WPFG

working papers in functional grammar

wpfg no. 45
February 1992

ProfGlot: a multi-lingual natural language processor

Simon C. Dik and Peter Kahrel
University of Amsterdam

ProfGlot

a multi-lingual natural language processor

Simon C. Dik
Peter Kahrel

Institute for General Linguistics
University of Amsterdam
Spuistraat 210
1012 VT Amsterdam
The Netherlands

1992

1 Introduction

This article describes a computer program called ProfGlot, written in Prolog and using the theory of Functional Grammar in the version described in Dik (1989b). ProfGlot is an integrated system in two senses: it can deal with different languages in terms of very similar structures and procedures; and it not only has the capacity of producing linguistic expressions in these languages, but also of parsing them, of translating in all directions between these languages, and of drawing certain logical inferences on the basis of given linguistic expressions. ProfGlot thus simulates some essential components of the linguistic competence of a multi-lingual speaker. The actual computer programs described in this article are distributed by Amsterdam Linguistic Software, P.O. Box 3602, 1001 AK Amsterdam (email amling@sara.nl). ALS also distribute an elementary computational Prolog course for linguists (Dik and Kahrel 1991). A detailed description is given in Dik (f.c.).

ProfGlot simulates some aspects of the linguistic competence of a multi-lingual speaker, with a fair knowledge of Danish, Dutch, English, French, German, and Spanish. It has the following capacities:

- ProfGlot can generate a great variety of grammatical construction types in these languages;
- ProfGlot can analyse or parse a subset of the constructions which it can generate, again in these languages;
- ProfGlot can translate the construction types generated in all directions between the languages;
- ProfGlot can perform a number of logical operations on the constructions which it can generate (e.g. it can paraphrase these constructions and otherwise infer a number of things from them);
- ProfGlot can also combine these operations: e.g., it can parse an English sentence, translate it into French, and infer a number of things from the French translation.

Some important features of ProfGlot are the following:

- ProfGlot is composed of a number of separate modules which can communicate with each other since they 'speak the same language': the theoretical language of Functional Grammar.
- Thanks to the design features of Prolog, the program is largely formulated in a declarative mode, containing facts and rules that together define well-formed constructions and permissible actions. This 'declarative' knowledge can, however, be put to 'procedural' tasks in ways which will become clear in the actual program description.
- In the process of constructing the multilingual competence of ProfGlot, care has been taken to separate the language-independent rules and

principles as much as possible from the language-particular facts and rules. This has led to a system in which the language-specific modules are comparatively small, and the greater part of the program consists of rules and principles which could also be used for other languages. This means that it is not too difficult to extend the competence of ProfGlot with further languages.

The aims of the present computational exercise are mainly theoretical. The project is part of a wider endeavour to model the linguistic capacities of natural language users by means of the theory of Functional Grammar (for other studies in this direction, compare Connolly and Dik 1989).

1.1 Some practical points

Prolog has several dialects. The programs described here have been written in standard Prolog (also called 'Edinburgh Prolog'), as described in Clocksin and Mellish (1987) and Bratko (1986).

Even for standard Prolog there are several different interpreters and compilers available. These may differ in minor points, but they will support the present programs, sometimes after minor adaptations.

Most Prolog interpreters meant to run on a PC do not offer sufficient space for exploiting the full capacities of the program. The most recent version[1] of LPA Prolog Professional Compiler, however, does allow ProfGlot to be run even on a PC. Some use has been made of certain built-in facilities of LPA-Prolog. These can be easily converted into standard Prolog conventions available in other interpreters.

Obviously, ProfGlot can also be run on bigger systems. Moving to such bigger systems will become necessary when the limited lexica of the present version are extended beyond a certain limit.

It is important to note that, even within the constraints imposed by the linguistic theory (Functional Grammar) and the programming language (Prolog), many rules and principles can be formulated in different ways and according to different algorithms. This means that even rules which do work satisfactorily could be formulated in other, and perhaps better ways. 'Better' in this context is a difficult notion, since it is composed of values on the dimensions 'space' (length of the program) and 'time' (processing time). Simpler, more elegant formulations may require more processing time, and therefore we cannot apply a simple one-dimensional simplicity metric to these programs.

One of the advantages of programming linguistic phenomena is that any relevant feature of linguistic organization must be made explicit and can thus be studied with an eye to the improvement of either the overall architecture or the local formulation of the rules and principles involved. For this reason, it is our conviction that computational programming of this kind will develop (in fact, will continue to develop) into an essential tool for the theoretical linguist.

1.2 Previous work on computational Functional Grammar

Pioneering work on computational Functional Grammar was done by Kwee (1979), who first designed a Functional Grammar generator in the computer language Algol68, and elaborated on this in several later studies (1981, 1987, 1988a, 1988b). The first study using Prolog is Connolly (1986), who developed rules for English constituent ordering by means of Prolog testing. Gatward, Johnson and Connolly (1986) showed how Functional Grammar could be used in a natural language processing system. Van der Korst (1987, 1989) first designed an English-French translator. Work on Functional Grammar-Prolog generators was done by Samuelsdorff (1989), Bakker (1988a, 1989), and Bakker, Van der Korst and Van Schaaik (1988). Parsing strategies were discussed in Janssen (1989), Gatward (1989), Dignum (1989a), and Kwee (1989). Voogt-Van Zutphen (1987, 1989) showed how the information contained in *Longman's Dictionary of Contemporary English* could be automatically converted into Functional Grammar predicate frames. Meijs (1988, 1989) and Vossen (1989) discussed how the network of definitional relations within such a dictionary can be exploited for semantic analysis. Dik (1987a, 1987b, 1987c, 1989) sketched how Functional Grammar could be used in a wider cognitive environment, compare also Dik (1989b, 1989c). The application of Functional Grammar ideas within a knowledge base environment was developed in Weigand (1986, 1987, 1989, 1990), Dignum et al. (1987), Capel and Westra (1987), and Dignum (1989b). Much of this work was collected in Connolly and Dik (1989). For further work related to the present enterprise, see the References.

2 Introducing Functional Grammar

ProfGlot implements the theory of Functional Grammar (FG), in the version described in Dik (1989b)[2], adapted to the requirements of Prolog programming. The adaptations are of different kinds:

- all FG structures, rules, and principles have been cast in a format which can be interpreted by Prolog.
- many rules and principles have been formulated more precisely than in the earlier informal descriptions, so as to make them work within a computational model.
- at a number of points, programming in Prolog has led to modifications, simplifications, and substantial improvements in the Functional Grammar formalism.

2.1 Outline of the Functional Grammar generator

The generator forms the central component of ProfGlot. Other modules (the parser, the logic, the translator) use the rules and the output of the generator. The overall structure of the generator is laid out in Figure 1.

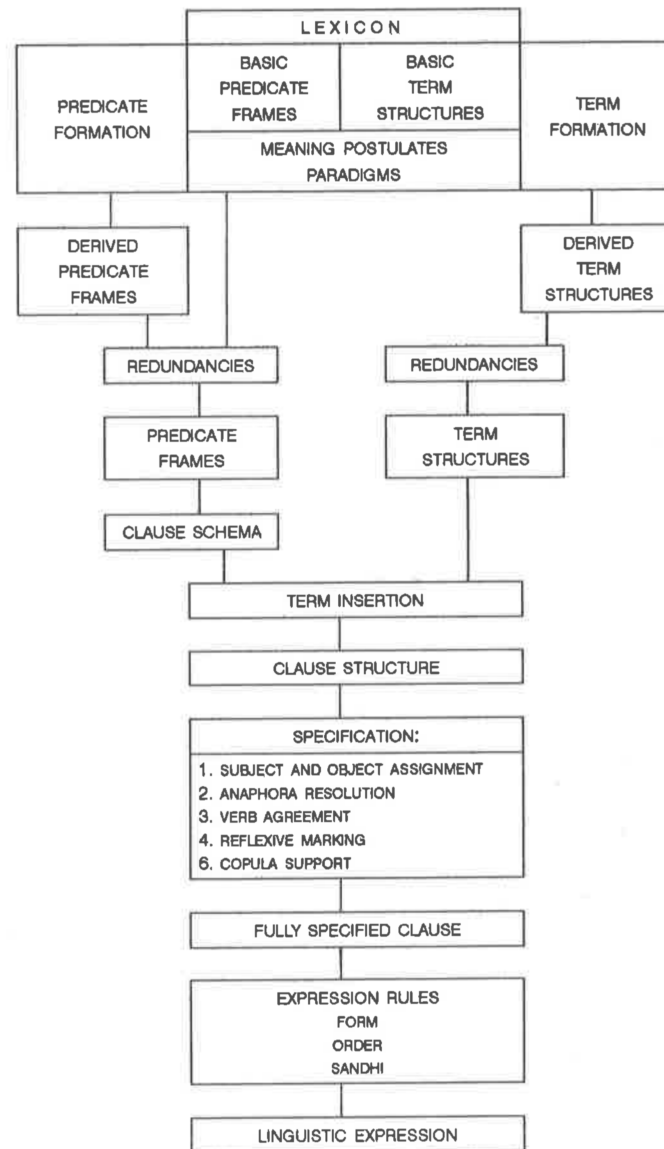

Figure 1. Lay-out of the Functional Grammar generator

The main steps in the construction of a linguistic expression are the following:

- building up a *clause structure*;

- turning the clause structure into a *fully specified clause* through a series of operations together called the *specification*;
- mapping the fully specified clause onto a linguistic expression through a system of *expression rules*.

The nucleus of the clause structure consists of a *predicate frame*, which may be basic or derived. Basic predicate frames are contained in the lexicon, which contains all the basic, non-derivable contentives of the languages, with a specification of their non-derivable formal and semantic properties. An example of a basic predicate frame is the following:

(1)
```
bpredv(eng,[[walk],[act,mve],[[[anim],t,[ag]]]]).
```

This predicate frame defines the structure:

(2)
```
[[walk],[act,mve],[[[anim],t,[ag]]]]
```

as a basic verbal predicate (bpredv) of English (eng). The structure takes the form of a Prolog list, consisting of three sublists, as follows:

(3)
```
[
[walk],                 : the form
[act,mve],              : the type
[[[anim],t,[ag]]]       : the argument positions
]
```

The first sublist defines the *form* of the predicate (generally, the stem from which the other forms can be most easily derived). The second sublist defines the *type* of the predicate, which consists of features (in this case 'action' and 'movement') which are essential to the behaviour of the predicate in the grammar. The third sublist contains one or more sublists defining the argument positions of the predicate. Each argument position consists of three parts, as follows:

(4)
```
[
[anim]              : the selection restriction
t                   : the term position
[ag]                : the semantic function
]
```

The *selection restriction* is used to block undesired output such as:

(5) a. *The table walks.*
    b. *The cloud kisses the car.*

The *term position* is the place where a term may be inserted. The *semantic function* specifies the role of the entity in question in the State of Affairs designated by the predicate frame.

The predicate *walk* has only one argument position (it is a 'one-place' predicate). The following is an example of a two-place predicate:

(6) `bpredv(eng,[[kiss],[act],[[[anim],t,[ag]],[[anim],t,[pt]]]]).`

In (6), *kiss* is defined as a two-place relation between an animate agent and an animate patient.

Adjectives and nouns are likewise coded in predicate frames:

(7) `bpreda(eng,[[clever],[grad,eval,comm],[[[anim],t,[zero]]]]).`
(8) a. `bpredn(eng,[[man],[hum,masc]]).`
b. `bpredn(eng,[[john],[hum,masc,proper]]).`

Just like verbs, adjectives consist of a form, a type, and a list of argument positions. Thus, (7) defines *clever* as a one-place, gradable, evaluative, and commentative predicate applicable to animate entities. (8a) defines *man* as a basic nominal predicate of the type 'human, masculine'. (8b) similarly defines the proper noun *john*.

Associated with the predicate frames the lexicon contains *meaning postulates/definitions*, and *paradigms*. An example of a meaning postulate is:

(9)
```
mean(eng,
     [[kiss],[act],[[[anim],X1,[ag]],[[anim],X2,[pt]]]],
     [_,[[touch],[act],[[[anim],X1,[ag]],[[concr],X2,[pt]]]],
     [_,[[idiom],'the lips',[instr]],_,_]]).
```

This meaning postulate claims that to say that 'X1 kisses X2' is to say that 'X1 touches X2 with the lips'. The general structure of meaning postulates is:

(10) mean(L,Definiendum,Genus,Differentia).

In this case, 'kiss' (Definiendum) is defined as 'touch' (Genus) 'with the lips' (Differentia). The combination of Genus and Differentia forms the Definiens. Note that the Definiens has the same syntactic properties as the Definiendum. This means that in a structure in which we find the Definiendum, we can replace it by the Definiens, thus arriving at a paraphrase of the original structure.

The paradigms contain the irregular, unpredictable forms of a predicate. Examples are:

(11) a. `paradigm(eng,[child,children]).`
b. `paradigm(eng,pres,be,[is,am,are]).`

According to the principle of 'lexical priority' (see Dik 1989c), the expression rules which define the forms of constituents will first check whether the required form is contained in a paradigm. Only when no such form is found will the regular rule be applied.

Derived predicate frames contain those predicates which can be productively formed by means of predicate formation rules. Derived predicate frames have the same syntax as basic predicate frames.

The 'types' of predicate frames only contain the non-redundant semantic features characterizing the predicate. Redundant features (in the sense in which 'human' entails 'animate', and 'action' entails 'control' and 'dynamism') are introduced by means of redundancy rules.

The abstract underlying clause structure which is built up around a predicate frame has a standardized syntax for all linguistic expressions of the six languages which ProfGlot has knowledge of. This standardized syntax is outlined in Figure 2.

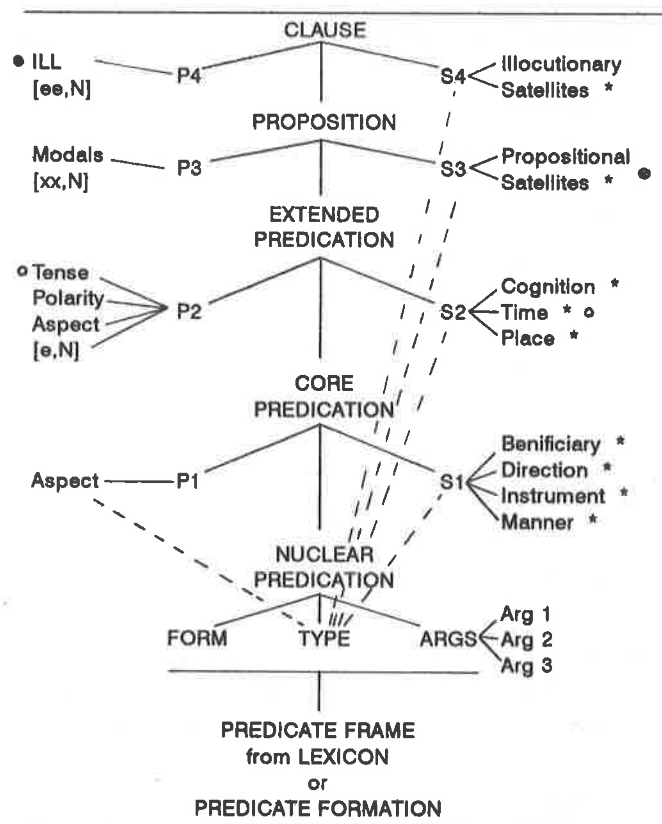

Figure 2. Standardized syntax for clause structures.

The overall structure of the clause can be represented as follows:

(12) `[P4,[P3,[P2,[P1,[NUCLEUS],S1],S2],S3],S4]`

The nucleus or nuclear predication consists of a predicate frame as selected from the lexicon or created through predicate formation. Predicate frames have up to three argument positions (in this program). The 'P' elements symbolize grammatical operators at four different levels, and the 'S' elements symbolize lexical modifiers or satellites at these same levels.[3] The elements of the different layers have the following semantic import:

(13) nucleus:
defines a type of State of Affairs (SoA).
predicate operators P1,
predicate satellites S1:
give additional specifications of the nature of the nuclear SoA.
predication operators P2,
predication satellites S2:
define the 'location' of the SoA with respect to spatial, temporal, and objective-cognitive coordinates.
propositional operators P3,
propositional satellites S3:
define the Speaker's attitude towards and evaluation of the content of the proposition
illocutionary operators P4,
illocutionary satellites S4:
define, specify, and motivate the illocutionary force or speech act value of the clause as a whole.

In the present program these various elements have been specified as follows:

(14) core predication = [P1,[NUCLEUS],S1]
P1 = Progressive Aspect or empty ([]).
S1 = Satellites of Manner, Instrument, Direction, and Beneficiary, possibly empty ([*]).

(15) extended predication = [P2,[CORE PREDICATION],S2]
P2 = Tense, Polarity (empty/positive/negative), Perfect Aspect or empty, and the SoA variable 'e', with a random integer as identifier.
S2 = Satellites of Place, Time, and 'Cognition' (Reason, Condition, Concession), possibly empty.

(16) proposition = [P3,[EXTENDED PREDICATON],S3]
P3 = Modal operators (empty/predictive/possibility), and propositional variable 'xx', plus random identifier.
S3 = Modal/attitudinal satellites such as *probably* and *cleverly*, possibly empty.

(17) clause = [P4,[PROPOSITION],S4]
P4 = Illocutionary operators, illocutionary variable 'ee' plus random identifier.
S4 = Illocutionary satellites, possibly empty.

Figure 3 below gives an example of an actual underlying clause structure generated by the program. In this clause structure, the asterisks indicate satelite positions that have not been filled. (The stucture would be expressed as *Frankly, eh, Mary probably hits the flower today)*.

```
[792]
P4: [decl,[ee,2]]
P3: [[],[xx,4]]
P2: [[pres,sg,p3,[]],[],[],[e,23]]
P1: []
PR: [[act],[hit]]
TY: [act,dyn,contr]
AR: SE: [anim]
    TE: OP: [def,sg,[x,21]]
        R1: [[mary],[hum,fem,proper,anim,concr,vert]]
    FU: [subj,ag]
AR: SE: [concr]
    TE: OP: [def,sg,[x,8]]
        R1: [[flower],[inanim,concr]]
    FU: [obj,pt]
S1: [[*],[*],[*],[*]]
S2: [[*],[today],[*],[*]]
S3: [probably]
S4: [frankly, eh,]



Press any key to continue or Esc to return ...
```

Figure 3: Underlying clause structure

Within the overall structure of the clause, there are certain dependencies between different elements at the same layer or at different layers. These dependencies have been indicated by dotted lines in Figure 2. For example, the specification of Time satellites should correlate with the specification of the Tense operator, as is clear from:

(18) a. *John arrived yesterday.*
b. **John arrived tomorrow.*

(19) a. *John will arrive tomorrow.*
b. **John will arrive yesterday.*

In order to form a clause structure we take the following steps:

[1] take a basic predicate frame from the lexicon or create a derived predicate frame through predicate formation;

[2] add redundant features to the Type of the predicate frame through redundancy rules.

[3] create a core predication schema through specifying P1 operators and S1 satellites around the nuclear predicate frame;

[4] create an extended predication schema through specifying P2 operators and S2 satellites around the core predication schema.

[5] turn the extended predication schema into an extended predication through inserting appropriate terms into its argument and satellite slots.

[6] create a proposition by specifying P3 operators and S3 satellites around the extended predication;

[7] create a clause structure through specifying P4 operators and S4 satellites around the proposition.

At step [5], terms are inserted into the open argument and satellite slots of the extended predication schema. *Terms* are expressions which can be used to refer to (concrete or abstract) entities of different types. They range from simple pronouns such as *he* to complex noun phrases such as:

(20) *the man who thought that Sarphati Street was the most beautiful street in Amsterdam*

Just like clause structures, terms have been assigned a uniform syntax, as outlined in figure 4 on the following page. A term structure is a list, consisting of two sublists, one for term operators, and one for one or more restrictors:

(21) term structure = `[Operators,Restrictors]`

The term operators specify parameters relevant to the set of intended referents as a whole, while the restrictors specify properties that an entity must have in order to qualify as a potential referent of the term.

The term operators have positions for 'definiteness' (including demonstratives and quantifiers) and 'number', and a position for the term variable 'x', provided with a random identifier. The restrictors (in this program) are divided over four potential positions R1, R2, R3, and R4:

(22) term structure = `[[Def,Num,Var],[R1,R2,R3,R4]]`

- R1 is always filled and consists of a nominal predicate plus its type.
- R2 is a position for attributive adjectives and participles; no iteration of these attributive modifiers has been allowed so far.
- R3 is a position for adpositional attributes such as *John's*, *of John*, *in the city* (as in: *the house in the city*), and *with the red dress* (as in: *the girl with*

the red dress). Note that R3 contains terms, and that further terms can be embedded into these terms, as in: *the house of the father of the boy with the blue eyes*. This creates a danger of 'infinite recursion' which Prolog is unable to handle unless certain measures are taken in order to stop the recursive iteration at a certain point. Such measures have indeed been taken, so that quite complex R3 modifiers can be formed without the system running into infinite loops (see section 4.2).

- R4 is a position for relative clauses, which themselves have the syntax of extended predications (see Figure 2). Again, measures have been taken to control infinite recursion without disallowing relative clauses within relative clauses within ... relative clauses.

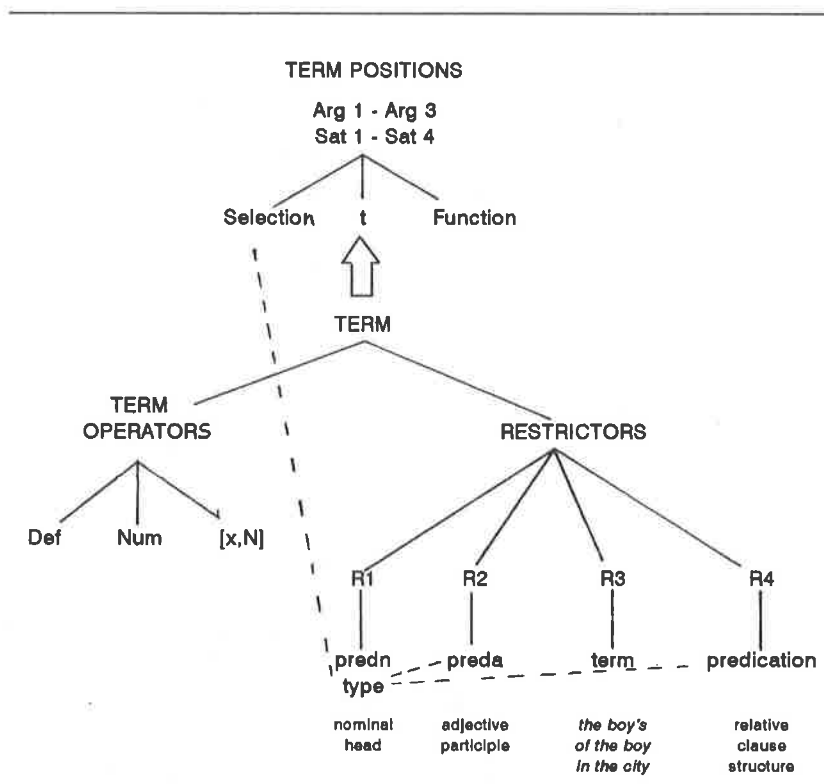

Figure 4. Standardized syntax for term structures

Figure 5 on the following page is an example of an actual term structure generated by the program. This term would be expressed as *Mary's clever father*.

```
?-term(eng,[[Def,Num,Var],[R1,R2,R3,R4]],[]).

Def = def
Num = sg
Var = [x,13]
R1 = [[father],[hum,masc,anim,concr,vert]]
R2 = [[clever],[grad,eval,comm,state]]
R3 = [[],[[def,sg,[x,10]],[[[mary],[hum,fem,proper,anim,concr,vert]]]],[poss]]
R4 = []


SPACE BAR = next solution _
```

Figure 5. A term structure.

There are certain dependencies within the structure of terms, indicated by dotted lines in Figure 2. Adjectives and relative clause structures have been made sensitive to the type of the head noun R1, so as to prevent output such as:

(23) **the clever stones who kiss the book*

And the whole term has been made sensitive to the selection restriction of the term position which it is to be inserted into, so as to prevent output such as:

(24) **The book kissed the stones.*

The guiding principle has here been: if ProfGlot can be taught to speak prose first, it may learn poetry later. In general, all productions are not only syntactically well-formed, but also semantically coherent, though not always pragmatically plausible.

When clause structures have been formed in the way sketched above, they go through a number of mappings which are together called the *specification*. The specification operates on the propositional part of the clause structure, and is defined as follows:

(25)
```
clause_specify(L,R1,R7) :-
    subj_obj_assignment(L,R1,R2),
    verb_agreement(L,R2,R3),
    anaphora(R3,R4),
    reflexive(R4,R5),
    equi(R5,R6),
    copula_support(L,R6,R7).
```

Note that an input structure R1 is passed through a number of mappings which finally yield the output structure R7. Some of these mappings are tied to the language concerned (English, French, Dutch, etc.) through the language identifier L. For example, subject and object assignment possibilities are different for the three languages, and have therefore been parametrized by the language identifier. Note that the output of each mapping is handed

over as input to the next mapping. Some of the mappings do not necessarily affect every input structure. This has been taken care of by adding a default 'identity mapping' which hands over its input unchanged to the next mapping. The different mappings of the specification have the following effects:

(26) *subj_obj_assignment*
assigns the functions subject and object in alternative ways to terms in the clause structure, yielding active, passive, and 'dative' variants of the same underlying structure.

(27) *verb_agreement*
copies the relevant features of person, number, and gender from the subject term to the Tense operator on the predication.

(28) *anaphora*
anaphora has so far been handled through the insertion of an anaphorical 'free' term structure into one of the term positions: through the mapping 'anaphora' this free term structure assimilates the features of person, number, and gender of a possible antecedent.

(29) *reflexive*
marks an anaphoric term as reflexive if that is the way it must be expressed.

(30) *equi marking*
marks an anaphoric term as 'equi' if it requires zero expression, as in *John wants φ to kiss Mary.*

(31) *copula support*
adds a copular verb to a non-verbal predicate as required for the proper operation of the expression rules.

The result of the specification is a *fully specified clause*. This structure contains all the elements essential for both the semantic interpretation and the formal expression of the clause structure. The fully specified clause forms the input to the expression component.

The *expression rules* serve to map the fully specified clause onto one or more linguistic expressions through which it can be expressed. They have been organized in the following way:

(32)
1. formal expression rules
2. placement of constituents
3. sandhi

The formal expression rules specify the forms which the constituents of the clause structure take, given the operators and functions with which they occur in the fully specified clause. In the formal expression of constituents, only that information (apart from the forms themselves) is retained which is of relevance to the correct placement of constituents. In certain cases special codes are added in the formal expression with the sole purpose of correctly activating the placement rules. 'Form' and 'order' are thus not fully independent of each other. The placement of constituents is achieved in the following way (compare Connolly 1986):

- there is a unified 'ordering template' containing a great number of positions, all initially empty, into which constituents may be placed.
- there is a sequence of 'placement rules' such that each placement rule carries one or more constituents of the clause to its position or their positions in the template.
- different combinations of placements may be used to get the right output order for different clause types within and across languages.
- each placement rule hands its output over as input to the next placement rule.
- when a placement rule does not apply (e.g., because the relevant constituent is not present in the input), the input is handed over unchanged through an 'identity placement'.
- all positions in the template not filled by a constituent are left 'empty' as marked by the empty list [].
- all empty lists (and other bracketings) are finally removed through a 'flattening operation'.

Sandhi rules effect 'low level' formal adjustments after the placement of constituents. They are mainly used in the French expression component to account for such differences as between:

(33)	before sandhi		after sandhi
	à la fille	—	*à la fille*
	à le garçon	→	*au garçon*
	cette fille	—	*cette fille*
	ce enfant	→	*cet enfant*
	le garçon	—	*le garçon*
	le enfant	→	*l'enfant*

These sandhi rules imply some concessions to the Functional Grammar principle that constituents, once introduced, should not be changed. As the examples show, some such changes are effected in the sandhi component (but only there).

It should be noted that no systematic attempt has been made so far to make a principled distinction between a phonological representation and an orthographic representation within the expression component. Certain regularities which involve phonology rather than orthography have nevertheless been captured where this seemed to allow for a more principled and/or less complex formulation of the expression rules. The mixture of phonological and orthographic phenomena involved has been termed 'pseudo-phonology'. Obviously, a more consistent treatment of such cases would require a better distinction between the levels of phonology and orthography.

One important feature of clause structure which is completely absent in the program as developed so far is the specification of pragmatic functions such as Topic and Focus. Integrating these pragmatic functions into the system is one of the most important points on the agenda for the further development of the ProfGlot program.

3 Overall structure and description of the ProfGlot program

This section outlines the overall structure of the ProfGlot program, the different modules of which it is composed, the functions of these modules, and the way in which they interact in performing various linguistic tasks. The overall structure of ProfGlot is presented in Figure 6 on the next page, in which boxes represent modules of the program, the shaded boxes represent inputs to and outputs of the modules, and the arrows represent the different relations between the modules and the input and output structures. We restrict ourselves here to the two languages English and French. The relevant modules can be replaced by the corresponding modules for any of the other languages. The separate modules to be described below have the following characteristics:

ProfGlot: user interface and starting menus;

BasFac: basic Prolog facilities, output facilities, metalinguistic definitions and operations;

EngLex: English lexicon, containing predicate frames, meaning postulates, lexical satellites, and paradigms of non-productive forms;

FreLex: French lexicon;

UniGen: 'universal' generator for defining fully specified underlying clauses for English, French, and Dutch;

ParSel: that part of the expression component in which the relevant irregular forms are selected from their paradigms;

EngExp: English part of the expression component;

FreExp: French part of the expression component;

UniExp: 'universal' (part of) the expression component, mapping underlying fully specified clauses onto linguistic expressions.

UniPar: parser for mapping sentences onto underlying clause structures or fully specified clauses;

UniTra: translator between the different languages;

UniLog: 'universal' logic for inferring linguistic expressions from linguistic expressions.

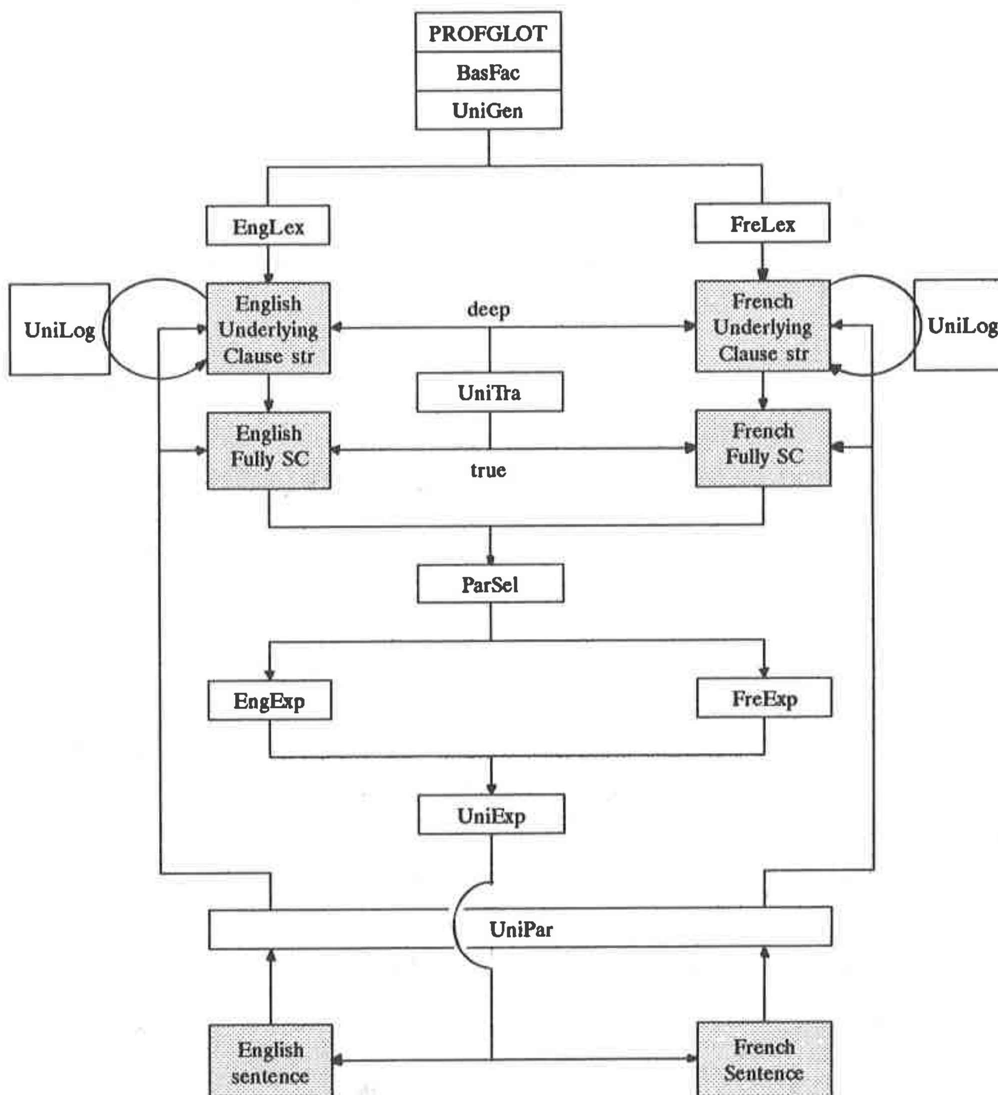

Figure 6. Structure of the ProfGlot program

When the program is started, a menu is presented (see figure 7 on the folowing page) in which the user can choose a function, which will be described below.

```
   Please indicate what you want PROFGLOT to do.

 1 Generate and present underlying clause structures
 2 Generate sentences
 3 Parse given input sentences
 4 Translate from one language to another
 5 Draw logical inferences from underlying clause structures

   Esc Exit
   Change settings
   Set printer

 Choice:
```

Figure 7: The main menu.

3.1 Generate and present

Using BasFac, UniGen, and one of the language-specific lexica, ProfGlot can generate underlying clause structures in the language in question. Through a formatting routine these structures are written on the monitor in a readable form; in figure 3 you can see an example.

3.2 Generate sentences

Using ParSel, UniExp, and the relevant language-specific expression component, ProfGlot can output these fully specified clauses as sentences of the relevant language; figure 8 shows English sample output.

```
[34] the professor deplores that john hits mary.

[35] john's hitting mary is deplored by the professor.

[36] the fact that john hits mary is deplored by the professor.

[37] it is deplored by the professor that john hits mary.

[38] that john hits mary is deplored by the professor.

[39] the professor deplores mary's being hit by john.

[40] the professor deplores the fact that mary is hit by john.

[41] the professor deplores that mary is hit by john.

[42] mary's being hit by john is deplored by the professor.

[43] the fact that mary is hit by john is deplored by the professor.
```

Figure 8: Output of the sentence generator

3.3 Parse given input sentences

Using BasFac, UniPar and UniGen, ParSel, UniExp, plus the relevant language-specific lexicon and expression component (e.g., EngLex and EngExp), ProfGlot can parse a class of input sentences of the relevant

language (in this case, English). Note that the parser strategy adopted here differs in certain respects from what is commonly understood by a parser:

- The parser maps the input sentences immediately onto the underlying clause structure, without any intermediate level of some kind of 'syntactic' tree. Such trees have no status within this model. Note that in this way the parser provides immediate access to the rich store of semantic information contained in Functional Grammar clause structures.
- Two parsing strategies have been implemented: 'true parse' reconstructs the fully specified clause underlying the input sentence; 'deep parse' directly reconstructs the underlying clause structure.
- The parser makes essential use of the generator, in the sense that most rules of the parser take the following form:

(33) Surface form F can be parsed as underlying structure US if there are rules of the generator through which F can be generated from US.

In this sense, the parser is a kind of inverted generator: a sentence is analysed by considering how it could be formed.

When the user chooses to parse a sentence, the program will prompt him to indicate which language and which parse mode (true or deep) should be used; next he will be prompted to type a sentence. After the sentence has been processed, the corresponding underlying structure is dispayed; see figure 9 for an example.

```
Type a sentence and press Enter
Do not forget to type a . ? or !

Sentence: did john give a book to mary ?

P = [[interr,[ee]],[[[],[xx]],[[[past,sg,p3,_8228],[],[e]],[[],[[[act],[give]],[
act,dyn,contr],[[[anim],[[def,sg,[x]],[[[john],[hum,masc,proper,anim,concr,vert]
]]],[subj,ag]],[[inanim],[[indef,sg,[x]],[[[book],[inanim,concr]],[],[],[]]],[ob
j,pt]],[[anim],[[def,sg,[x]],[[[mary],[hum,fem,proper,anim,concr,vert]]]],[rec]]
]],[[*],[*],[*],[*]]],[[*],[*],[*],[*]]],[*]],[*]]

SPACE BAR = next solution _
```

Figure 9. The parser.

3.4 Translate

By using UniTra, ProfGlot can translate fully specified clauses in all directions between the languages. Because of the standardized nature of such underlying structures, translation at this level is much easier than at the level of surface sentences (compare Van der Korst 1989). Two translation procedures have been implemented: 'true translation', which bridges the gap

between two languages at the level of the fully specified clause; and 'deep translation' which crosses over from one language to another at the level of the underlying clause structure. Since the underlying clause structure is neutral as between active and passive realizations, deep translation will give both active and passive correspondents in the target language. True translation will yield active correspondents for active sentences, and passive correspondents for passives. UniTra allows for the following procedures, which will be illustrated by sample output below:

(34) 1. generate a clause structure in L1;
2. translate it into L2;
3. express it in L2 (see figure 10).

(35) 1. generate a fully specified clause in L1;
2. translate it into L2;
3. express it in L2 (see figure 11).

(36) 1. true-parse an L1 input sentence;
2. translate the resulting structure into L2;
3. express it in L2.

(37) 1. deep-parse an L1 input sentence;
2. translate the resulting structure into L2;
3. express it in L2 (see figure 12).

```
[[decl,[ee,2]],[[[],[xx,21]],[[[pres,sg,p3,[]],[],[],[e,24]],[[],[[[act],[cheat]
],[act,dyn,contr],[[[anim],[[def,sg,[x,14]],[[[painter],[hum,masc,anim,concr,ver
t]],[],[],[]]],[subj,ag]]]],[[*],[*],[*],[*]]],[[*],[*],[*],[*]]],[*]],[*]]

[[decl,[ee,2]],[[[],[xx,21]],[[[pres,sg,p3,[]],[],[],[e,24]],[[],[[[act],[trich]
],[act,dyn,contr],[[[anim],[[def,sg,[x,14]],[[[peintre],[hum,masc,anim,concr,ver
t]],[],[],[]]],[subj,ag]]]],[[*],[*],[*],[*]]],[[*],[*],[*],[*]]],[*]],[*]]

Possible expressions of source structure:
 the painter cheats.


Possible expressions of target structure:
 le peintre triche.
```

Figure 10: Output of translate 1

```
Underlying clause structure in eng:
[[decl,[ee,22]],[[[],[xx,17]],[[pres,[],[],[e,8]],[[],[[hit],[act,dyn,contr],[[[
anim],[[def,sg,[x,3]],[[[professor],[hum,masc,anim,concr,vert]],[],[],[]]],[ag]]
,[[concr],[[def,sg,[x,3]],[[[john],[hum,masc,proper,anim,concr,vert]]]],[pt]]]],
[[*],[*],[*],[*]]],[[*],[*],[*],[*]]],[*]],[*]]

Equivalent UCS in fre:
[[decl,[ee,22]],[[[],[xx,17]],[[pres,[],[],[e,8]],[[],[[frapp],[act,dyn,contr],[
[[anim],[[def,sg,[x,3]],[[[professeur],[hum,masc,anim,concr,vert]],[],[],[]]],[a
g]],[[concr],[[def,sg,[x,3]],[[[jean],[hum,masc,proper,anim,concr,vert]]]],[pt]]
]],[[*],[*],[*],[*]]],[[*],[*],[*],[*]]],[*]],[*]]

Fully specified underlying clause structure in fre:
[[decl,[ee,22]],[[[],[xx,17]],[[[pres,sg,p3,masc],[],[],[e,8]],[[],[[[act],[frap
p]],[act,dyn,contr],[[[anim],[[def,sg,[x,3]],[[[professeur],[hum,masc,anim,concr
,vert]],[],[],[]]],[subj,ag]],[[concr],[[def,sg,[x,3]],[[[jean],[hum,masc,proper
,anim,concr,vert]]]],[obj,pt]]]],[[*],[*],[*],[*]]],[[*],[*],[*],[*]]],[*]],[*]]

Possible expressions of this FSUCS:
 le professeur frappe jean.
```

Figure 11. Output of Translate 2.

```
Type a sentence and press Enter
Do not forget to type a . ? or !

Sentence: mary was given a book by john .

 jean donnait un livre à marie.

 un livre etait donné à marie par jean.

yes

?- _
```

Figure 12. Deep translation.

In figure 12 it can be seen that the passive input sentence was parsed to the level where the syntactic functions Subject and Object are not relevant. This structure is translated into the corresponding structure in the target language (here French), resulting in the expression of an active and a passive clause. If the same sentence had been input into the true translator, only the passive sentence would have been returned.

3.5 Logical inferences

The inferential capacity of ProfGlot is contained in UniLog, a program which can derive 'propositions' from 'propositions' while preserving logical validity. Remember that 'proposition' here means the clause structure minus the illocutionary elements P4 and S4:

(38) clause = [P4,PROPOSITION,S4]

The rules of UniLog allow for the production of a variety of valid inferences from underlying clause structures. The inferred propositions have the same

syntax as the original ones. They can thus be output in the same language, or be translated into another language and then output in that language.

The two figures below illustrate two capacities of the logic module. In figure 13 the user chose to store a proposition. The program prompted the user to enter a sentence and distilled from that sentence the proposition, which is stored in the database for future reference.

```
Type a sentence and press Enter
Do not forget to type a . ? or !

Sentence: john sold a book to mary .


I will remember the proposition:

[[[],[xx]],[[past,[],[],[e]],[[],[[sell],[act,dyn,contr],[[[anim],[[def,sg,[x]],
[[[john],[hum,masc,proper,anim,concr,vert]]]],[ag]],[[inanim],[[indef,sg,[x]],[[
[book],[inanim,concr]],[],[],[]]],[pt]],[[anim],[[def,sg,[x]],[[[mary],[hum,fem,
proper,anim,concr,vert]]]],[rec]]]],[[*],[*],[*],[*]]],[[*],[*],[*],[*]]],[*]]

yes

?- .
```

Figure 13: Asserting knowledge.

In figure 13, the program generated a sentence, which is printed under `The following sentence:`. Using the underlying structure of that sentence, the logic module made inferences, which are also in the form of underlying structures – these are expressed and printed on the screen.

```
The following sentence:

 mary moves john.

entails:

 mary moves john.

 john is moved by mary.

 a female person moves johr..

 john is moved by a female person.

 mary moves a male person.

 it is noT true that john is noT moved by mary.
```

Figure 14. Output of infer

4 Some special features

4.1 Linguistic complexities

In its basic running mode, ProfGlot produces only simple declarative sentence types. All kinds of more complex constructions are 'closed off' in the program, since the chance that they will be chosen is initially set at 0%. Any of these complications can be evoked by defining some non-zero chance that the relevant item will be chosen at those junctures where it can be chosen in the program.

Consider one simple example. In its basic mode, the program does not produce adjectives, because the initial chance that adjectives will be chosen is set at 0%. Before or while running the program, however, we can define a new chance for adjectives, which can vary between 0 and 100%. If we define the chance for adjectives at 25%, there is a chance of 1 out of 4 that an adjective will be selected at those points where an adjective can be selected at all. Since many of these settings have been defined in the program, we can in this way greatly vary and complicate the output. All settings are compatible with each other: they can be activated in different combinations with any specification of chances in order to study the behaviour of, and the interaction between the relevant grammatical phenomena. The following linguistic complexities can be triggered through these settings:

Sentence and clause types:

- yes/no questions, question word questions
- constructions with embedded propositions and predications, including (some) nominalized and infinitival realizations

Term structures:

- personal and anaphoric (including reflexive) pronouns
- attributive and predicative adjectives
- attributive adpositional term modifiers: *john's book, the man in the garden*
- relative clauses.

Term operators:

- singular and plural
- definite and indefinite
- demonstratives
- quantifiers: *every, all, many, few*

Predicate elaboration:

- present and past tense
- progressive aspect

- perfect aspect
- negative and emphatic positive polarity
- predictive and potential modality

Satellites:

- satellites of level 1, predicate satellites:
  Beneficiary, Direction, Instrument, Manner
- satellites of level 2, predication satellites:
  Time, Place, and 'Cognition' (e.g. Reason).
- satellites of 'Cognition' with the internal form of propositions or predications (subordinate clauses):
  Reason, Condition, Concession
- satellites of level 3, propositional satellites:
  probably, *cleverly* (as in: *Cleverly, John answered the question.*).
- satellites of level 4, illocutionary satellites:
  frankly

Predicate formation:

- agent nouns derived from action verbs
- participles derived from verbs
- comparatives (positive, negative, equality) derived from gradable adjectives
- degree + adjective: *very good*, *surprisingly good*
- term predicates derived from terms:
  john is a good man.
- adpositional predicates:
  the boy is in the garden.
  this book is for john.

4.2 Avoiding infinite recursion

There are four construction types in the grammar which, if left unconstrained, would lead to infinite recursion or 'endless loops'. Since Prolog, through backtracking, always tries to find all solutions to a certain problem, the system would, whenever it enters such a loop, go on and on finding more deeply embedded constructions, and thus never come out of the loop again. This danger can be avoided, however, by manipulating the 'settings' for these construction types in such a way that the chance that the construction will be chosen is kept below 100%. The chance that the option will be chosen *again* will then decrease with each recursive step (compare Kwee 1979 for this type of strategy). This is relevant to the following constructions:

Attributive adpositional modifiers

(39) a. john's *book*
b. *the man* in the garden
c. *the man* with the book

Note that what we have here is a term embedded within another term. This would, without some kind of constraint, lead to infinite recursion. If we keep the chance that adpositional modifiers of this type will be chosen below 100%, however, this is avoided. At the same time, the higher we define the chance, the greater the odds that we get more complicated constructions of this type. Thus, manipulating the chance can be used both for increasing the depth of recursion, and for avoiding infinite recursion.

Relative clauses
Again, there is a danger of infinite recursion: a relative clause may contain a term which again contains a relative clause, etc. Again, the depth of embedding can be monitored by manipulating the chance that a relative clause will be construed at those points at which it can be construed.

Embedded propositions/predications
The setting 'emb(N)' opens up verbs (such as *expect, regret*) which take embedded complements. With a suitably high value for N in 'emb(N)' such constructions as the following can be generated:

(40) *john believes that the man deplores that the girl wants the boy to buy the book.*

Again, we keep the embedding chance below 100% in order to avoid infinite recursion.

Satellites in the form of adverbial clauses
Adverbial clauses may create infinite recursion in such constructions as:

(41) *john failed because he was ill because he had eaten rotten fish because ...*

Again, the problem can be monitored through appropriately manipulating the chance for 'subsat(N)'.

5 Conclusion and outlook

We have, within the limits of the present paper, only been able to give a sketch of the capacities of the ProfGlot system. This system has the following distinctive properties:

- It is based on a linguistic theory, the theory of Functional Grammar as developed since Dik (1978).
- It demonstrates that it possible to program FG in Prolog.
- It provides a particular solution to the problem of how a grammatical model can be used both in generating and in parsing linguistic expressions.
- It demonstrates that the FG underlying clause structure is an efficient *interlingua* for monitoring translation.
- It shows how the propositional part of the FG underlying clause structure can be used as a vehicle for logical reasoning.
- It demonstrates how all these capacities can be formulated with a maximum of cross-linguistically applicable rules and a minimum of language-dependent modules.

For a full-scale discussion of the actual ProfGlot program, see Dik (fc.). On how to obtain the program for developmental and research purposes, see section 1.

Further development of ProfGlot is envisaged and under way, in the following directions:

- Integrating ProfGlot with a full-scale lexicon. See Dik, Meijs, and Vossen (fc.).
- Further development of the capacities of the generator, the parser, the translator, and the logic.
- Applying ProfGlot to further languages, especially non-Indo-European ones. Some work has been done on Italian, Latin, Japanese and Arabic.
- Developing a 'discourse competence' for ProfGlot, so that not only isolated sentences, but coherent texts can be processed. In this connection: integrating pragmatic functions such as Topic and Focus into the program.

Notes

1. LPA Prolog Professional Compiler, version 3.15, 15 June 1990, available from Logic Programming Associates Ltd, London.

2. This version incorporates the improvements proposed in Hengeveld (1989).

3. Extensive argumentation for the layered structure of the clause can be found in Hengeveld (1989), Dik (1989f), and Dik, Hengeveld, Vester, and Vet (1990).

References

Bakker, Dik
1988 'De implementatie van een grammaticamodel' ['The implementation of a model of grammar'], *Tijdschrift voor Taal- en Tekstwetenschap* 8: 239-260.
1989 'A formalism for Functional Grammar expression rules', in: Connolly and Dik (eds.), 45-63.

Bakker, Dik, Bieke van der Korst, and Gerjan van Schaaik
1988 'Building a sentence generator for teaching linguistics', in: Zock and Sabah (eds.), vol. 2: 159-174.

Bolkestein, A.M., J. Nuyts and C. Vet (eds)
1990 *Layers and levels of representation in language theory.* Amsterdam: John Benjamins.

Bratko, Ivan
1986 *Prolog programming for artificial intelligence*. Reading, Mass.: Addison-Wesley.

Capel, Casper and Didan Westra
1987 *LIKE: Linguistic Knowledge Base Environment*. MA thesis, Department of Mathematics and Computer Science, Free University of Amsterdam.

Clocksin, W. F., and C. S. Mellish
1987 *Programming in Prolog*. Heidelberg: Springer.

Connolly, John H.
1986 'Testing Functional Grammar placement rules using Prolog', *International Journal of Man-Machine Studies* 24: 623-632.

Connolly, John H., and Simon C. Dik (eds.)
1989 *Functional Grammar and the computer*. Dordrecht: Foris.

Dignum, Frank
1989a 'Parsing an English text using Functional Grammar', in: Connolly and Dik (eds.), 110-134.
1989b *A language for modelling knowledge bases; based on linguistics, founded in logic*. [Diss. Free University Amsterdam].

Dignum, Frank, T. Kemme, W. kreuzen, H. Weigand and R.P van de Riet
1987 'Constarint modelling using a conceptual prototyping language. *Data and Knowledge Engeneering* 2, 213-254.

Dik, Simon C.
1978 *Functional Grammar*. Amsterdam: North-Holland [3rd printing 1981, Dordrecht: Foris].
1987a 'Functional Grammar and its potential computer applications', in: W. Meijs (ed.), *Corpus Linguistics and beyond*, 253-268. Amsterdam: Rodopi.
1987b 'Generating answers from a linguistically coded knowledge base', in: Kempen (ed.), 301-314.
1987c 'Linguistically motivated knowledge representation', in: M. Nagao (ed.), *Language and artificial intelligence*, 145-170. Amsterdam: North-Holland.
1989a 'FG*C*M*NLU: Functional Grammar Computational Model of the Natural Language User', in: Connolly and Dik (eds.), 1-28.
1989b *The theory of Functional Grammar*. Part 1: *The structure of the clause*. Dordrecht: Foris.
1989c 'The lexicon in a computational Functional Grammar.' Paper, Institute for General Linguistics, University of Amsterdam.
fc. *Functional Grammar in Prolog. An integrated implementation for English, French, and Dutch*. Berlin: Mouton-De Gruyter.

Dik, Simon C., Kees Hengeveld, Elseline Vester, and Co Vet
1990 'The hierarchical structure of the clause and the typology of adverbial satellites', in: Bolkestein, Nuyts, and Vet (eds.): 25-70.

Dik, Simon C. and Peter Kahrel
1991 *ProCourse: a Prolog course for linguists*. Amsterdam: Amsterdam Linguistic Software.

Dik, Simon C., Willem Meijs, and Piek Vossen
fc. Lexigram: a functional lexico-grammatical tool for knowledge engeneering. Paper.

Gatward, Richard A.
1989 'Implementation efficiency considerations in parsing Functional Grammar', in: Connolly and Dik (eds.): 77-91.

Gatward, Richard A., S.R. Johnson and J.H. Connolly (eds)
1989 'A natural language processing system based on Functional Grammar. In *Proceedings of the International Conference on 'Speech input/output: techniques and applications.* London, 24-26 March 1986. *IEE Conference Publication No. 258*, 125-128. London: IEE.

Hengeveld, Kees
1989 'Layers and operators in Functional Grammar'. *Journal of Linguistics*, 25, 127-157.

Hoekstra, Teun, Harry van der Hulst, and Michael Moortgat (eds)
1981 *Perspectives on Functional Grammar.* Dordrecht: Foris.

Janssen, Theo M.V.
1989 'Towards a universal parsing algorithm for Functional Grammar', in: Connolly and Dik (eds.): 65-75.

Kempen, Gerard (ed.)
1987 *Natural language generation; new results in artificial intelligence, psychology and linguistics.* Dordrecht: Martinus Nijhoff.

Korst, Bieke van der
1987 'Twelve sentences; a translation procedure in terms of Functional Grammar'. *WPFG* 19.
1989 'Functional Grammar and machine translation', in: Connolly and Dik (eds.): 289-316.

Kwee Tjoe Liong
1979 *A68-FG(3); Simon Dik's funktionele grammatika geschreven in algol 68 versie nr. 03. [Simon Dik's functional grammar written in algol68].* Publications of the Institute for General Linguistics 23, University of Amsterdam.
1981 'In search of an appropriate relative clause', in: Hoekstra et al. (eds.): 175-189. Dordrecht: Foris.
1987 'A computer model of Functional Grammar', in: Kempen (ed.): 315-331.
1988a 'Natural language generation: one individual implementer's experience', in: Zock and Sabah (eds.), vol. 2: 98-120.
1988b 'Computational theoretical linguistics: generating (English) sentences in (Dik's) Functional Grammar', in: D. Bojadziev, P. Tancig, and D. Vitas (eds.), *Proceedings of the VIth (Yugoslav) Conference (on) Computer Processing (and) Society of Applied Linguistics of Slovenija.* 75-90.
1989 'An ATN parser for English FG? Or maybe an active chart?', in: Connolly and Dik (eds.): 93-107.

Meijs, Willem
1988 'Knowledge-activation in a large lexical data-base: problems and prospects in the LINKS-project'. *Amsterdam Papers in English,* 1. Dept. of English, University of Amsterdam.
1989 'Spreading the word: knowledge-activation in a functional perspective', in: Connolly and Dik (eds.): 201-215.

Samuelsdorff, Paul O.
1989 'Simulation of a Functional Grammar in Prolog', in: Connolly and Dik (eds.): 29-44.

Van de Auwera, Johan and Louis Goosens (eds)
1987 *The ins and outs of predication.* Dordrecht: Foris.

Voogt-van Zutphen, Hetty
1987 'Constructing an FG lexicon on the basis of LDOCE'. *WPFG* 24.
1989 'Towards a lexicon of Functional Grammar', in: Connolly and Dik (eds.): 151-176.

Vossen, Piek
1989 'The structure of lexical knowledge as envisaged in the LINKS-project', in: Connolly and Dik (eds.): 177-199.

Weigand, Hans
1986 'An overview of the conceptual language KOTO'. Report nr. IR-112. Dept. of Mathematics and Information Theory, Free University Amsterdam.
1987 'Functional Grammar as a formal language', in: Van der Auwera and Goossens (eds.): 179-194.
1989 'A dialectical model of modality', in: Connolly and Dik (eds.): 247-271.
1990 *Linguistically motivated principles of knowledge base systems.* Dordrecht: Foris.

Zock, Michael and Gérard Sabah (eds)
1988 *Advances in natural language generation; an interdisciplinary perspective.* 2 volumes. London: Pinter Publishers.