An Implemented HPSG Grammar for SHARDS

Raquel Fernández

Department of Computer Science
King's College London
raquel@dcs.kcl.ac.uk

Abstract

This report is an extension of (Gregory 2001) and presents an overview of the implementation of the HPSG grammar currently used in SHARDS —a Semantically-based HPSG Approach to the Resolution of Dialogue Fragments (Ginzburg, Gregory, and Lappin 2001). The grammar is based on the theoretical framework presented in (Ginzburg and Sag 2001) and it is encoded in ProFIT (Erbach 1995).

1 Introduction

This report describes the basic components and structure of the implemented HPSG grammar currently used in the SHARDS system (Ginzburg, Gregory, and Lappin 2001). The report is organised as follows: we start by introducing the main components of SHARDS, giving a quick overview of the particular version of HPSG we assume and describing ProFIT, the language in which the grammar is encoded. We then present the grammar implementation in Section 2. In Section 3 we discuss the analysis of fragments and the resolution procedure of SHARDS. Finally, we conclude outlining some future work.

1.1 SHARDS

SHARDS is an implemented system which provides a procedure for computing the interpretation of clausal fragments. The system comprises two main components: a grammar and a resolution procedure. The grammar, encoded in ProFIT (Erbach 1995), assigns HPSG features structures to English sentences. Once a sentence has been parsed, the second component of the system resolves the ellipsis sites on the basis of contextual information contained in preceding sentences, located in a structured record stored in memory.[1]

The current grammar, which is the main topic of this report and will be described in detail in the next sections, is a substantially modified version of

[1] For a complete description of the context and ellipsis resolution postprocessor of SHARDS see (Gregory 2001).

the grammar described in (Gregory 2001), employed by (Lappin and Gregory 1997), (Gregory and Lappin 1999) and (Ginzburg, Gregory, and Lappin 2001).

1.2 A particular version of HPSG

The grammar currently used in SHARDS follows a particular version of the HPSG framework, initiated by (Sag 1997) and developed in (Ginzburg and Sag 2001).

One of the main features of this recent theoretical work is the assumption of a construction-based approach, wherein the notion of construction is analysed in terms of types, type hierarchies and type constraints. Within this assumption, generalisations about constructions (i.e. information about phrases) are encoded by cross-classifying them in multidimensional type hierarchies. The typed-based approach to construction analysis is accurately modelled by a hierarchical system of construction types, stating constraints on the most specific types of construction, on the most general type *phrase*, or on any of the intermediate types recognised by the grammar.

The semantics developed for such grammatical proposal is based on the framework of Situation Semantics. Phrases are classified not only in terms of their phrase structure schema or X-bar type, but also with respect to a further informational dimension of CLAUSALITY. Clauses are divided into *inter alia* declarative clauses, which denote propositions, and interrogative clauses denoting questions.

The grammar pretends to model utterance types, with special attention to fragments and dialog. To account for this two highly context-dependent phenomena, the theory of context in dialogue developed in the framework of KOS (Ginzburg 1996; Ginzburg 2002; Cooper, Larsson, Hieronymus, Ericsson, Engdahl, and Ljunglof 2001) is integrated into the HPSG formalisation.

1.3 ProFIT

ProFIT (Erbach 1995) is an extension of Prolog which allows to declare an inheritance hierarchy, features and templates. Both typed features structures.[2] and Prolog terms can be used in ProFIT programmes. ProFIT compiles the typed feature terms into a Prolog term representation, so that no special unification algorithm nor meta-interpreter is needed. ProFIT thus is an efficient language for the implementation of grammars developed with type features structures in Prolog programmes.

A ProFIT programme consists of:

- Type declarations
- Feature declarations

[2]For a formal characterisation of typed feature structures see (Carpenter 1992). Sometimes, e.g. in Logic Programming, typed feature structures are referred to as sorted features structures.

- Templates

1.3.1 Type declarations

In order to be able to use typed features terms, the types and features must be declared in advance. The type declarations begin with a declaration of the most general type `top` and all subtypes not explicitly declared under other supertypes must be subtypes of `top`. The subtype declarations have the syntax given in (1):

(1) $Supertype > [Subtype_1, ..., Subtype_n]$

The declaration in (1) states that every $Subtype_i$ is a subtypes of *Supertype* and that all $Subtype_i$ are mutually exclusive, i.e. they are disjoint. It is also possible to declare subtypes that are not mutually exclusive as in the case of multi-dimensional inheritance hierarchies. In this kind of hierarchies each dimension is declared as a separate list of subtypes connected by the operator $*$, as shown in (2):

(2) $Supertype > [Subtype_1,..., Subtype_n] * ... * [Subtype_{k.1},..., Subtype_{k.m}]$

1.3.2 Feature declarations

Following the notion of appropriateness of Carpenter's logic of typed feature structures, in a ProFIT programme one must declare which features are introduced by each type. A feature is introduced only at the most general type for which the feature is appropriate and the features introduced by a type are inherited by all its subtypes. Each feature has a particular type as its value. When this type restriction is omitted, then the feature value is assumed to be of type `top`. The syntax of feature declarations is given in (3). Feature declarations can also be combined with type declarations using the syntax given in (4).

(3) *Type* `intro` $[Feature_1 : type\ restr_1, ..., Feature_n : type\ restr_n]$

(4) $Type > [Subtype_1,...]$ `intro` $[Feature_1 : type\ restr_1,...]$

1.3.3 Templates

Templates are an abbreviatory device to encode frequently used structures. ProFIT templates are defined by expressions of the form shown in (5) and are called using the prefix @ (@*Name*).

(5) $Name := Definition$

The *Name* may be a predicate whose arguments usually also occur within the *Definition*. The *Definition* is a ProFIT term, consisting of a specification of a type (6), a specification of a path using **!** to list features and their values (7) or a conjunction of terms using the sign & (8).

(6) $<$ *Type*

(7) i. *Feature* ! *Value*

ii. *Feature* ! *Feature* ! *Value*

(8) *Term* & *Term*

The sign >>> provides a mechanism to abbreviate paths, as long as the type and feature declarations are strong enough to ensure that there is a unique path. The *Definition* may also include variables and template calls. We will see several example of ProFIT templates in Section 2.3.

2 The grammar implementation

The implemented grammar includes:

- A type hierarchy, i.e. a linguistic ontology,
- a declaration of which features are appropriate for each type,
- a declaration of what type of value is appropriate for each feature,
- a specification of all constraints that (either phrasal or lexical) types must satisfy and
- a set of grammar rules, to be used by the bottom-up parser.

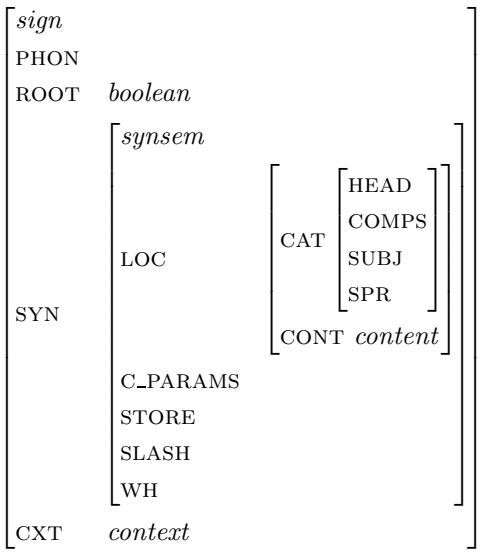
The grammar must thus provide a complete specification of what types of feature structures exist and how those types are organised into a hierarchy. It must also specify which features are appropriate for each grammatical type, as well as what value is appropriate for each feature.

2.1 Linguistic ontology, feature typing and appropriateness

The grammar follows the HPSG sign-based approach in which all linguistic signs, both phrasal and lexical, are modelled as typed feature structures. The ProFIT declaration for the type *sign* states that its immediate subtypes are *phrase* and *lex(ical sign)* and that the features appropriate for such a type are PHON, SYNSEM, CONTEXT and ROOT. For each feature F_i, it is declared the appropriate type T_i of its value (with the syntax F_i:T_i). In the present program, when no T_i is specified, that means that the value of F_i is a list.

(9)
```
sign >     [lex,phr]
           intro [phon,root:bool,syn:synsem,cxt:context].
synsem >   [canonical,gap]
           intro [loc:local,c_params,store,slash,wh].
local      intro [cat:category,cont:content].
category   intro [head:heads,comps,subj,spr].
```

According to these ProFIT declarations, the AVM representation for the type *sign* would be the following:



2.1.1 The semantic ontology

With some simplifications,[3] our semantic type hierarchy follows the semantic ontology developed in (Ginzburg and Sag 2001)[4], which *inter alia* distinguishes between questions, propositions, facts, outcomes, situations and SOAs. Here is part of the type and feature declarations of our grammar:

(10)
```
content >     [message,soa,param,quantifier,
              situation,addressee,speaker].
message >     [proposition, question, fact, outcome].
proposition   intro [sit:situation, soa:soa].
question      intro [params, prop:proposition].
soa           intro [quants, nucleus:nucleus].
quantifier    intro [det:det_rel,restind:param].
param         intro [index:index,rest].
```

We will focus our attention on the *proposition* and *question* types, which are the semantic types associated with the content of declarative and interrogative clauses, respectively. These are the only clausal types the current grammar covers at the moment.

The type *proposition* introduces two features, the feature SIT, whose value will be the situation involved in the relevant proposition, and the feature SOA.

[3] We don't distinguish between austinian and propositionally constructed content, neither between realist and irrealist SOA, for instance.

[4] See chapter 3 specially.

All SOAs allow the features QUANT(IFIERS) and NUCLEUS. The other relevant subtype of *message* is the type *question* which introduces a feature PROP, whose value is a proposition. Given that in the semantic theory we follow questions are identified with proposition abstracts (with open propositions), the type *question* is also appropriate for the feature PARAMS, whose value will be a (possibly empty) set of parameters—implemented as a list in the current grammar — corresponding to the set of entities that gets abstracted away.[5] The feature declarations just introduced are express in AVM notation in (11):

(11) a.
$$\begin{bmatrix} \textit{proposition} \\ \text{SIT } \textit{situation} \\ \text{SOA} \begin{bmatrix} \textit{soa} \\ \text{QUANTS} \\ \text{NUCLEUS} \end{bmatrix} \end{bmatrix}$$

b.
$$\begin{bmatrix} \textit{question} \\ \text{PARAMS} \\ \text{PROP} \begin{bmatrix} \textit{proposition} \\ \text{SIT } \textit{situation} \\ \text{SOA} \begin{bmatrix} \textit{soa} \\ \text{QUANTS} \\ \text{NUCLEUS} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

2.1.2 Context and Contextual Parameters

Like (Ginzburg and Sag 2001), we adopt an approach based on the theory of context in dialogue developed in the framework of KOS (Ginzburg 1996; Ginzburg 2002; Cooper, Larsson, Hieronymus, Ericsson, Engdahl, and Ljunglof 2001). Following the HPSG formalisation of this approach, in our grammar the value of the feature CONTEXT contains three features: C(ONTEXTUAL)-INDICES, MAX-QUD (Maximal Question Under Discussion) and SAL-UTT (Salient Utterance).

(12)
```
context   intro [c_indices:c_inds, max_qud, sal_utt].
c_inds    intro [c_spkr:index, c_addr:index].
```

These C-INDICES encode contextual information about the speaker and the addressee of the utterance. The attribute MAX-QUD, whose value is of type *question*, represents the question currently under discussion. SAL-UTT takes as its value sets of type *local* and represents a distinguished constituent of the utterance whose content is the current value of MAX-QUD. SAL-UTT can be thought as a means for underspecifying the focal (sub)utterance or as a potential *parallel element* in the sense of (Dalrymple, Pereira, and Shieber 1991). Specifically, it

[5] Parameters are contextual indices carrying some semantic restrictions.

is compute as the (sub)utterance associated with the role which receives widest scope within MAX-QUD.

These two features make possible an account of fragments (like short answers and sluices) without the need to resort to any form of syntactic reconstruction, while still encoding a limited amount of syntactic parallelism.[6] The values of MAX-QUD and SAL-UTT will be computed by the Context Resolution Procedure of SHARDS.

Our grammar also includes a features C(ONTEXTUAL)-PARAM(METER)S, introduce by (Ginzburg and Cooper 2001). The value of this feature is the set of all contextual parameters of an utterance (proper names, indexicals). The presence of this feature allow signs to play a role similar to the role traditionally associated with "meanings", i.e. as functions from contexts to contents.[7] What the *content* attribute of a sign encodes is the value that the meaning function takes, given the values of the contextual parameters.

2.2 A multi-dimensional phrase hierarchy

In the last years, multiple inheritance has been extensively used for the description of different aspects of natural languages, some examples being the structure of the lexicon (Koenig 1999) and the syntax of English relative clauses (Sag 1997). Our grammar, following (Sag 1997) and (Ginzburg and Sag 2001), uses a multi-dimensional type hierarchy to classify phrases.

As in (Ginzburg and Sag 2001), phrases are cross-classified along two dimensions: Headedness and Clausality. The choices within one dimension are mutually exclusive, however a structure can be assigned types from different dimensions without postulating a unique most general common subtype.[8] (13) illustrates part of the implemented phrase hierarchy according to the syntax described in Section 1.2.1. The first line of the example is the declaration of the two dimensions, Headedness and Clausality, as two hierarchies belonging to the type *phrase*. Next, the declaration of the subtypes for each dimension is shown. In (14) we can see the declaration of the maximal phrasal types.[9]

(13)
```
phr >       [headed,nheaded]*[clause,nclause].
headed >    [head_comps,head_subj,head_spr,head_filler,sai,
            head_adj,head_only].
clause >    [core_cl,rel_cl].
core_cl >   [declarative,interrogative,imperative,exclamative].
```

[6] As SAL-UTT has as its value an object of type *local*.

[7] Montague (1974a), Kaplan(1989).

[8] In Carpenter's formalisation of typed feature logic, the type hierarchy is required to be a bounded complete partial order (BCPO), which means that any two types which have a common subtype must have a unique most general common subtype. In other words, the type hierarchy is required to be one-dimensional. In contrast, in a multiple inheritance a type has supertypes from different hierarchies or "dimensions" without the need of introducing a unique most general common subtype.

[9] Recall that a maximal type is a type with no subtypes.

(14)
```
declarative >     [decl_hd_su_cl,decl_ns_cl,inv_decl,decl_frag_cl].
interrogative >   [wh_intr,polar_intr,sluice_intr,is_intr].
head_subj >       [decl_hd_su_cl].
head_filler >     [wh_intr].
wh_intr >         [ns_wh_intr, su_wh_intr].
sai >             [polar_intr,inv_decl].
head_only >       [decl_ns_cl, is_intr,hd_frag_ph,root_cl].
hd_frag_ph >      [decl_frag_cl,sluice_intr].
is_intr >         [reprise_intr,dir_is_intr].
```

Note that multiple inheritance arises when some type Y occurs in the right-hand side of more than one type declaration, which is the case for most of the subtypes declared in (14).

One important thing to note is that in this version of the grammar elliptical constructions are analyse by means of the type `hd_frag_ph` (*head-fragment-phrase*), which is a headed phrase. In contrast, in former versions of the grammar used by SHARDS, fragments were analysed as non-headed bare phrases, the fragment being the non-head daughter.

2.3 Constraints as templates

One of the main differences between the present implementation and earlier versions of the grammar (which reflects a tendency in the evolution of the HPSG framework) is the substitution of most of the grammar principles (the valence Principle, the Semantic Principle and so on), which were implemented as prolog rules, by constraints on phrasal types expressed through ProFIT templates.

For example, (15) one of the prolog clauses to implement the Valence Principle in previous versions of the grammar,[10] has been replaced by the template in (16) representing the Empty Complements Constraint((Ginzburg and Sag 2001), p. 33):

(15)
```
val_p  (<head_comps &
        syn!cat!comps!MComps &
        syn!cat!subj!Subj &
        head_dtr!Head &
        head_dtr!syn!cat!comps!HComps &
        head_dtr!syn!cat!subj!Subj &
        nhead_dtrs!Dtrs):-
                                extract_syn(Dtrs,Syn),
                                append(MComps,Syn,HComps),
                                val_p(Head).
```

(16)
```
ecc  :=  <phr &
         syn!loc!cat!comps![].
```

[10] See (Gregory 2001).

The template in (16) states that the value of COMPS will be the empty list for all structures of type *phrase*. Since complements are introduced as sisters of the lexical head by a constraint on the type *head_complement_phrase*, (16) guarantees that, within any phrase, complements have already been cancelled from the COMPS list at he level of the phrase's lexical head.

Thus, following the construction-based approach, each type of phrase obeys type-specific constraints. The constraints on types within the Headedness dimension mainly express notions related to the immediate dominance schemata (17), while the constraints in the Clausality dimension mostly encode semantic properties of different clausal types (18). In our grammar, a template *Definition* always has the same structure: First, the type it applies to, using the prefix <; second, the template calls, if any; and finally, the constraints on paths to enforce either unification or the presence of specific values.

(17)
```
head_subj(N,V,P)  :=  <head_subj &
                      @ecc &
                      @head_feature_p &
                      phon!P &
                      syn!loc!cat!subj![] &
                      head_dtr!V &
                      head_dtr!<phr &
                      head_dtr!syn!loc!cat!subj![Subj] &
                      nhead_dtrs![N & syn!Subj].
```

(18)
```
intr  :=  <interrogative &
          @core_cl &
          syn!loc!cont!<question.
```

The template in (17) encodes the constraint associated with the phrasal type *head-subject-phrase*. Its head daughter is of type *phrase* and it has only one non-head daughter, whose SYNSEM value is identified with the value of the feature SUBJ of the head daughter that selects it. The phrase itself has an empty value for the corresponding feature. All this information interacts with the constraints expressed in the `ecc` and `head_feature_p` (see below) templates. On the other hand, the template in (18) represents a constraint on interrogative clauses and it encodes the information that the content of an interrogative is of type *question*. It also calls the `core_cl` template which enforce the verbal form to be of type *clausal*.

2.3.1 The GHFP as a set of non-default constraints

(Ginzburg and Sag 2001) (p. 33) formulate a generalisation of the Head Feature Principle as a default constraint on phrases of type *headed-phrase*:

(19) Generalized Head Feature Principle (GHFP)

$$\begin{bmatrix} \textit{hd-ph} & \\ \textsc{synsem} & / \boxed{1} \\ \textsc{head-dtr} & \begin{bmatrix} \textsc{synsem} & / \boxed{1} \end{bmatrix} \end{bmatrix}$$

The GHFP requires that the SYNSEM value of the mother of a headed phrase is identical to that of its head daughter by default.[11] The GHFP replaces the non-defeasible Head Feature Principle and the Valence Principle of (Pollard and Sag 1994) and allows considerable simplification of the grammar. As a default constraint, it requires that the features within the SYNSEM value of the head daughter and those of its mother have identical values, except in cases where this is explicitly contradicted by constraints on specific subtypes of *hd-ph*.

Since our implementation does not make use of non-monotonicity, the GHFP has been replaced with a set of non-default constraints expressed by means of templates which are called by particular subtypes of *head-phrase*.

(20)
```
head_feature_p  :=  syn!loc!cat!head!X &
                    head_dtr!syn!loc!cat!head!X.
```

(21)
```
ghd_feature_p  :=  syn!slash!Slash &
                   syn!store!Store &
                   head_dtr!syn!slash!Slash &
                   head_dtr!syn!store!Store.
```

(22)
```
ghfp  :=  <nclause &
          syn!Syn &
          head_dtr!syn!Syn.
```

(20) is the most general case: almost all phrases call this template. This is not the case, however, for the type *head-only-phrase*, where the HEAD value is recovered from the context (see Section 3).

The template in (22) is the one that corresponds to the GHFP, but it will be only called by bare non-clausal phrases where it is possible to identify the SYNSEM value of the mother with that of its head daughter.

2.3.2 The Clausality dimension

The constraints in the Clausality dimension mostly encode the semantic properties of clauses. The content of a clause will always be some subtype of the semantic type *message*, as ensured by a constraint on the type *clause* expressed by the following template:

(23)
```
clause  :=  <clause &
            syn!loc!cont!<message.
```

[11] The '/' notation indicating default information is introduced by (Lascarides and Copestake 1999).

Additional templates, like the one in (18) above, specify the content of each clausal type in order to establish a correlation between clausal constructions and types of meaning.

Given that the content of a VP is a SOA, (Ginzburg and Sag 2001) posit the following default constraint on the type *declarative_clause*:

(24)
$$\begin{bmatrix} \textit{decl-cl} & \\ \text{CONT} & \begin{bmatrix} \text{SOA} & / \boxed{1} \end{bmatrix} \\ \text{HEAD-DTR} & \begin{bmatrix} \text{CONT} & / \boxed{1} \end{bmatrix} \end{bmatrix}$$

Again, in our implementation this default constraint has been replaced by a template only identifying the mother's nucleus with that of its head daughter. Such a template is called by the appropriate maximal declarative types which supply the remaining information.

(25)
```
decl_sem  :=  syn!loc!cont!sit!<s &
              syn!loc!cont!soa!nucleus!Soa &
              head_dtr!syn!loc!cont!nucleus!Soa.
```

2.4 The DCG rules

The grammar also comprises a set of Prolog grammar rules to be used with the bottom-up parser `buparser2.pl` (Clocksin and Mellish 1984; Gazdar and Mellish 1989). These rules are essentially of the same form as DCG rules, with the difference that in a ProFIT programme the operator `--->` is used, instead of the usual `-->`.

In general, the head of the rule has two arguments. The second one represents the phonological string of the node and is equal to the concatenation of the corresponding phonological arguments in the body of the rule. The first argument is a ProFIT term representing the node as a linguistic sign in the form of a ProFIT template. The arguments of the template unify with the corresponding arguments in the body of the rule and its last argument (the phonological argument) also unifies with the second argument of the head of the rule. The following example shows the rule for a polar interrogative:

(26)
```
s(S & @polar_intr(A,N,V,P),P)  --->  aux_node(A,Pa),
                                     np1(N,Pn),
                                     vp(V,Pv),
                                     {append(Pn,Pv,Temp)},
                                     {append(Pa,Temp,P)}.
```

2.5 Other Prolog rules: Quantifier Storage

We adopt an implementation of quantifier storage where stored quantifiers are passed up to the mother in a headed structure not from all daughters (like

in (Pollard and Sag 1994)), but only from the head daughter, as proposed by Pollard & Yoo (1998) and adopted by (Ginzburg and Sag 2001).[12] This proposal is formalised as the Store Amalgamation Constraint, according to which the STORE value of a verb will be the set union of the STORE values of the verb's ARG-ST members. Given that our grammar does not incorporate an ARG-ST feature, we implement this constraint as a Prolog rule which affects the valence features and which is called by the different VP projections in the DCG rules:

(27)
```
store_amalg(<sign &
            syn!store!Store &
            syn!loc!cat!subj!Subj &
            syn!loc!cat!comps!Comps)  :-
                              extract_store(Subj,SStore),
                              extract_store(Comps,CStore),
                              append(SStore,CStore,Store).
```

(28)
```
vp(@vp_tran(V,N,P),P)  --->  tran_node(V,Pv),
                             np1(N,Pn),
                             {append(Pv,Pn,P)},
                             {store_amalg(V)},
                             {c_params_amalg(V)}.
```

The STORE value of the verb is passed up from head daughter to mother according to a Store Inheritance Principle and eventually, at a clausal node, Quantifier Retrieval transfers the stored quantifiers to the mother's QUANTS list. There is an additional issue however: STORE members can be either parameters or quantifiers, given that quantifiers introduce a generalised quantifier in their STORE value and *wh* elements introduce a parameter.[13] That means that we need to check the STORE list to apply the Quantifier Retrieval to stored quantifiers only.

(29) a.
```
check_store(Phr & <phr &
            head_dtr!syn!store!Store)  :-
                                  store_list(_,Store,Phr).
```

b.
```
store_list(_,[],Phr) :- inheritance(Phr & syn!store![]).

store_list(P & <param,[P],Phr) :-
                        inheritance(Phr & syn!store![P]).

store_list(Q & <quantifier,[Q],Phr) :- retrieval(Phr).

store_list(Q & <quantifier,[Q|Rest],Phr) :-
                                   store_list(_,Rest,Phr).

inheritance(Phr & <phr &
            syn!store!Store &
            head_dtr!syn!store!Store).
```

[12] In this proposal STORE is a local feature, although our implementation does not adopt this modification.

[13] See (Ginzburg and Sag 2001) for justification.

```
c.  retrieval  (<phr & <declarative &
               syn!store!MStore &
               syn!loc!cont!soa!quants!Retrieval &
               head_dtr!syn!store!HStore) :-
                          append(MStore,Retrieval,HStore).
```

The Prolog clause in (29a) is then called at clausal level by the DCG rules:

```
(30)    s(@root_cl(S,P),P)  --->  s(S & @polar_intr(A,N,V,P),P),
                                  {check_store(S)}.
```

3 Fragments and the context resolution procedure of SHARDS

The main objective of SHARDS was to build a system for resolving elliptical constructions like fragments in dialog. As we have seen before, in early versions of the system phrasal utterances like sluicing and short answers were specified to constitute non-headed structures, the fragment being the non-head daughter.[14] In this new version of the grammar we analyse non-sentential utterances as instances of different subtypes of the headed phrase *head-only-phrase*.

To account for short answers and sluices, (Ginzburg and Sag 2001) posit the type *head-fragment-phrase*. The template in (31) is the implemented version of the constraint on such a type:

```
(31)    hd_frag_ph(H,P)  :=  <hd_frag_ph &
                             @head_only(H,P) &
                             syn!loc!cat!head!<verb &
                             syn!loc!cat!head!vform!<fin &
                             syn!loc!cat!subj![] &
                             syn!c_params!Params &
                             head_dtr!syn!loc!cat!head!<noun &
                             head_dtr!syn!loc!cont!index!Index &
                             head_dtr!syn!loc!cat!Cat &
                             head_dtr!syn!c_params!Params &
                             cxt!sal_utt![syn!c_params!Params &
                                     syn!loc!cat!Cat &
                                     syn!loc!cont!index!Index].
```

The current analysis is limited to NP fragments, so the HEAD value of the head daughter is restricted to be *nominal* (`<noun` in the example). The template ensures that the category of the head daughter is identical to the one specified by the contextually provided SAL-UTT. It also coindexes the head-daughter with the SAL-UTT, which has the effect of unifying the content of the former with a contextually provided content. The mother is specified to be of the same

[14] For a complete explanation of this proposal, see (Ginzburg, Gregory, and Lappin 2001).

category as finite verbs, which accounts for the fact that such phrases may function as stand-alone clauses.

As an example, we will consider the subtype *declarative-fragment-clause* used to analyse short answers. The template for the type *declarative-fragment-clause* does not specify its content. It only requires that phrases of this type be specified as IC+.

(32)
```
decl_frag_cl(H,P)  :=  @decl &
                       @hd_frag_ph(H,P) &
                       syn!loc!cat!head!ic!<plus.
```

The semantic content of the fragment is left to be recovered from context by the resolution procedure, which is called once a sentence has been parsed. The first step of the procedure consist in breaking down the AVM of each parsed sentence into two lists: the MDR (Mother-Daughter Relation) list and the QUD (Questions Under Discussion) list. The MDR list comprises a structured object called a Mother-Daughter Relation, whose three argument places are the mother AVM, the type of daughter-relation between the mother and the daughter, and the daughter AVM. The list structure is convenient for searching purposes and its transitive network encoding dominance relations will be used to re-assemble the AVM after resolution.

The QUD list contains the CONTENT values of all the clausal constituents of the parsed sentence. If this value is already a *question*, then it is left as it is; if it is a *proposition*, then it will be the content of a polar question.

In order to construct a context record, in the next step both the MDR list and the QUD list of the parsed sentence are stored in memory indexed by a counter for that sentence.

Once a particular ellipsis structure is identified, the MAX-QUD and SAL-UTT features are assigned values from the context record and, finally, the matrix AVM is recursively rebuilt.

(33)
```
ellip( AVM & <root_cl &
     head_dtr!@decl_frag_cl(syn!c_params!{[Ans|_]},_) &
     head_dtr!syn!loc!cont!soa!nucleus!Nucleus &
     cxt!max_qud!Qud &
     cxt!max_qud!params!{[Qu]} &
     cxt!max_qud!prop!soa!nucleus!Nucleus &
     cxt!sal_utt!Sal &
     cxt!sal_utt!syn!wh![Qu] &
     cxt!sal_utt!syn!c_params!{[Qu|_]},
     Quds, LU, AVM, 3 ) :-
                        member( Qud, Quds ),
                        const_of( Sal, LU ),
                        unify_if_possible( Ans, Qu ).
```

(33) shows one of the Prolog rules in the Resolution Procedure. The variable `Qud` is a member of the QUD list, while the variable `Sal` represents one of the

constituents of the last parsed utterance (LU). `Qud` is required to be the value of MAX-QUD, which in this case will be the content of some question in the context. The NUCLEUS of the head daughter (the fragment) is identified with the NUCLEUS of `Qud`. On the other hand, SAL-UTT has as its value one of the constituents of this question in context (which is restricted to be a *wh*-phrase by the declaration `syn!wh![Qu]`). The *parameter* introduced by this *wh*-phrase is identified with the value of the feature PARAMS in MAX-QUD.

4 Conclusions and future work

The grammar described here provides an implementation for the core constructions covered by (Ginzburg and Sag 2001), which confirms the computational viability of such theoretical framework. The implemented grammar is able to analyse a range of elliptical structures and dialogue fragments covering short answers and reprise and non-reprise sluices. However, it is designed to be a flexible base for further extensions covering other types of elliptical dialogue moves. In particular, it is being used in combination with the dialogue move engine of GoDiS (Larson, Ljunglof, Cooper, Engdahl, and Ericsson 2000) to implement the different forms and readings of clarification requests described in (Purver, Ginzburg, and Healey 2001).

References

Carpenter, B. (1992). *The logic of typed feature structures*. Cambridge Tractats in Theoretical Computer Science. Cambridge University Press.

Clocksin, W. and C. Mellish (1984). *Programming in Prolog*. Springer-Verlag.

Cooper, R., S. Larsson, J. Hieronymus, S. Ericsson, E. Engdahl, and P. Ljunglof (2001). Godis and questions under discussion. In *The TRINDI Book*. Gothenburg: University of Gothenburg. Available from http://www.ling.gu.se/research/projects/trindi.

Dalrymple, M., F. Pereira, and S. Shieber (1991). Ellipsis and higher order unification. *Linguistics and Philosophy* (14), 399–452.

Erbach, G. (1994). Multi-dimensional inheritance. In *Proceedings of KONVENS'94*, Vienna.

Erbach, G. (1995). ProFIT: Prolog with features, inheritance and templates. In *Proceedings of the Seventh European Conference of the ACL*, pp. 180–187.

Gazdar, G. and C. Mellish (1989). *Natural Language Processing*. Addison-Wesley Publishing Company.

Ginzburg, J. (1996). Interrogatives: Questions, facts, and dialogue. In S. Lappin (Ed.), *Handbook of Contemporary Semantic Theory*. Oxford: Blackwell.

Ginzburg, J. (2002). A semantics for interaction in dialogue. Forthcoming for CSLI Publications. Draft chapters available from: http://www.dcs.kcl.ac.uk/staff/ginzburg.

Ginzburg, J. and R. Cooper (2001). Clarification, ellipsis, and the nature of contextual updates. Under review for *Linguistics and Philosophy*.

Ginzburg, J., H. Gregory, and S. Lappin (2001). SHARDS: Fragment resolution in dialogue. In I. v. d. S. Harry Bunt and E. Thijsse (Eds.), *Proceedings of the Fourth International Workshop on Computational Semantics*. BIDIALOG.

Ginzburg, J. and I. Sag (2001). *Interrogative Investigations*. CSLI.

Gregory, H. (2001). A ProFIT grammar and resolution procedure for fragments in dialogue. Technical Report TR-01-03, Department of Computer Science, King's College London.

Gregory, H. and S. Lappin (1999). Antecedent contained ellipsis in HPSG. In G. Webelhuth, J. Koenig, and A. Kathol (Eds.), *Lexical and constructional aspects of linguistic explanation*, pp. 331–356. CSLI Publications.

Koenig, J.-P. (1999). *Lexical Relations*. CSLI Publications.

Lappin, S. and H. Gregory (1997). A computational model of ellipsis resolution. In *Proceedings of the Conference on Formal Gramamr*, Aix en Provence. ESSLLI.

Larson, S., P. Ljunglof, R. Cooper, E. Engdahl, and S. Ericsson (2000). GoDiS–an accommodating dialogue system. In *Proceedings of ANLP/NAACL-2000 Workshop on Conversational systems*.

Lascarides, A. and A. Copestake (1999). Default representation in constraint-based frmaeworks. *Computational Linguistics* (55), 55–105.

Pollard, C. and I. Sag (1994). *Head Drieven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications.

Purver, M., J. Ginzburg, and P. Healey (2001). On the means for clarification in dialogue. Presented at the SIGdial 2001 workshop on discourse and dialogue.

Sag, I. (1997). English relative clause constructions. *Journal of Linguistics* (33), 431–484.