that it would be more appropriate for the value of INFO-STRUCT to be structure shared with the value of CONTENT.

Steedman (1991) argues that there is a correspondence between information structure, intonation and syntactic constituency, and it is a strength of CG that it allows suitable syntactic constituents. Engdahl and Vallduví (1996) argue that there is no correspondence between information structure and syntactic constituency, and that it is a strength of HPSG's multidimensional representation that we are not forced to assume such a correspondence.

Perhaps both approaches over-emphasise the role of syntax, in an area where semantics and pragmatics should be more central. In the PLUS system, a pragmatics-based Dialogue Manager (Jokinen, 1996) explicitly manages information structure, using a semantics-based representation. Central Concept (topic) and NewInfo (focus) are represented using QLFs with explicit indices for discourse referents. This facilitates distinguishing old and new information, but the QLF lacks explicit representation of scope. This suggests that it would be interesting to include focus scope ("narrow focus" and "wide focus") in an MRS-based representation with HPSG, in a similar way to quantifier scope.

Response planning in the PLUS Dialogue Manager always starts from NewInfo, adding other content (such as Central Concept linking) only when necessary. This gives rise to natural, elliptical surface generation. It also makes possible a proper approach to time constraints. This approach to generation from NewInfo has been developed further by Jokinen et al. (1998).

6 Other Approaches

In conclusion we take a brief look at some other approaches. One approach to generation is to take a grammar which was developed for parsing, and invert it for generation. Since most existing HPSG grammars were developed for parsing, this approach has a point. It has been applied to HPSG by Wintner et al. (1997). Their work is also representative of the recent development of new implementations for HPSG based on abstract machines, which are expected to supercede the current Prolog-based implementations such as the ProFIT system used here.

As noted in the introduction, another approach is to convert HPSG into TAG (Kasper et al., 1995) for generation. Similar work on compiling HPSG for efficient parsing (Torisawa and Tsujii, 1996) should be equally applicable to generation.

Finally, we come back to Systemic Functional Grammar (SFG) which we contrasted with HPSG in the introduction. Since SFG and HPSG share a similar underlying logic of typed feature structures, it should be possible to use tools such as ProFIT and AMALIA (Wintner et al., 1997) for SFG, by implementing the system network as a type hierarchy.

A more surprising approach would be to attempt to use a systemic generation algorithm with an HPSG grammar. A systemic generation algorithm traverses the system network, making choices within the systems as it goes, and collecting realization rules which will decide the final output. To apply this to HPSG would mean implementing the HPSG type hierarchy as a system network, and traversing it with a systemic generation algorithm, making choices within the subtypes as it goes, and collecting type constraints which will decide the final output.

Acknowledgements

This work was done as a visiting researcher of Sharp Corporation at UMIST, England and at NAIST, Japan. I am grateful to Mr Yoshikazu Nakagawa (Sharp), Prof Jun-ichi Tsujii (UMIST) and Prof Yuji Matsumoto (NAIST) for making this possible. I am also indebted to Bill Black, Yasuharu Den, Kristiina Jokinen and John Phillips for their ideas, discussions and comments on earlier drafts.

References

Ann Copestake, Dan Flickinger, and Ivan Sag. 1997. Minimal Recursion Semantics: An introduction. Ms. Stanford University.

Elisabet Engdahl and Enric Vallduví. 1996. Information Packaging in HPSG. In C. Grover and E. Vallduví, editors, *Edinburgh Working Papers in Cognitive Science, Vol. 12: Studies in HPSG*, pages 1–32. University of Edinburgh.

Gregor Erbach. 1995. ProFIT: Prolog with Features, Inheritance, and Templates. In *Seventh Conference of the European Chapter of the Association for Computational Linguistics*, pages 180–187, Dublin.

Gerald Gazdar and Christopher Mellish. 1989. *Natural Language Processing in Prolog: An Introduction to Computational Linguistics.* Addison-Wesley.

Masahiko Haruno, Yasuharu Den, and Yuji Matsumoto. 1996. A chart-based semantic head driven generation algorithm. In G. Adorni and M. Zock, editors, *Trends in Natural Language Generation: An Artificial Intelligence Perspective*, pages 300–313. Springer.

Kristiina Jokinen, Hideki Tanaka, and Akio Yokoo. 1998. Planning dialogue contributions with new information. In *9th International Workshop on Natural Language Generation*, Niagara-on-the-Lake, Ontario.

Kristiina Jokinen. 1996. Reasoning about coherent and cooperative system responses. In G. Adorni and M. Zock, editors, *Trends in Natural Language Generation: An Artificial Intelligence Perspective*, pages 168–187. Springer.

Robert Kasper, Bernd Kiefer, Klaus Netter, and K. Vijay-Shanker. 1995. Compilation of HPSG to TAG. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 92–99, Cambridge, MA.

Torbjörn Lager and William J. Black. 1994. Bidirectional incremental generation and analysis with categorial grammar and indexed quasi-logical form. In *7th International Generation Workshop*, pages 225–228, Kennebunkport, Maine.

John D. Phillips. 1993. Generation of text from logical formulae. *Machine Translation*, 8:209–235.

Carl Pollard and Ivan A. Sag. 1994. *Head-driven Phrase Structure Grammar*. CSLI Publications and University of Chicago Press.

Mark Steedman. 1991. Structure and Intonation. *Language*, 67(2):260–296.

Kentaro Torisawa and Jun'ichi Tsujii. 1996. Computing phrasal-signs in HPSG prior to parsing. In *16th International Conference on Computational Linguistics (COLING-96)*, pages 949–955, Copenhagen.

Gertjan van Noord. 1990. An overview of head-driven bottom-up generation. In R. Dale, C. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation*, pages 141–165. Academic Press.

Graham Wilcock and Yuji Matsumoto. 1996. Reversible delayed lexical choice in a bidirectional framework. In *16th International Conference on Computational Linguistics (COLING-96)*, pages 758–763, Copenhagen.

Graham Wilcock and Yuji Matsumoto. 1998. Head-Driven Generation with HPSG. In *17th International Conference on Computational Linguistics (COLING-ACL-98)*, Montreal (to appear).

Graham Wilcock. 1997. Lexicalization of Context. 4th International Conference on HPSG, Ithaca, NY. To appear in G. Webelhuth, J.-P. Koenig and A. Kathol, editors, *Lexical and Constructional Aspects of Linguistic Explanation*. CSLI Publications (in press).

Shuly Wintner, Evgeniy Gabrilovich, and Nissim Francez. 1997. AMALIA - a unified platform for parsing and generation. In *Recent Advances in Natural Language Processing (RANLP-97)*, pages 135–142, Tzigov Chark.