[From: *Georgetown University Round Table on Languages and Linguistics 1989*, James E.Alatis, ed.]

Using unification grammars for analysis and synthesis

Margaret King
ISSCO and ETI
Université de Génève

Foreword. Despite the single authorship of the paper, the work reported here is that of quite a large group of people; since the author's role was relatively limited, it seems only appropriate to open this paper with a list of them. Rod Johnson was primarily responsible for the overall shape of the system, and he, Mike Rosner, Dominique Petitpierre and John Carroll are responsible for the development of the software tools described. The linguistic descriptions of German and of French are the work of Susan Warwick, C.J. Rupp, Graham Russell, and Thérèse Torris. Supplementary work on dictionary coding for German, French, and Italian has been contributed by Riccardo Boschetti, Kirsten Falkedal, Nora Nadjarian, Pascale Dhoop, Sandra Manzi and Lucia Tovena.

Despite the impression of disparate areas of responsibility suggested by the above, it should be emphasized that one of the main principles underlying this work is that, at the design level, there should be a strong mutual influence between software specification and the expressive needs of linguists. Hence, we have adopted a philosophy of rapid prototyping whereby each version of the system is prototyped, is used in linguistic work, and the experience thus gained influences the following version.

This does not mean, of course, that the linguist should have to worry about computational considerations while he is actually constructing a linguistic description: on the contrary, we take as a further principle that the language used for writing linguistic descriptions should be natural for use by a linguist, that is, it should resemble closely the kinds of formalism with which he is familiar from work in theoretical linguistics, and that he should not have to concern himself with how the software of the system applies the linguistic description in order to fulfill a particular task. In particular, he should not have to concern himself with procedural questions, such as the order in which rules are applied or in which computational procedures are executed.

Some background. Before going into any more detail about the prototype we are currently working on at ISSCO, it will be useful to explain a little about the background of the work. Some eighteen months ago, in the autumn of 1987, a Swiss association (Swisstra) concerned with establishing an expertise in machine translation within Switzerland, gave us a mandate to investigate methodologies for the evaluation of machine translation systems. Influenced partly by work done at Hewlett Packard on establishing test suites for natural

language analyzers working on English, partly by techniques used in software engineering for proving the correctness of compilers, we set out to investigate the feasibility of setting up benchmark tests to be used in determining the coverage of a machine translation system. In order to experiment with tests of this type, we needed access to a machine translation system whose functioning we understood well; on the grounds that those who construct something best understand how what has been constructed works, we then decided to construct a small research prototype ourselves.

Since the prototype (known by the rather inelegant name of UD) is part of the work on evaluation, and is not intended as a commercial prototype, we have been able to allow ourselves to make use of relatively untried technology. This primarily manifests itself through the use of unification grammars both for analysis and for synthesis, in an attempt to ensure that the same linguistic description can be used both for analysis and for synthesis, in the influence of recent work in theoretical linguistics on the way the linguistic descriptions are written (much more weight is given to the lexicon than is traditionally the case in machine translation systems, for example), and in the reflection of work in situation semantics to be found in some of the linguistic representations. It would be impossible to discuss all of this here, where I shall mostly concentrate on the language used for linguistic descriptions. The interested reader is referred to Johnson and Rosner 1989, Rupp 1989, and Carroll et al. forthcoming, for discussion of some of the other questions.

Unification-based approaches to grammar. A basic property of unification-based formalisms is the use of feature structures as their informational domain. Different formalisms differ in the way feature structures are defined and represented: cf., for example, the 'terms' of DCGs, the directed acyclic graphs of PATR-II, GPSG's feature bundles, FUG's 'functional structures', and LFG's 'f-structures'. In UD, a feature structure is a set of attribute-value pairs, where a value may be atomic, as in:

```
| number = singular |
```

may be a path, as in

```
| agreement: number = singular |
```

or may be complex. A complex value may consist of a set of attributes and their associated values, as in:

```
| agreement:  number = singular  |
|             person = 3         |
```

or may be a reentrant feature structure, i.e. a structure where two or more attributes share the same value, as in:

```
| head:    category = vp                                  |
|          agreement:  <#1>  | number = singular  |      |
|                            | person = 3         |      |
|                                                         |
| subject:   | category = np  |                           |
```

```
|              | agreement => #1      |                |
```

Feature structures are combined by unification. The definition of unification used by UD is that two feature structures may be unified providing that they are not mutually inconsistent, the result being a third feature structure which contains all the information contained in each of the two original feature structures. Thus, given three feature structures:

```
A = |  category = adjective              |
    |  agreement: case = dative          |


B = |   category = noun                  |
    |   agreement: case = dative         |
    |                  number = singular |
    |                   gender = masculine |


C = | agreement: case = dative           |
    |                  number = singular |
    |                 gender = masculine |
```

then

```
A |_| C =  | category = adjective                |
           |  agreement: case = dative          |
           |                  number = singular |
           |                  gender = masculine |

B |_| C =  | category = noun                     |
           | agreement: case = dative            |
           |                  number = singular  |
           |                  gender = masculine |

A |_| B = failure
```

The grammar rules of UD take the form of classical phrase-structure rules annotated by a set of equations which express the constraints to be satisfied by the feature structures referred to in the two sides of the rule. Thus, the following is a conventional s - > np vp rule, which additionally says that the np should be interpreted as the subject of the vp, and that there is agreement between the subject and the verb:

```
s -> np vp

    < s head > = < vp head >
    < s head subject > = np head >
    < np agreement > = < vp agreement >
```

The appendix contains a complete grammar for a fragment of English, adapted by Johnson (1988) from an example given in Shieber (1986).

Special characteristics of UD. So far, what has been described is a fairly conventional unification-based grammar. UD offers a number of extensions, including the use of lists and Prolog-like terms as data types. But perhaps the most interesting extension is the possibility to use 'relational abstractions'. Essentially, a relational abstraction allows the linguist to express a generalization once, and to refer to it as often as he wishes in the constraint equations. (A simple concrete example would be to express agreement constraints, instead of doing so explicitly as in the rule given above.) The power of this facility is most obvious, perhaps, at the level of the lexicon, where it allows the work of defining lexical information to be separated from the laborious work of coding the actual entries: the linguist constructing the grammar defines the abstractions, the dictionary coder needs only to know the name of the abstraction and how to use it. Thus, a lexical entry for the German verb *sehen* will appear as:

```
sehen    *    v/adj

     { - prefix }    !Pref (none)
     !Nonrefl        !Loctype ([])
     !Subcat (np (nom),  np (acc), vp (bse),  sor )
```

where all the elements preceded by ! refer to relational abstractions. The same entry, when fully expanded, gives a great deal of detailed linguistic information:

```
cat = v
form = sehen
gin = #57
gout = #57
head: morph:  sep = none
        sem:  desc:  arg:   1 = #52
                            2 = #41
                     loc: cond =  [ overlap ( <31> type = loc
                                              val = loc.12
                                    <34> type =  loc
                                              val = loc.11)
                                         overlap ( #31
                                              <28> type = loc ) ]
                          ind => #34
                     pol = 1
                     rel = sehen
        disc:   loc:   ind => #28
        ref:   loc:   ind => #31
 syn:  infl: agr:  <19> num   = pl
                        pers = 1/3
             tns:  fin    = pres
                   fut    = no
                   perf = no
        mood:  = subjunc
        perf-aux = none
        pref = none
        refl = none
        subj:  <45>  sem: desc -> #52
                     syn:  infl:  agr => #19
```

```
                                    case = nom
                                nform = norm
        vform = fin
        voice = active
        vtype = norm
null = no
subcat =  [ <22> cat = np
                 head => #45
                 subcat =  []
             <23> cat = np
                  head: sem: desc -> #41
                  syn:  infl: case = acc
                        nform = norm ]
```

More detail about the dictionaries used with UD can be found in Warwick (1986).

Current status. At the moment, substantial linguistic descriptions for German and French have been constructed and tested for analysis. (The German lexicon, which is the larger, contains around 2,400 items, of which slightly less than half are verb entries). Our main current preoccupation is synthesis: a first version of a synthesizer which uses the same linguistic descriptions as as analysis has recently been completed, and is now being used to investigate what constraints are necessary for synthesis to run satisfactorily. Transfer will be added, we hope, later this year, in the light of extensive experimentation with the parallel linguistic descriptions.

Appendix: A simple UD grammar.

```
##Taken from Johnson (1988), adapted from Shieber (1988).

# Declare

    Category = cat


# Grammar


# Rule sentence formation

    $ -> NP VP
        < * cat> = s
        < NP cat> = np
        < VP cat> = vp
        < * head form> = finite
        < VP subcat> = [NP]
        < * head> = < VP head>


# Rule trivial verb phrase

    VP -> V
```

```
        < VP cat> = vp
        < V cat> = v
        < VP subcat> = < V subcat>
        < VP head> = < V head>


# Rule complements

    VP -> Fun Arg
        < VP cat> = < Fun cat> = vp
        < Arg cat> = np/vp
        < Fun subcat> = [ Arg | < Vp subcat> ]
        < VP head> = < Fun head>


# Lexicon root

sleeps      v  < * head form> = finite
               < * subcat> = [ NP ]
               < NP cat> = np
               !Agree (NP, 3, singular)
               < * head sem pred> = sleep
               !Arg ( 1, NP)

sleep       v  < * head form> = finite
               < * subcat> = [ NP ]
               < NP cat> = np
               !Agree (NP, _, plural)
               < * head sem pred> = sleep
               !Arg ( 1, NP)

sleep       v  < * head form> = nonfinite
               < * subcat> = [ NP ]
               < NP cat> = np
               < * head sem pred> = sleep
               !Arg ( 1, NP)

storms      v  < * head form> = finite
               < * subcat > = [ Obj, Subj ]
               < Obj cat> = < Subj cat> = np
               !Agree ( Subj, 3, singular)
               < * head sem pred> = storm
               !Arg ( 1, Subj)
               !Arg ( 2, Obj)

storm       v  < * head form> = finite
               < * subcat > = [ Obj, Subj ]
               < Obj cat> = < Subj cat> = np
               !Agree ( Subj, _, plural)
               < * head sem pred> = storm
               !Arg ( 1, Subj)
               !Arg ( 2, Obj)

storm       v  < * head form> = nonfinite
               < * subcat > = [ Obj, Subj ]
               < Obj cat> = < Subj cat> = np
               < * head sem pred> = storm
```

```
                !Arg ( 1, Subj)
                !Arg ( 2, Obj)

Stormed     v  < * head form> = pastpart
                < * subcat > = [ Obj, Subj ]
                < Obj cat> = < Subj cat> = np
                < * head sem pred> = storm
                !Arg ( 1. Subj)
                !Arg ( 2, Obj)

has             v  < * head form> = finite
                < * subcat> = [ Vcomp, Subj ]
                < Vcomp cat> = vp
                < Vcomp head form> = pastpart
                < Vcomp subcat> = [ Subj ]
                < Subj cat> = np
                !Agree ( Subj, 3, singular)
                < * head sem pred> = perfective
                !Arg ( 1, Vcomp)

have          v  < * head form> = finite
                < * subcat> = [ Vcomp, Subj ]
                < Vcomp cat> = vp
                < Vcomp head form> = pastpart
                < Vcomp subcat> = [ Subj ]
                < Subj cat> = np
                !Agree ( Subj, _, plural )
                < * head sem pred> = perfective
                !Arg ( 1, Vcomp)

persuades  v  < * head form> = finite
                < * subcat> = [ Obj, Vcomp, Subj ]
                < Obj cat> = < Subj cat> = np
                < Vcomp cat> = vp
                < Vcomp head form> = infinitive
                < Vcomp subcat> = [ Obj ]
                !Agree ( Subj, 3, singular )
                < * head sem pred> = persuade
                !Arg ( 1, Subj )
                 !Arg ( 2, Obj )
                 !Arg ( 3, Vcomp)

to              v  <* head form> = infinitive
                < * subcat> = [ VP, NP]
                < NP cat> = np
                < VP cat> = vp
                < VP head form> = nonfinite
                < VP subcat > = [ NP ]
                < * head sem> = < VP head sem>

arthur        np !Agree ( *, 3, singular)
                < * head sem> = arthur

cornwall   np !Agree ( *, 3, singular)
                < * head sem> = Cornwall

knights     np !Agree ( *, 3, plural)
```

```
                < * head sem> = knights

# Define relations


Agree  ( X,
         < X head agreement person >,
         < X head agreement number> )

Arg ( N, X ) < * head sem arg N > = < X head sem >

# Lookup root < * cat > = v/np

# Restrict Cat Form

          < * cat> = Cat
          < * head form> = Form
```

References

Carroll, J., G.J. Russell, and S. Warwick. (forthcoming) Adapting unification grammars for generation. ISSCO WP56.

Johnson, R. 1988. Le traitement automatique de la langue. In: Workshop on Artificial Intelligence and Robotics, Sion.

Johnson, R., and M. Rosner. 1989. A rich environment for experimentation with unification grammars. ACL-Europe.

Rupp, C. J. 1989. Situation semantics and machine translation. ACL-Europe.

Shieber, S. M. 1986. An introduction to unification based approaches to grammar. Chicago: University of Chicago Press.

Warwick, S. 1986. Building up a lexicon for NLP applications. LDV-Forum, 5.4, Sept.