# Parsing as Dynamic Interpretation

Harry Bunt and Ko van der Sloot

Institute for Language Technology and Artificial Intelligence ITK
P.O.Box 90153, 5000 LE Tilburg, The Netherlands
email: {bunt|sloot}@kub.nl

**Abstract**

In this paper we consider the merging of the language of feature structures with a formal logical language, and how the semantic definition of the resulting language can be used in parsing.

For the logical language we use the language EL, defined and implemented earlier for computational semantic purposes. To this language we add the basic constructions and operations of feature structures. The extended language we refer to as 'Generalized EL', or 'GEL'. The semantics of EL, and that of its extension GEL, is defined model-theoretically: for each construction of the language, a recursive rule describes how its value can be computed from the values of its constituents. Since GEL talks not only about semantic objects and their relations but also about syntactic concepts, GEL models are nonstandard in containing both kinds of entities.

Whereas phrase-structure rules are traditionally viewed procedurally, as recipes for building phrases, and a rule in the parsing-as-deduction is viewed declaratively, as a proposition which is true when the conditions for building the phrase are satisfied, a rule in GEL is best viewed as a proposition in Dynamic Semantics: it can be evaluated recursively, and evaluates not to true or false, but to the minimal change in the model, needed to make the proposition true.

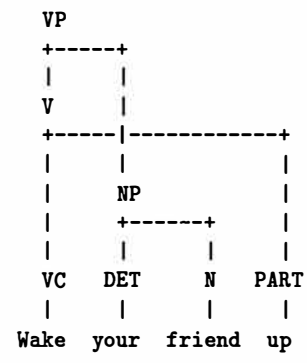
The viability of this idea has been demonstrated by a proof-of-concept implementation for DPSG chart parsing and an emulation of HPSG parsing in the STUF environment.

## 1 Discontinuous Phrase Structure Grammar

### 1.1 Introduction

DPSG, or 'Discontinuous Phrase Structure Grammar', has been developed over the years in the context of building natural-language understanding systems, such as the TENDUM dialogue system (Bunt et al., 1985). It has been applied in experimental systems both for parsing and for generation (see Bunt, 1987; 1991; Bunt — Thesingh — Van der Sloot, 1987). It has grown out of the tradition of augmented context-free grammars, where the augmentations in the case of DPSG are (1) conditions on features; (2) the formulation of semantic rules coupled to the syntactic rules in a rule-by-rule fashion.

The distinguishing property of DPSG is that its rules do not construct ordinary phrase-structure trees, but structures that allow crossing branches, called 'discontinuous trees'. Partly inspired by McCawley (1982), these constituents structures have been given a formal definition (Bunt, 1991). The use of discontinuous trees is intended to provide an intuitively appealing and computationally simple treatment of bounded discontinuities. An example is the following:




To generate such a structure DPSG uses rules which, disregarding the augmentations with feature conditions and semantics, look as follows,

where square brackets indicate 'internal context' elements of discontinuous constituents:

```
VP   --> V  + NP
NP   --> DET + N
V    --> VC + [NP] + PART
VC   --> wake
DET  --> your
N    --> friend
PART --> up
```

For further details see Bunt (1991), which also describes a chart parser for DPSG.

### 1.2 ID-LP separation in DPSG

The above rule format is nowadays obsolete, since it has been recognized that, by separating considerations of immediate dominance (ID) and linear precedence (LP), greater generality can be achieved. This was first demonstrated in GPSG (Gazdar et al., 1985), and has subsequently been exploited to an even greater degree in HPSG (Pollard — Sag, 1987; forthc.). It is, of course, also well recognized in GB theory.

The possibility in principle to apply the ID-LP separation in DPSG, which may be somewhat surprising since a discontinuous PS rule by definition would seem to say something about dominance as well as about precedence, was already indicated in Bunt (1991).

A discontinuous rewrite rule like the one used above — `V --> VC + [NP] + PART` — is equivalent to the following combination of dominance- and precedence constraints, where CD stands for context daughters:[1]

```
ID:   V --> {VC, PART}
CD:   {NP}
LP:   {VC < NP, NP < PART}
```

The dominance and precedence aspects of the rewrite rule have been separated here; one can subsequently consider the possibility to generalize the LP part and make it global for the entire grammar, as GPSG and HPSG are aiming to do, or for a part of the grammar, as is done in the latest version of DPSG.

In this example, the ID/LP formulation clearly has no advantages over the traditional formulation; such advantages can only be expected if the same dominance rule allows various precedences. This is precisely what is often the case with discontinuities; for example, in Dutch an intervening adverb (phrase) can be placed on a variety of positions.

### 1.3 Semantics in DPSG

In developing DPSG for implementing natural language fragments, we have chosen a rule format somewhat like that of Montague Grammar, where syntactic phrase-structure rules and feature conditions are coupled with semantic rules to build up semantic representations in parallel with syntactic structures.

For these semantic rules we have chosen the language EL ('Ensemble Language'), designed specifically for representing natural language semantics (see Bunt, 1985). This language combines typed lambda abstraction with concepts from Ensemble Theory, an extension of classical set theory developed for dealing with mass terms (Bunt, 1985; see also Lewis, 1992) and with data structures and operations known to be useful from computational semantics, such as lists, list projection, singleton, cartesian product, etc.

## 2 EL and feature languages

### 2.1 Feature structures as formal-language expressions

When we regard feature structures, as used for instance in HPSG, as expressions in a formal language, the question arises how this language relates to more traditional formal languages.

Feature structures have three basic ingredients, not commonly found in the formal languages of logic:

1. the pairing of an attribute and a value to form a feature;[2]
2. the combination of features into 'bundles';
3. the use of markers to indicate re-entrancy, or 'structure sharing'.

[1] The rules as given here are, again, extremely oversimplified copared to full-fledged DPSG-rules, which include local and global conditions on features, feature propagation constraints, and semantic composition rules.

[2] The term 'feature' is sometimes used in the literature for the combination of an attribute and a value, like `[case: genitive]` and sometimes for an attribute alone. In this paper we will use the term only in the former sense.

These three ingredients, plus the possibility to use features nested as attribute values, define the core of the language of feature structures. Additional constructions found in feature structures include lists, sets, negation, disjunction and occasionally others. Syntactically, the core of the language of feature structures can be defined as follows.

1. If $A$ is an attribute constant and $v$ a value constant or variable, then $[A : v]$ is a feature expression ("atomic feature specification").
2. If $A$ is an attribute constant and $f$ a feature expression, then $[A : f]$ is a feature expression ("complex feature specification").
3. If $f_1$ and $f_2$ are feature expressions, then $[f_1, f_2]$ is a feature expression ("feature bundle").
4. If $e$ is a feature expression and $i$ a structure sharing marker, then $(i : e)$ is a ("marked") feature expression.
5. If $A$ is an attribute constant and $i$ a structure sharing marker, then $[A : i]$ is a feature expression (with structure sharing).

The heart of the corresponding semantic definition is that of the atomic feature specification. We view feature specifications as descriptions of predicates; regarding, for example, the feature specification [gender: neuter] as denoting the predicate of having neuter gender. The semantics of feature bundles is nothing else than the conjunction of the corresponding predicates. The semantics of feature nesting is more complex. To formulate it properly, one must make a 3-way distinction between different kinds of feature attributes, exemplified by the attributes gender, head, and complement-daughters, respectively. Attributes of the first kind, like gender and case, take atomic values to form predicates; one noticeable point is that attributes of the other types cannot have atomic values. Attributes of the second type, like head and synsem, function merely as *labels*, allowing one to refer to certain feature complexes (like the head features) in grammatical principles like the Head Feature Principle. These attributes are semantically vacuous. Attributes of the third kind, like head-daughter and complement-daughters are again different, in that they describe dominance relations between words or phrases, rather than local grammatical properties. This is reflected in the fact that the values of these attributes must refer to words or phrases, i.e. they must be complex feature structures of which the phonology attributes are fully specified.[3] In a model-theoretic semantics, the difference comes out in the fact that these attributes denote relations among the words that inhabit the model and the phrases that can be formed from these words. Such a formalization, which we will not make explicit here, does bring out the asymmetry of HPSG that dominance relations are represented within signs, whereas precedence relations are not. Depending on how far one wants to go in formalizing the language in which the principles of HPSG are formulated, one may want to add the precedence relation to an enriched feature language.

There is one more important aspect of feature structures that must be dealt with: structure sharing. We do this by interpreting structure sharing markers, introduced in the above syntactic rules 4 and 5, as a special kind of variable, and defining the semantics of the expressions in the intuitively obvious way by means of the unification of the values of the subexpressions marked with the sane marker (see also below).

It is clearly possible to add the above five syntactic rules to those of a standard logical language, such as first-order predicate logic. To the rules defining the correct expressions of predicate logic we simply add the above rules and we stipulate that feature specifications can be used everywhere where a one-place predicate constant is allowed. The resulting language can be used to make statements about linguistic material. For instance, the following formula expresses that there is a singular noun with neuter gender in the sentence $S$ :

$$(\exists x) \quad \begin{array}{l} (\mathrm{IN}(x, S) \ \& \\ \mathrm{NOUN}(x) \ \& \\ [\mathrm{number} : \mathrm{sing}, \ \ \mathrm{gender} : \mathrm{neuter}](x)) \end{array}$$

In this way, feature structures can be added to the EL language. This opens the possibility to formulate DPSG rules entirely as expressions in Generalized Ensemble Language (GEL). Evaluating

[3] If empty nodes are allowed, the phonology attribute should obviously be allowed to have an undefined value - which would still make the value of the attribute fully specified.

such GEL-expressions then comes down to trying to prove statements about linguistic material - which is a way of thinking about parsing, as we know from the parsing-as-deduction paradigm.

## 2.2 Grammatical information in GEL

What do we have to add to EL in order to encode grammatical information? We will consider the case of grammatical information as expressed in Head-driven Phrase Structure Grammar to answer this question.

In HPSG, the sign is the informational unit. Signs make up the lexicon, and are used in the rules of HPSG. A sign is a complex feature structure with certain particular attributes and classes of values; a sign is thus readily expressed in GEL, once the relevant attribute and value constants have been added to the GEL vocabulary.

Rules in HPSG come in three forms: ID schemata, LP restrictions, and general principles (like the Subcategorization Principle and the Head Feature Principle). These rules are all statements about signs.

ID schemata describe, in terms of the various immediate dominance relations distinguished in HPSG (head - head daughter, head - complement daughter, head - adjunct daughter, head - marker, head - filler) the configurations of signs that are permitted. Since the various dominance relations are treated in HPSG as feature attributes, and dominance structures are encoded in feature structures, this means altogether that ID schemata are actually constraints on admissible phrasal signs. Due to the logical power of EL, these constraints are readily expressed in GEL, once we are able to represent signs.

LP restrictions are formally different from ID schemata, since they describe constraints on signs with linear precedence relations, which are not treated in HPSG as feature attributes, and not part of the information in signs, but belong to the metalanguage. For the re-expression of LP restrictions in GEL we have three options:

1. we follow the HPSG strategy, leaving the linear precedence relation at the metalanguage and thus outside the feature formalism;
2. we add a predicate constant denoting the LP relation to GEL, and formalize what the theory says on linear precedence;
3. we introduce a feature attribute 'right neighbour', comparable to HPSG's daughter attributes and treat LP relations as parts of signs.

To make our implementation close to other existing implementations (in particular to the STUF implementation of HPSG, discussed below), we take the first option, although theoretically one could do better.

General principles, finally, describe how the values of certain feature attributes depend on those of the daughters. They can be seen as describing the propagation of feature values upon phrase building. These dependencies can be described in a first-order language, and are thus easily expressed in GEL.

The crucial innovative feature needed in GEL is the possibility to have shared subexpressions within an expression. To this end a new construct has been added which employs special variables ('unifiable variable'), binding shared subexpressions. These variables are denoted by an unique name which starts with '@'. A simple example of a GEL expression with shared subexpressions is:

```
[synsem: [local    : name: @1,
          contents: lambda(_x,
                           application(@1,_x)
                           )]]
```

This could be a template for a propername entry in a lexicon (see below on the definition and use of templates). Whenever @1 is instantiated, this information is shared between the name and the semantic contents. There are no restrictions on the kind of value that can be assigned to a unifiable variable; any GEL expression will do.

An important question is, of course, how this assignment is done. To this end the constructions 'unifiable' and 'unify' are introduced, the use of which will be illustrated below. A unify construction takes any number of arguments, tries to unify them in the usual way (see Shieber et al., 1983), and delivers the resulting GEL expression

or NULL, if the unification fails. Unifiability is defined using unify: unifiable($a_1, \ldots, a_n$) returns TRUE if unify($a_1, \ldots, a_n$) is succesful, and FALSE otherwise.

# 3 GEL semantics and evaluation

The semantics of EL, and that of its extension GEL, is defined model-theoretically: for each construction of the language there is a rule describing how its value can be computed from the values of its constituent expressions, down to the atomic constituents. Here the recursion ends, and the values of the atomic constituents are looked up in the model, which is a structured specification of these values.

The semantics of GEL is defined model-theoretically, in the same way as that of EL, but since GEL talks not only about 'semantic' objects but also about syntactic objects and their relations, GEL models contain both linguistic and nonlinguistic entities.

When defining a formal semantics for the GEL extension with feature structures, the first thing to consider is the semantics of a simple feature specification. We view a feature attribute like gender as a mathematical function, with entities like feminine as values. This captures the idea that an attribute has a value, and a unique one.

Viewing feature attributes as functions raises the question to what objects these functions apply: what is their domain? We think this is fairly obvious: the domain consists of words and phrases. A feature specification then amounts to a predicate, which can be used to express a syntactic property of a word or phrase. The semantics of a predicate being a set (or a characteristic function), a GEL expression like [gender: fem] receives as its interpretation the set of those words and phrases that have feminin gender.[4]

To formally interpret a feature specification, we apparently need a model which includes words as the 'individuals' to which feature attributes apply, and which also includes syntactic concepts like feminin, interrogative and mass as individual objects that may occur as feature values.

A model for the GEL sublanguage of feature structures is thus a triple:

$$M = < W, \{AV_1, \ldots, AV_k\}, F >$$

where $W$ is a countable set of words, $\{AV_1, .., AV_k\}$ is a finite collection of finite sets of objects called 'atomic feature values', and $F$ is a function assigning interpretations to constants. $F$ assigns atomic feature values to value constants; to an attribute constant $A$, $F$ assigns a function from $W$ to some value set $AV_i$. The sets $AV_i$ are assumed to be disjoint.

If $\mathcal{V}$ is the recursive evaluation function assigning interpretations to GEL expressions, we get, in first approximation, the following semantics for a simple feature specification:

$$\mathcal{V}([A : v]) = \{w \text{ in } P(W) | F(A)(w) = \mathcal{V}(v)\};$$
$$\text{if } v \text{ is atomic, then } \mathcal{V}(v) = F(v)$$

where $P(W)$ denotes the set of all phrases (words and word sequences) that can be formed from $W$.

# 4 Parsing as dynamic interpretation

Phrase-structure rules are traditionally viewed procedurally, as recipes for building phrases. In the parsing-as-deduction approach a rule is viewed declaratively, as a proposition which is true when the conditions for building the phrase are satisfied. A rule expressed in GEL is best viewed as a proposition in Dynamic Semantics: it can be evaluated recursively, and evaluates not to true or false, but to the minimal change in the model, needed to make the proposition true.

The viability of this idea has been demonstrated by a proof-of-concept implementation for DPSG chart parsing and an emulation of the STUF implementation of HPSG. We describe these in the following sections.

## 4.1 DPSG rules in GEL

To illustrate the use of GEL in the implementation of DPSG, we consider a (simplified) DPSG rule, which combines a central determiner and a

[4] Treating feature attributes as functions seems to us intuitively more satisfying than treating them as atomic entities, as Gazdar and Pullum (1987) have proposed.

```
NPCENTRE_2
ID rule   : NPCENTRE --> {a:CENTRALDET, b:NOM };
CD rule   : {};
LP constr : {a < b};
GEL rule  : conditional(
                  conjunction( eq( Form_of( a ), Form_of( b ) ),
                               eq( Gender_of( a ), Gender_of( b ) ),
                               memberof( { <Mass>, <Ground>, <Coll> },
                                         Form_of(a) ) ),
                  unify( Head_Feature_Instantiation( b ),
                         Form(    Form_of( b ) ),
                         Gender(  Gender_of( b ) ),
                         Person(  Person_of( b ) ),
                         Content( partselection(
                           Content_of( b), lambda(_x,
                           application( Content_of(a), _x ))) )),
                  undef );
```

nominal into an 'npcentre'. To facilitate the reuse of the parsing strategy implemented for the original DPSG format, the categorial ID-parts as well as the CD- and LP-parts of the rule have been kept separate from the rule part where the real work is done; this part, specifying the conditions and actions on features as well the semantic composition, is expressed in GEL. The GEL expression is of the form `conditional(A, B, C)`; according to the EL semantics underlying GEL, the interpretation of such an expression gives the value of `B` if `A` evaluates to `TRUE`, and that of `C` otherwise. In the conjunction describing the conditions for succesfully applying the rule, the equality relation of EL ('`eq`') is used to test form and gender agreement of the constituents `a` and `b`. In the same way we test whether the form of constituent a is a member of an enumerated set, using the relation '`memberof`'.

When the parameters `a` and `b` in this rule are instantiated by feature structures representing constituents of category `CENTRALDET` and `NOM` (the ID-condition), where the central determiner immediately (CD-condition) precedes the nominal (LP-condition), then the GEL rule says that if the constraints on form and gender are satisfied, these constituents are the daughters of an NP centre whose head features, form feature, gender feature and person feature are inherited (through unification) from the nominal constituent, and whose semantic content is constructed compositionally from the contents of the daughters.[5] Dynamic interpretation of the rule, using the chart as a model, has the effect of *creating* and `NPCENTRE` node with these properties.

This approach has been implemented succesfully by extending the implemented machinery for EL evaluation with the GEL augmentations (basically, the representation of complex feature structures and the operations on those), and merging this with the DPSG parser described in Bunt (1991), originally developed by van der Sloot (1990).

## 4.2 HPSG/STUF emulated in GEL

### 4.2.1 The Stuttgart Type Unification Formallism (STUF)

A second proof-of-concept implementation of the idea of parsing as dynamic interpretation has been made in the form of an emulation of the parser and grammar development environment called STUF (Stuttgart Type Unification Formalism), originally developed in the LILOG project (Herzog et al., 1986).

STUF is a descendant of PATR-II (Shieber et al., 1983), providing a formalism and a software environment for writing grammars of various

[5] The semantic part says, more specifically, that the NP centre denotes the union of those parts of the denotation of the (mass) nominal's denotation that satisfy the predicate denoted by the cenral determiner. For more explanation see Bunt, 1985.

kinds. STUF has a much richer language for specifying feature structures, based on Kasper and Rounds' feature logic (Kasper — Rounds, 1986). In STUF it is possible to directly specify complex embedded structures, including disjunctive and negative information. The core of the STUF language is formed by the following definition of the syntax of feature terms:[6]

```
atom              atomic value or template
_X                variable
attribute: S      feature selection
templ(S1,..,Sn)   parametrized template
[S1 S2 ..]        conjunction
{S1 S2 ..}        disjunction
not S             negation
```

Variables are used to describe structure sharing. They have the same interpretation as structure indices in the matrix notation used above.

Templates are named complex feature terms, used to organize information more compactly. An example is:

```
intransitive_verb :=
     [syntax [value: [ syntax: s
                       semantics: _X]
              direction: left
              argument:  syntax: np]
      semantics: _X]
```

Templates may be functional, having parameters which are substituted by actual values when applied. A simple example is:

```
invert_boolean(plus)  := minus.
invert_boolean(minus) := plus.
```

Lexical entries are simply feature term definitions where the name is interpreted as a word that may appear in an input sentence. Templates make these definitions very simple. Examples:

```
John := [ syntax: np ].
goes := intransitive_verb.
```

The STUF implementation of HPSG is not entirely faithful to the theory, in that (1) there is in fact no ID/LP separation in STUF rules, which means that every ID schema has as many LP variants as the LP constraints allow; and (2) the General Principles are instantiated for every ID schema.

The following elements taken from an implementation of a fragment of English in the PLUS system (Black et al., 1991; Rentier, 1993) illustrate the actual use of STUF. In the next subsection we will illustrate the GEL emulation of STUF by describing the corresponding GEL structures.

**1.** Some functional templates:

```
head(X)   := synsem:local:head: X.
major(X)  := head(maj: X).
inv (X)   := head(maj: X).

comps(X)  := synsem:local:comps: X.
comps0    := comps(undef).

sat       := subj0, comps0.
ssat      := subj0, comps0, mod0.
unsat     := comps(def).

subj(X)   := synsem:local:subj: X.
subj0     := subj(undef).

mod(X)    := synsem:local:mod: X.
mod0      := mod(undef).
```

**2.** Some nonfunctional templates:

```
nominal   := major(n), mod0.
nphrase   := nominal, ssat.
```

**3.** Some General Principles and auxiliary templates:

```
'HeadInherit'     := mother: head(X),
                     h_dtr: head(X).
'ContentInherit'  := mother: content(X),
                     h_dtr: content(X).
'CompsInherit'    := mother: comps(X),
                     h_dtr: comps(X).
'GapInherit'      := mother: gap(X),
                     ( ( nonhead: gap0,
                         h_dtr: gap(X) );
                       ( h_dtr: gap0,
                         nonhead: gap(X) ) ).
'BindInherit'     := mother: bind(X),
                     ( ( nonhead: bind0,
                         h_dtr: bind(X) );
                       ( h_dtr: bind0,
                         nonhead: bind(X) ) ).
```

[6] We omit the use of semantic subsorts and paths here. For the original definitions see Dörre — Seiffert, 1991 and Dörre — Raasch, 1991.

```
'NonLocalInherit' :='GapInherit',
                    'BindInherit'.
'ModInherit'      := mother: mod( _X ),
                     h_dtr:  mod( _X ).
'ModCombine'      := mother:( mod0,
                              content(_X),
                              context(_Z) ),
                     h_dtr:    _Y,
                     nonhead:(sat,
                              gap0,
                              mod(_Y),
                              content(_X),
                              context(_Z) ).
'Complementation' := 'HeadInherit',
                     'ModInherit',
                     'ContentInherit',
                     'NonLocalInherit'.
'Adjunction'      := 'HeadInherit',
                     'CompsInherit',
                     'NonLocalInherit',
                     'ModCombine'.
```

**4.** Some rules (ID schemata with instantiated LP restrictions and General Principles):

```
'RightComplementation' :=
    mother -> h_dtr, nonhead --
    'Complementation',
    nonhead: ( major( not( d ) ) ).
'LeftComplementation'  :=
    mother -> nonhead, h_dtr --
    'Complementation',
    nonhead: ( major(d) ).
'LeftAdjunction'       :=
    mother -> nonhead, h_dtr --
    'Adjunction'.
```

**5.** Some lexical entries (slightly simplified):

```
car :=  nphrase,
        form(N),
        pers('3rd'),
        parm(P),
        restr1((rel:'CAR', arg0: P)).
who :=  nphrase,
        det( wh ),
        pers('3rd'),
        gend(not(neut)),
        case(not(gen)),
        gap0,
        parm(P),
        restr1((rel:'WHO', arg0: P)).
```

### 4.2.2 STUF emulated in GEL.

To illustrate the emulation of STUF in GEL, we describe the GEL counterparts of the STUF functional and other templates, ID rules instantiated for LP rules and for General Principles, and a few lexical entries.

**0.** We first define an empty sign to get unifications going:

```
SIGN := synsem:
          [ local: [ head: [ maj  : @,
                             aux  : @,
                             inv  : @,
                             case : @,
                             form : @ ],
                     comps: @,
                     subj:  @,
                     mod:   @ ],
            nonlocal:
                    [ bind: @,
                      gap:  @ ],
            cont:
                    [ parm :
                            [ pers: @,
                              gend: @,
                              num : @ ],
                      restr: @ ]].
```

The symbol '@' is to be considered as an anonymous variable; no two occurrences are taken as identical.

**1.** Some functional templates

```
Head( @X )      := synsem:local:head: @X.
Major( @X )     := Head( maj:   @X ).

Inv( @X )       := Head( inv:   @X ).
Inv0            := Inv( undef ).

Comps( @X )     := synsem:local:comps: @X.
Comps0          := Comps( <> ).

Sat             := unify( SIGN,
                          Subj0,
                          Comps0 ).
Ssat            := unify( Sat, Mod0 ).
Unsat           := Comps( Def ).

Subj( @X )      := synsem:local:subj:  @X.
Subj0           := Subj( undef ).

Mod( @X )       := synsem:local:mod: @X.
Mod0            := Mod( undef ).
```

**2.** Some nonfunctional templates:

```
Nominal := unify( SIGN, Major( N ), Inv0,
                  Mod0, Subj0 ).
Nphrase := unify( Nominal, Ssat ).
```

**3.** Some General Principles:

```
HeadInherit( @HD ) :=
  conditional( unifiable( @HD, Head(@X) ),
               Head(@X),
               undef ).

ContentInherit( @HD ) :=
  conditional( unifiable( @HD, Content(@X) ),
               Content(@X),
               undef ).

CompsInherit( @HD ) :=
  conditional( unifiable( @HD, Comps(@X) ),
               Comps(@X),
               undef ).

GapInherit(@HD,@N) :=
        conditional(
          unifiable( @N, Gap0 ),
          conditional(
            unifiable( @HD, Gap( @G1 ) ),
            Gap( @G1 ),
            undef ),
          conditional(
            unifiable( @HD, Gap0 ),
            conditional(
              unifiable( @N, Gap( @G2 ) ),
              Gap( @G2 ),
              undef ),
            undef ) ).

BindInherit(@HD,@N) :=
        conditional(
          unifiable( @N, Bind0 ),
          conditional(
            unifiable( @HD, Bind( @G1 ) ),
            Bind( @G1 ),
            undef ),
          conditional(
            unifiable( @HD, Bind0 ),
            conditional(
              unifiable( @N, Bind( @G2 ) ),
              Bind( @G2 ),
              undef ),
            undef ) ).

NonLocalInherit( @HD, @N ) :=
    unify(
        SIGN,
        GapInherit( @HD, @N ),
        BindInherit( @HD, @N ) ).

ModInherit(@HD) :=
  conditional( unifiable( @HD, Mod( @X ) ),
               Mod( @X ),
               undef ).

ModCombine(@HD,@N) :=
        conditional(
          unifiable( @N, Sat,
                     Gap0,
                     Mod(@HD),
                     Content( @X ),
                     Context( @Z ) ),
          unify( SIGN, Mod0,
                       Content( @X ),
                       Context( @Z ) ),
          undef ).

Complementation( @HD, @N ) :=
     unify(
        SIGN,
        HeadInherit( @HD ),
        ModInherit( @HD ),
        ContentInherit( @HD ),
        NonLocalInherit( @HD, @N ) ).

Adjunction( @HD, @N ) :=
    unify(
        SIGN,
        HeadInherit( @HD ),
        SubjInherit( @HD ),
        CompsInherit( @HD ),
        ModCombine( @HD ,@N ) ).
```

**4.** Some rules (ID schemata with instantiated LP restrictions and General Principles):

```
RightComplementation(@Head,@Comp) :=
    conditional(
      unifiable( @Comp, Major( notu( D ) ) ),
      Complementation( @Head, @Comp ),
      undef ).

LeftComplementation( @Comp, @Head ) :=
    conditional(
      unifiable( @Comp, Major( D ) ),
      Complementation( @Head, @Comp ),
      undef ).

LeftAdjunction( @Adj, @Head ) :=
    Adjunction( @Head, @Adj ).
```

**5.** Some lexical entries:

```
car := unify(Nphrase,
             Form(N),
             Pers(3rd),
             Parm(@1),
             Restr([rel:Car,
                    argzero:@1])).

who := unify(Nphrase,
             Det(Wh),
             Pers(3rd),
             Gend(notu(Neut)),
             Case(notu(Gen)),
             GapO,
             Parm(@1),
             Restr([rel:Who,
                    argzero:@1])).
```

Comparing the STUF and GEL descriptions, we see that, once the necessary functional templates and other auxiliary structures have been put in place, a relatively simply pattern of relations emerges. It my be noted that the GEL emulation has a clearer and more explicit representation of HPSG's general principles than the original STUF implementation, though the STUF representation is more compact. As a result, the fact that the theory's General Principles are instantiated in every GEL rule (just like in STUF) is hardly a drawback, although it is not in accordance with the theory. The principles are explicitly available in the implementation, ready for inspection and modification. Note also that the lexical representations in STUF and GEL are identical except for minor notational details; this is very important in practice, since all that an HPSG grammar writer is concerned with is the specification of lexical elements. In fact, the correspondence between lexical items in STUF and GEL is so straightforward that an automatic conversion from one format to the other would be possible.

### 4.3 Rule application as evaluation

Since the LP restrictions and General Principles are instantiated in each ID-rule, application of STUF HPSG rules simply comes down to the application of these ID rules; there are no additional checks. Therefore, standard parsing procedures can be applied. We have implemented a simple chart parser, using the matrix-driven parsing strategy described in Bunt (1991),[7] which applies the ID rules by invoking the implemented 'GEL machine' to evaluate the GEL rule.

## 5 Conclusions and future work

Where STUF and similar formalisms and implementations developed in recent years, such as TFS, CUF and PLEUK, is an advancement compared to PATR-II because of its more powerful language for expressing linguistic information, GEL offers further extended possibilities to formalize grammatical information and represent it in a computationally attractive form. Not only ID-schemata and lexical entries can be given a formal representation, but LP-restrictions and General Principles as well. At the same time, GEL retains the advantages of a fully declarative representation and of integrated representation of syntactic and semantic information.

Further work will make clear how attractive this approach can be for building efficient parsers and generators that work directly on the constraint-based representation of rules, rather than by internally compiling them first into more traditional formats.

[7] The implemented parser is simplified in that the particular provisions for dealing with discontinuous constituents have for the moment been left out. The implementation, done in C, is the work of Ko van der Sloot.

# References


Black, W. et al. (1991) "A Pragmatics-vased Language Undrstanding System". In: *Information Processing Systems and Software: Results of Selected Projects.* EC, Esprit, Brussels.

Bunt, H. (1985) *Mass terms and model-theoretic semantics.* Cambridge University Press, Cambridge, England.

Bunt, H. (1987) "Utterance generation from semantic representation augmented with pragmatic information". In G. Kempen (ed.) *Natural language generation.* Kluwer/Nijhoff, The Hague.

Bunt, H. (1991) "Parsing with Discontinuous Phrase Structure Grammar". In M. Tomita (ed.) *Current Issues in Parsing Technology.* Kluwer, Boston.

Bunt, H. — J. Thesingh — K. van der Sloot (1987) "Discontinuities in trees, rules and parsing". In *Proceedings of the Third Conference of the European Chapter of ACL*, Copenhagen.

Dörre, J. — I. Raasch, (1991) *The Stuttgart Type Unification Formalism - User Manual.* IWBS Report 168, IBM Scientific Center, Stuttgart.

Dörre, J. — R. Seiffert (1991) *Sorted Feature Terms and Relational Dependencies.* IWBS Report 153, IBM Scientific Center, Stuttgart.

Gazdar, G. — E. Klein. — G. Pullum — I. Sag (1985) *Generalized Phrase-Structure Grammar.* Harvard University Press, Cambridge, MA.

Gazdar, G. — G. Pullum (1987) *A Logic or Category Definition.* Cognitive Science Research Paper CSRP 072, University of Sussex.

Herzog, O. et al. (1986) *LILOG - Linguistic and logic methods for the computational understanding of German.* LILOG Report 1b, IBM Scientific Center, Stuttgart.

Kasper, R. — W. Rounds (1986) "A logical semantics for feature structures". In: *Proceedings of the 24th Annual Meeting of the ACL.* Columbia University, New York.

Lewis, D. (1992) *Parts of Classes.* Basil Blackwell, Oxford and Cambridge, MA.

J. McCawley (1982) "Parentheticals and Discontinuous Constituent Structure". *Linguistic Inquiry*, 13, 91-106

Pollard, C. — I. Sag (1987) *Information-based Syntax and Semantics.* Vol I, Fundamentals. CSLI Lecture Notes 13, Center for the Study of Language and Information, Stanford.

Pollard, C. — I. Sag (forthc.) *Information-based Syntax and Semantics.* Vol II. Preliminary version dstributed as 'Topics in Constraint-based Syntactic Theory', Universität des Saarlandes, Saarbrücken, 1992.

Rentier, G. (1993) "The PLUS Grammar". *PLUS deliverable D3.2*, ITK, Tilburg, May 1993.

Shieber, S. — H. Uszkoreit — F. Pereira — J. Robinso — M. Tyson (1983) "The formalism and implementation of PATR-II". In: J. Bresnan (ed.) *Research on Interactive Acquisition and Use of Knowledge.* SRI International, Artificial Intelligence Center, Menlo Park, Cal.

Sloot, K. van der (1990) "The TENDUM 2.7 parsing algorithm for DPSG". *ITK Research Memo*, ITK, Tilburg.

Tomita, M.(ed.) (1991) *Current Issues in Parsing Technology.* Kluwer, Boston.