WPFG

working papers in functional grammar

wpfg no. 61, september 1996
prijs: *f* 6.00

Comments on some aspects of ProfGlot (explanatory notes and critical assessment)

Kwee Tjoe Liong
Universiteit van Amsterdam

Working Papers in Functional Grammar (WPFG)

ISSN: 0924-1205
Editor: Lachlan Mackenzie
Executive editor: Gerry Wanders

WPFG, which appears at irregular intervals, publishes papers which are (a) not (yet) ready for official publication, but sufficiently interesting as contributions to ongoing discussions within FG and (b) papers that will be officially published, but publication of which will take at least one year after publication in WPFG. For all information contact the following address:

The editor of WPFG
IFOTT, University of Amsterdam
Spuistraat 210, 1012 VT Amsterdam, the Netherlands

INTERNET fg@let.uva.nl (Subj: wpfg)
telefax +31.20.5253052

Papers kunnen worden besteld door het verschuldigde bedrag over te maken op Postbank rekening nr 5515023 t.n.v. UvA-FdL-IFOTT, Amsterdam, onder vermelding van 'WPFG' en de gewenste nummers (zie binnenzijde van achterkaft voor de prijslijst).

From abroad, please send your order to the editorial address, and transfer the required amount in Dutch guilders (NLG =ƒ, see pricelist on inside of back cover) in one of the following ways:

a. Send an international money order or a EURO cheque made out in Dutch guilders to the editorial address (payable to "IFOTT, Fac. der Letteren, Universiteit van Amsterdam").
b. Transfer the money directly through Eurogiro to Postbank account no. 5515023 payable to "UvA-FdL-IFOTT". Please add ƒ 6.50 to cover bank charges.
c. All other cheques or money orders are acceptable ONLY if ƒ 20 is added to cover the extra bank charges.

Please specify the required items.
The papers will be sent to you as soon as we receive your payment.
For standing orders contact the editorial address.

The information given here was correct at the time of printing (September 1996), but may be subject to changes. — Please refer to the most recent issue of WPFG available to you if you want to place an order.

See inside of back cover for the list of papers that have appeared up to now. Although all titles are still available, we prefer that you consult the final (official) version of those papers which have subsequently been published in journals or books (marked by asterisk and square brackets).

We still have a few copies in stock of Simon Dik's 1978 monograph Stepwise Lexical Decomposition (ƒ 5), which can be ordered in the same way as a Working Paper.

Comments on some aspects of ProfGlot (explanatory notes and critical assessment)

Kwee Tjoe Liong

vakgroep Alfa-informatica & Instituut voor Functioneel Onderzoek van Taal en Taalgebruik (IFOTT), FdL UvA
(Dept. of Computational Linguistics & Institute for Functional Research into Language and Language Use,
Faculty of Arts, University of Amsterdam, NL)
T.L.Kwee@let.uva.nl

1. Introduction

1.1. Preamble

The book *Functional Grammar in Prolog* (Dik 1992) °

> " gives a detailed description of a computer program called ProfGlot, written in Prolog and using the theory of Functional Grammar in the version described in Dik (1989) ... [which] simulates some essential components of the linguistic competence of a [human natural language] speaker." (Foreword, vi)

Its aims seem to be twofold, in the first place theoretical-linguistic, but at the same time also computational, witness the following statements :

> " 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 ... . however, many grammatical and computational problems have been tackled through strategies which may well be useful to whoever is working on computational linguistic topics." (Introducing ProfGlot, 1)

This double objective may be the reason why it is not so easy to distinguish, in Dik's presentation of his system, the linguistic theory from the computational implementation. On the one hand, it is true that we are informed that

> " even within the constraints imposed by the linguistic theory (Functional Grammar) and the programming language (Prolog), many rules and principles [*that is, from FG*] can be formulated in different ways and according to different algorithms. This means that even rules [*that is, in Prolog*] which do work satisfactorily could be formulated in other, and perhaps better ways." (Introducing ProfGlot, 2)

but on the other hand we are also told that

> " at a number of points, programming ... has led to modifications, simplifications, and substantial improvements in the Functional Grammar formalism." (Introducing FG, 19)

As a result, it is difficult to exactly recognise on which points either of the two sides might have influenced the other side in some, possibly even crucial and decisive, way or another.

Such a situation, however, where a linguistic theory and a computer program are intimately entangled, is less than felicitous, since under these circumstances it will be very hard to discern essential aspects from accidental ones.

° Simon C. Dik, 1992. Functional Grammar in Prolog - an integrated implementation for English, French, and Dutch. Berlin : Mouton de Gruyter. Quotes from the book are given with abbreviated chapter titles. The relevant chapters are : Foreword, 1.Introducing ProfGlot, 3.Introducing FG, 4.Overall structure of ProfGlot, 9.UniGen, 10.UniExp, 13.UniPar.

Comments on some aspects of ProfGlot (explanatory notes and critical assessment)

comment — *noun* :
3. explanatory or critical matter added to a text
4. a note explaining or criticising a passage in a text
(from : Collins Dictionary of the English Language, 1979)

Contents :

—

Note. I am much obliged to the present and former members of the WPFG editorial team (Gerry Wanders, Lachlan Mackenzie, Peter van Baarle) for moral and technical support. Responsibility for everything in this paper remains of course solely and entirely mine.

1.2. Counterbalance, and plan

The present paper is basically intended as a critical assessment of some aspects of Dik (1992), in an attempt to clearly separate theoretical-linguistic matters from computational. It should be a warning to prospective readers of the book who, being knowledgeable about one of the two fields, either linguistic theory or computer programming, would like to learn a bit more about the other side of the coin. This is because there is some risk of misunderstanding, on the one hand, how to write programs for a given collection of grammatical rules, and, on the other hand, how to theoretically describe the construal of expressions of a natural language such as English.

Although the book has been written for theoretical linguists and computational linguists alike, it is highly probable that its audience will be less symmetrical than was intended, due to the existence of other, competing, linguistic theories, systems, or formalisms, several of which enjoy quite a reputation in the field of computational linguistics. For Functional Grammar to conquer an equally esteemed position, we fear, ProfGlot unfortunately fails its promotional goals.

The paper, therefore, is mainly addressed to functional grammarians, lest that linguistic community believe that ProfGlot really is the model to follow when it comes to programming the theory that is so dear to them all. In that context it is probably necessary, but at the same time certainly helpful, first to present and explain the ProfGlot system just as it is. As far as this explanatory side is concerned, knowledge of Prolog or of any other programming language is not required at all for an understanding. Elementary acquaintance with, or rather a feeling for, what is called the algorithmic approach will be sufficient. Details are explained along the way when needed, but never in too technical a fashion.

To outsiders who are curious to know more about Functional Grammar (the explanatory sections can serve as a quick and perforce incomplete introduction to the ideas of the theory before they embark on a deeper study of the system), I would like to indicate here in passing that *it ain't necessarily so*, that is, the theory need not be exactly like what they would gather from the book. That is what is argued in the first critical section where, once the set-up and the main lines of the central module of ProfGlot have been clarified, some of the most striking implications are treated from a theoretical point of view.

In the second critical section it is shown that specific claims as to the novelty of the parsing strategy of ProfGlot do not hold. Since theoretical linguists are likely to be less familiar with the computational background, pains are taken to prepare them as extensively as possible for the (rather succinct) argument, and in as generally understandable terms as possible.

Discussion of design and technical code is relegated to the last half of the last section, in a minor part which is the only subsection that is intended for more experienced readers and writers of Prolog.

1.3. Computational Functional Grammar

This is not a relative outsider's criticism. The task may be an ungrateful one, but I think the ProfGlot program should be taken seriously, not be ignored or laid aside after only a short look. In the end, my ultimate hope is for a better computational Functional Grammar. I do not pretend to know better than Dik what the theory of Functional Grammar is, of course, let alone what it should be. Some of the critical comments are based on a careful comparison between certain aspects of his computer realisation and previously stated explicit goals and principles of his linguistic theory, some on personal experience in writing a program that faithfully simulates grammatical rules and thus brought to light inconsistencies in those rules, and yet others on principles and guidelines that are inherent to a preferred view on computational linguistics. Such guidelines, external to the linguistic theory proper as they may be, were introduced into the Functional Grammar community a while ago by Connolly (1989) when he identified the following three stages in the transfer of human language-related knowledge to the computer :

(a) the externalisation of the knowledge
(b) the formalisation of the externalised knowledge
(c) the encoding of the formalised knowledge into computer-usable form

As a matter of fact, in the field of computational linguistics two perspectives can be found. In the above approach, human natural language is considered to be the object of study for the discipline of linguistics, and theoretical linguistic grammars as such are put on computer.

In another approach, the computer is taken as the starting point, being treated as a programmable automaton which can be made to display natural language behaviour. In this line, theories are very likely to be influenced by programs, not to speak of what in early and even not so early Artificial Intelligence is called the *working program attitude*, characterised by the slogan usually given as an answer to enquiries after the underlying theory (cf. Ritchie (1980), and also Winston (1977) as referred to by Kobsa (1987)) :

my program is my theory

I have often cited Connolly's three stages elsewhere, expressing as my opinion that a linguistic theory and a computer model of that theory are two separate entities which should always be clearly distinguished. In this view, then, the last of the three stages above (c) can only be taken up after completion of the other two (a and b), and these two stages (a and b) can in no way depend on the third one (c), let alone be replaced with it by way of a short cut. It is this line, where theories influence programs instead of the other way around, that I want to follow and to advocate, that is, computational theoretical linguistics as read from-right-to-left, or in the appropriate Functional Grammar format :

(ωx : linguistics(x) : theoretical(x) : computational(x))

1.4. Particular attention on two points

Of the various points in the ProfGlot system that may be open to doubt, only two procedures are discussed in depth in the present paper. They are, firstly, the fully specification of an underlying clause structure (or *specification* for short) in the universal generator module UniGen, and, secondly, the parsing or analysis of an input surface string (*parse* for short) in the universal parser module UniPar.

The treatment of each of these two topics is divided into two parts : first an explanatory section giving an introduction to the theoretical concepts and a survey of the module in question, and then a critical section in which specific suggestions or claims which can be found in the book are especially argued against.

In thus focusing the main discussion on these two points of Specification and Parse, other general criticisms of various programming aspects of design and code and their consequences with respect to the coverage of possible linguistic expressions and analyses are left aside. Only a few examples are commented upon in detail in the last section, as a kind of side remark.

Outline

In view of the main division into two topics, where each of them is split into two parts, the organisation of this paper can be seen with different directions in mind. As the printed version is only one-directional, the sections must be numbered in a linear sequence. Their interconnections, though, can be shown in a tabular form, as follows :

	generator	parser
summary & explanation	section 2	section 4
critical assessment	section 3	section 5

design and code :

linguistic aspects	section 6.1
computational aspects	section 6.2

As a result, readers may either just follow the linear sequence 2, 3, 4, 5 (*depth first*), or prefer to get an initial overall impression of the two most important modules of the ProfGlot system as it is, via 2, 4, 3, 5 (*breadth first*).

First of all, however, the general setting of the two procedures that are at the heart of these comments, Specification and Parse, should briefly be explained. That is what the next explanatory section will start with.

2. Summary of UniGen and UniExp

2.1. The ProfGlot system

In the summary which follows in the next subsections, readers are assumed to be acquainted with the standard outline of the theory of Functional Grammar (see Dik's (1989:53) diagram) and with the multi-layered model of the clause structure (Dik 1989:56-60, 67, 247-248), in short, with the following notions :

- the lexicon of basic predicate frames and basic terms
- formation rules for derived predicate frames and derived terms in the fund
- term structures
- operators and satellites
- term positions (argument positions and satellite positions), and term insertion
- predication schemas, and predications (nuclear, core, and extended)
- propositions
- clause structures
- function assignment (syntactic, pragmatic), and fully specified underlying structures
- expression rules (form, order, prosody), and linguistic expressions

The integrated system called ProfGlot that is described in the book under discussion consists of a certain number of components which are claimed to represent the different language faculties such as producing, understanding, inferring or logical reasoning, and translating. In the theory these faculties are postulated to be language-independent, and the system therefore has universal modules which it calls generator, parser, logic, and translator. For specific languages, a lexicon and specific expression rules are provided, but since a large part of the expression rules can be treated as language-independent, the system possesses a universal module for expression as well. The whole of ProfGlot covers three languages, English, French, and Dutch.

The underscores in the previous paragraph already hint at the module names. Of these, UniLog and UniTra play no part in the sequel ; therefore they can be skipped in this summary, as can the language-specific lexicon and expression rules. This leaves us UniGen, UniExp, and UniPar for a detailed presentation.

UniGen is the central component of the system. This is clear from the fact that chapter 3 in the book, with the title Introducing Functional Grammar, consists of only one long section, called Outline of the Functional Grammar generator. After this introductory preview UniGen is treated later on again in chapter 9, in full detail, while UniExp and the English expression rules are combined in chapter 10. UniPar is treated by itself in chapter 13. To an important extent it rests on notions already known from UniGen and UniExp.

In summarising these three modules of UniGen, UniExp, and UniPar in the explanatory sections I take the opportunity to add a couple of remarks.

2.2. UniGen up to the clause structure

UniGen starts with an arbitrary predicate frame, no matter whether basic or derived (details of predicate formation aside), of some category. A predicate frame contains a linguistic form, a type specification, and a set of (descriptions of) argument positions. For the sake of uniformity it should be noted that such a frame is considered to be a nuclear predication schema, or nucleus for short, where the notion *schema* indicates that argument positions are yet unfilled.

A nuclear predication schema is to be enriched with, on the one hand, specific predicate operators and, on the other hand, a set of (descriptions of) predicate satellite positions, both depending on the particular type of the given predicate (such as action, process, and so on). The result is a core predication schema.

The core schema in turn is enriched, first with specific predication operators, and then with a set of (descriptions of) predication satellite positions which not only depend on the particular type of the predicate but also on the previously specified predication operators. This leads to an extended predication schema.

The extended schema has as yet unfilled (hence the conspicuous *descriptions of* above) positions for arguments (in the nucleus) and satellites (of two kinds). It should be noted, however, that, next to a specific predicate frame, it also has sets of predicate and predication operators which already have been specified.

Remark : I think this asymmetric treatment of operators and satellites is rather remarkable. The difference between them in the theory seems to be accidental, in fact just formal and superficial. Their role in contributing to the semantic content is essentially the same, witness their definition :

> " Operators are used to capture those modifications and modulations which can be brought about at the relevant level by grammatical means ; satellites those that can be brought about by lexical means." (Dik 1989:50)

An extended predication schema is turned into an extended predication by the operation of term insertion, to be applied successively to each of the argument positions (of the nuclear schema), predicate satellite positions, and predication satellite positions, in this order. Term insertion requires a prior application of term formation, and involves a possible application of relative, interrogative, and anaphoric term operators, but that is skipped here for the sake of brevity (the treatment of anaphoric pronouns will be commented upon in section 3).

What should not be forgotten, though, is the fact that a term structure may contain extended predications, or even extended propositions (which are to be treated in a while). Why and how or where exactly such extended predications and propositions can occur inside a term structure is not so important at this moment ; just think of an embedded subordinate clause (inserted as a term at an argument position), an adverbial subordinate clause (inserted as a term at a satellite position), or a relative clause that restricts the nominal head of a term.

What is more important here is that all steps of UniGen up to and inclusive of the one which is treated next (immediately below) may have to be executed within a single application of term formation. But the most important point in connection with the very last step in UniGen, to be explained in a while in all its details, is just this fact that any term may contain one or more predications or propositions (which will contain one or more terms that may contain ... and so on and so forth, in short, we have a classic example of recursion here).

An extended predication is turned into an (extended) proposition by enriching it with a set of specific propositional (modal) operators, and a set of specific propositional (modal and/or attitudinal) satellites which not only depend on the type of the particular predicate (which is contained in the nucleus), but also on the yet unspecified illocutionary operators.

Note that propositional satellites, unlike those at lower steps, must be specified immediately, for they are too late for term insertion and it is needless to wait for another round of that operation (because there is no such second round) ; the solution is to assume that they can only be lexical items, and never terms. On the other hand, they must wait for illocutionary operators to be specified before they can be so themselves ; the solution to this is to assume a kind of propositional schema (at variance with the notion of a *schema* as was stipulated above), and to postpone the specification of propositional satellites till the next higher step, or to repeat it in due time (UniGen, 94-95).

Remark : yet another problem is left unsolved, for propositional schemas are not always made into clause structures ; some are used to form propositional terms, and it is unclear what happens to propositional satellites in such a case.

An extended proposition, finally, is made into a clause structure by enriching it with a set of specific illocutionary operators (which should now induce the specification of the propositional satellites), and a set of specific illocutionary satellites which again depend on the type of the particular predicate.

With the formation of the clause structure we have reached the pre-final stage of UniGen. The chain of actions up to and including this last step is started by invoking a single procedure which is appropriately named *clause-structure*.

A clause structure becomes a fully specified underlying clause structure, or a fully specified clause for short, after application of a procedure *specification*, the final step in the module UniGen. The two procedures *clause-structure* and *specification* together constitute the procedure *fully-specified-clause*.

The procedure *specification* deserves some further explanation, first as to its why, then as to its how, and finally as to its what exactly. Details in the system with regard to these three aspects, especially some design decisions, are rather debatable. After an ample exposition in the next subsections, they will be the object of the first critical section that will follow (section 3).

Survey of procedures : UniGen clause structure

In order to keep track of the connections between the separate procedures that are treated in the verbal summaries, a survey of the definitions as given in the book in Prolog code is offered every now and then at appropriate moments.

For obvious reasons the rather technical Prolog notation has been simplified and adapted here. Following traditional usage in program writing, however, annotations in the code are preceded by a special marker, the percent sign %, as a separator for comments.

These first steps in the generation process are conveniently enumerated in the book as well, on less than half a page (Introducing FG, 25).

```
% Note :
% Prolog notation has been simplified and adapted for this survey of the relevant procedures.
% Commentary annotations are preceded by a special marker, the % sign.

core-pred-schema =
    arbitrary-pred,
    predicate-operators, % specify operator ProgrAspect
    sats1. % prepare term positions for, or lexically specify, satellites Man, Instr, Ben, Dir

ext-pred-schema =
    core-pred-schema,
    predication-operators, % specify operators Tense, PerfAspect, Polarity
    sats2. % prepare term positions for, or lexically specify, satellites Loc, Temp, Pol, Cogn

ext-pred =
    ext-pred-schema,
    do-all-terms(ins). % perform term-insertion at all term positions

prop = % branching off from main line, necessary for formation of propositional terms
    ext-pred,
    prop-operators, % specify operator Attitude
    sat3. % lexically specify propositional satellite

clause-structure =
    ext-pred,
    illo-operators, % specify operator Illocution (declarative or interrogative)
    prop-operators, % % specify operator Attitude
    sat3, % lexically specify propositional satellite
    sat4. % lexically specify illocutionary satellite

fully-specified-clause =
    clause-structure,
    specification. % or, equivalently, specify-clause
```

2.3. UniGen from clause structure to fully specified clause

Since the procedure *specification* is the topic of the critical section that follows after this explanatory one, its further treatment is rather detailed. For reasons of organisation it is put here, but it may be skipped at a first quick reading.

Specification, term-specification, clause-specify

As announced, the procedure *specification* deserves some further explanation in connection to the questions 'why', 'how', and 'what exactly'.

As for its motivation in the theory, the question is simple to answer, for there is more informational content in the observable linguistic expression than can be found in the clause structure as construed to this point. Since it is postulated that abstract underlying clause structures be complete in this respect and that expression rules cannot add anything more to their informational content, the difference should yet be accounted for. As for its motivation in the system, or rather of its place, or moment, in the chain of steps in the system as described, the question is much less simple to answer, especially when it is linked to the question of its 'what', and I shall not try to find an answer here, as the point is a bone of contention. The question of how to execute the final procedure is of course also closely connected to the exact details of what should be specified. This point is also open to discussion, and I shall return to this one in due time as well.

Let us now just look at the 'what exactly' as it is designed and realised in the system. The contents of the final procedure can be seen as a set of relations that should be specified between, let us say, a structure and its constituent parts (in the last resort, the structure here is the clause structure as a whole). One may think, for instance, of such relations as syntactic and pragmatic functions. The exact contents of this collection as realised in the ProfGlot system will be revealed shortly. For ease of reference I prefer to give the provisional label A to this specifying action, since it is not identical to the procedure *specification*. The reason for this will also reveal itself in a short while.

The procedure *specification* as defined in the ProfGlot system only involves the extended predication on the inside of the complete clause structure built by the procedure *clause-structure*. We have the following situation, and it should be clear, of course, what is meant by the informal use of "its" in the last line :

> performing *fully-specified-clause* in order to have a fully specified clause is defined as
> - performing *clause-structure* in order to have a clause structure,
> - applying *specification* to its extended predication. (cf. UniGen, 99)

In the simplest case, the intended relations between the extended predication and its terms could now be specified by means of the action A, but remember that each of the terms may contain one or more predications or propositions, so the action A must be applied to those structures as well, in a recursive way.

This calls for a careful approach. It will be helpful to think of the operation of term insertion, as introduced in a previous section, which is successively to be applied to each of the argument positions (in the nuclear predication schema), predicate satellite positions, and predication satellite positions, in this order. Here we have the operation of what is called *term-specification*, to be applied to terms in an extended predication in exactly the same order as term insertion is applied to term positions in the extended predication schema (note here that in the code of ProfGlot the full name *term-specification* is shortened to *tspec*, just like the name of the operation of term insertion is shortened to *ins*).

But *term-specification* is only an abbreviatory name. It doesn't mean that some term should be specified, but that *specification* should be applied to extended predications which possibly occur within the term structure in question. Such predications within a term may occur in one of two ways. Either the term is itself a propositional or predicational term, and then an extended predication can be found at or within the first restrictor position ; or else the term may contain an extended predication at its fourth restrictor position, as a relative clause. To complicate matters a bit further, terms may have an attribute in the form of an adpositional term at the third restrictor position, and this term may contain yet other predications or propositions.

In summary, the 'what' of the final procedure, that is, the action A in which various relations between a structure and its terms must be specified, should be applied not only at the local level of the extended predication itself, but also at all lower levels of extended predications that are possibly embedded in some way or another within its terms. This is done in a recursive fashion. With such a recursion, in general it doesn't matter when it is done (as long as it is done), either before or after performing the operation at the local level. In the case at hand of the particular specifying action A, however, application at embedded levels should precede application at the local level, in view of such phenomena as raising (an embedded subject getting a matrix object or subject function).

The anonymous specifying action known hitherto under the working title of A is called *clause-specify* in its ProfGlot realisation. Note that it still applies only to an extended predication, and not to a proposition, as erroneously remarked at a certain moment (Introducing FG, 28). We can summarise everything in the following three definitions (cf. UniGen, 99-100), after which we are fully prepared to study the specifying action *clause-specify* in detail.

applying *specification* to an extended predication is defined as
- doing *term-specification* in the prescribed way with all its terms,
- applying *clause-specify* to the extended predication at its own level.

applying *term-specification* to a predicational or propositional term is defined as
- applying *specification* to the extended predication at or within its first restrictor.

applying *term-specification* to a non-predicational-or-propositional term is defined as
- applying *term-specification* to the adpositional term at its third restrictor (if present),
- applying *specification* to the extended predication at its fourth restrictor (if present).

What clause-specify should do

As remarked at the beginning of this section, *specification* should make up for the difference in informational content between the linguistic expression and the clause structure as construed in the procedure *clause-structure*, for it is postulated in the theory that expression rules cannot add anything more to the informational content contained in abstract underlying clause structures.

For the sake of a proper treatment of embedded structures possibly contained in the overall clause, *specification* is split into a recursive part and a local part. The local part of *specification* is named *clause-specify* [*sic*]. Notwithstanding the last name, and contrary to what is said in the book, both operations apply to an extended predication. Not only may the choice of the various names used in the ProfGlot system sometimes leave an impression of arbitrariness, but the explanatory text is sometimes also rather sloppy as regards the terminology, which might be another source of confusion. I hope that the (partial) summary at least of UniGen and Unipar offered in the present paper will be helpful to prospective readers for an easier comprehension of these two components.

What, then, should a procedure such as *clause-specify* achieve at its local level, or, in other words, what should be the content of what I formerly called the specifying action A ? As already said, this content can be seen as a collection of relations (such as, for instance, syntactic and pragmatic functions) between a structure and its constituent terms, relations that can be distinguished in the observable surface form but are as yet absent in the structure in question.

As for the syntactic (also called *presentative*) functions (subject and object), Dik (1989:53, 58) suggests that they be assigned at the layer of the extended predication, after introduction of the predicational operators and satellites, and this corresponds to Hengeveld's (1989) representational level of analysis.

As for the pragmatic functions (topic and focus), in the context of the revised model with its multiple layers, Dik's (1989:53, 60) suggestion is that they be assigned at the highest layer, that of the complete clause structure, after the introduction of the illocutionary operators and satellites. This can be identified with Hengeveld's interpersonal level of analysis.

These considerations would imply a two-stage design for the specifying action A, first a syntactic AS at the extended predication and then a pragmatic AP at the clause. As for the realisation in ProfGlot, we are informed that

> " [o]ne 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 ... functions into the system is one of the most important points on the agenda for the further development of the ProfGlot program." (Introducing FG, 30)

This might be the reason why *specification* and *clause-specify* are only applied to extended predications, for pragmatic function assignment is not involved.

What clause-specify does

The procedure *clause-specify* as designed and defined in the ProfGlot system, applied at the layer of the extended predication contained in a clause structure, thus does less than what we would have expected on the basis of the theoretical considerations, but also more, since it performs tasks of three different kinds :

> " The operations in the specification concern such matters as how the State of Affairs as designated by the predication is going to be presented, what types of coreferentiality can or must be established between the terms in the clause structure, and how the clause structure can be prepared for entering the expression component." (UniGen, 100)

It does so in the following way :

> applying *clause-specify* to an extended predication is defined as
> step-by-step changing the relevant elements by sequentially applying the operations of
> - *subj-obj-assignment*,
> - *verb-agreement*,
> - *anaphora*,
> - *reflexive*,
> - *equi*,
> - *copula-support*. (cf. Introducing FG, 28-29 ; UniGen, 100)

The six operations involved here do not fall neatly into the three categories as cited above, but combine goals of different types. Usually the third category, preparation for expression, is combined with one of the other two. In the first operation, *subj-obj-assignment*, for instance, not only the term(s) in question (chosen only from argument terms in the nuclear predication part) are marked for 'subj' or 'obj', but also the predicate is marked for 'act' or 'pass', because

> " [t]his will later inform the expression rules about voice expression." (UniGen, 101)

Two operations exclusively prepare for later expression, *copula-support* (by 'copula-introduction' to the non-verbal predicate), and *verb-agreement* (that is, agreement in Person and in Number between finite verb form and subject term, and also in Gender as far as French is concerned). Only for the latter operation is an alternative method pointed out :

> " There are different ways in which such agreement could be captured. One way is to leave it to the expression rules to find the relevant agreement parameters in the clause structure. Another way, implemented here, is to inform the predicate (more particularly, the tense operator on the predication) of the relevant features. The expression rules will then find the relevant information 'locally'." (UniGen, 103)

The remaining three operations relate to coreferentiality. The first one is said to account for the resolution of previously inserted 'free' anaphorical terms and will thus provide a basis for the other two, which pertain to refl-marking and equi-substitution.

The Functional Grammar treatment of coreferentiality and anaphora is further discussed in the critical section after this summary.

2.4. UniExp

The theory postulates that expression rules just map fully specified underlying structures onto linguistic expressions, and don't do anything else. Of the three types of expression rules which are distinguished (form, order, and prosody), the third type has not yet been implemented in the ProfGlot system.

At first sight, form and order rules can be defined independently from each other, but in many languages it turns out that subtle interdependencies occur. UniExp, therefore, is designed as a three-stage module, covering not only the formal expression rules and the placement rules (which put the constituents in a linear order), but also sandhi rules (incorporated into the placement rules) for 'low level' adjustment of word forms in their linear sequence. The whole set of expression rules is performed in a single procedure *ex2-clause*, which in accordance with its name consists of two parts (Introducing FG, 29-30) :

(1) *formally-ex-clause* which reorganises a multi-layered fully specified clause structure (containing functions, operators, satellite terms, argument terms, and a predicate) into a combination of two sets, one consisting of the illocutionary operators and one of three sets of hierarchically organised word forms : one for the verbal complex, one for the full-terms, and one for the satellites at the four layers. The verbal complex (this name also holds in the case of a term predicate) is determined by the predicate in combination with the operators at all four layers and the syntactic functions. The full morphosyntactic form of each argument term is determined by its set of term operators and its semantic and syntactic functions. The same holds for each satellite term.

(2) *full-place* which proceeds in four steps :
• it fills a pre-set list of sixteen positions (that is, a template) with constituents taken from the above-mentioned three collections of word forms (that should thus be exhausted) ;
• it adds a final punctuation (depending on the illocution) ;
• it removes all boundaries between constituents in order to get a linear string of words (or, in Prolog jargon, a flat list of atoms) ;
• finally, it performs the necessary 'low level' formal adjustments described above (of word forms in their linear sequence) in what so far is no more than a kind of annex to the list flattening, an auxiliary procedure named *sandhi-list*.

Clearly, the procedure *formally-ex-clause* will depend heavily on the lexicon and inflectional paradigms which are language-specific, as will the procedure *sandhi-list* that, for the time being, is incorporated in the procedure *full-place*. On the other hand, the proper action of the procedure *full-place* itself is based on less language-specific ordering templates (also called functional patterns in the original, pre-1989 version of the theory) which are assumed to hold across languages of the same family. The theoretically important principle of LIPOC, or the language-independent preferred order of constituents ('more complex constituents tend to be placed later'), is not yet taken into account, however.

Survey of procedures : UniGen fully specified clause, UniExp

fully-specified-clause = % (UniGen 99)
    clause-structure, specification. % or, equivalently, specify-clause

specify-clause = % (UniGen 100)
    specification. % apply only to the extended predication within the clause structure

specification = % totally specify an extended predication (UniGen 99)
    do-all-terms(tspec), % perform term-specification at all term positions
    clause-specify. % locally specify the extended predication, at its own level

perform(tspec) = % do term-specification in case of a predicational or propositional term
    specification(R1). % specify the ext. predication in restrictor1 position (UniGen 100)

perform(tspec) = % perform term-specification at a term position (UniGen 99)
    perform(tspec,R3), % at inside term in restrictor3 position (= attr. adpositional term)
    specification(R4). % specify ext. predication in restrictor4 position (= relative clause)

clause-specify = % (UniGen 100)
    subj-obj-assignment, % at same time, add Voice to form of predicate in nucleus
    verb-agreement, % add Num, Pers, Gender to predication operator Tense
    anaphora, % anaphora-resolution, adjustment of referential term index
    reflexive, % reflexive-marking
    equi, % equi-substitution, equi-marking of term that should get zero expression
    copula-support. % copula-introduction

ex-clause = % express [*sic*] a clause (UniExp 143), by :
    fully-specified-clause, % first forming a fully specified underlying clause structure
    ex2-clause. % and then applying the expression rules to this underlying structure

ex2-clause = % apply the expression rules (UniExp 143), by :
    formally-ex-clause, % first determining the morphosyntactic form of all constituents
    full-place. % and then putting these in a template which is made into a list of words

formally-ex-clause = ex-verbal-complex, ex-full-terms, ex-sats. % (UniExp 130)

ex-verbal-complex = ex-voice, ex-progr, ex-aspect, ex-attitude, ex-polarity,
    ex-illo, ex-tense, flatten. % (UniExp 124)

go = % the top level instruction for production = generation and expression (UniExp 144)
    repeat, % perform the following procedures in order to produce one clause
    ex-clause, % express a clause as a list of words
    adjust-spelling, % tidy up the list for occurrences of the schwa (only for Dutch)
    writelist, newline, % output the list of words
    fail. % *da capo*, and "continue repeating ... until no more sentences can be formed" [*sic*]

% Note. 'repeat, X, fail' is a special Prolog construction for renewed performances of X
% which ensures endless repetition unless interrupted by an external action (see also section 6).

3. Comments on some aspects of UniGen : specification

3.1. Clause-specify : combining three different goals

Recall that the various operations collected into the procedure *clause-specify* serve three different goals, witness the description cited in the summary in section 2.3, and repeated here :

> " The operations in the specification concern such matters as how the State of Affairs as designated by the predication is going to be presented, what types of coreferentiality can or must be established between the terms in the clause structure, and how the clause structure can be prepared for entering the expression component." (UniGen, 100)

It is not quite clear whether the author of the book at this point assumes the programmer's role and just gives a description of the working of his ProfGlot system, or speaks in his capacity as a theoretical linguist and purposefully delivers a statement of intent.

With respect to two of these three goals, however, I have strong objections.

In the very first place, it is my contention that coreferentiality between terms is first and foremost a matter of semantics and therefore cannot be established afterwards, but, on the contrary, should precede term formation and insertion. Reference to entities is essentially at the basis of, and leads to, the formation of term structures. At most it can be incorporated into this procedure, but it can never follow it. Use of anaphoric personal pronouns is, furthermore, largely a matter of surface constituent order and morphosyntax, and therefore it clearly belongs to the expression rules. This all is argued for in section 3.2.

In the second place, preparing the expression component does not, in my view, belong to the specification of the abstract underlying structure but should be incorporated into the expression component itself, or may at most precede that component as an intermediate stage between, on the one hand, the construction of a fully specified clause, and, on the other hand, the linguistic expression of that abstract underlying structure in what can be called (and in other theories is indeed called) a surface structure. It is not a particular programmer's point of view, as it is motivated by two independent considerations, redundancy and morphosyntactic language-specificity, to be elaborated in section 3.3.

In this way, of the three goals cited above only one goal can be maintained, in a strict conception of the theoretical principles. This third goal, specifying the presentation of the State of Affairs as designated by the predication, rightly belongs to a specification procedure. But objections can be raised here as well, namely with regard to the contents and the moment of this specification. These objections are of a mixed nature. They pertain both to linguistic theory and to computational design. Because of the strong interrelationship between the two aspects, at least as far the assignment of syntactic functions is concerned, they are treated together in section 3.4.

3.2. Anaphora

The treatment of anaphora

Anaphoric pronouns, and anaphora in general, are the topic of an everlasting debate in all linguistic theories, as well as in logico-philosophical approaches.

The treatment proposed by Dik (1980a) is maintained essentially in unchanged form by Dik (1989), and in the ProfGlot system with only just a few revisions that seem to be a refinement of minor details. It is based on the application of an anaphorical term operator during term formation and term insertion. Dik (1980a) at least puts a condition of antecedent retrieval on the application of the anaphorical term operator, and requires co-indexing of the latter with the former, but in the computational system such conditions now have completely been left out.

In ProfGlot, during the operation of term insertion (see section 2.2, summary of UniGen), instead of a co-indexed anaphorical term operator being applied, not only can an ordinary third person personal pronoun be inserted, as a basic term, but it also gets an arbitrary index, as is the rule with all inserted terms except those which refer to the Speaker or the Addressee (UniGen, 86-87). Afterwards, though, this index may be adjusted in the step of *clause-specify* which takes care of anaphora resolution, but only on the following conditions :

> " [term] A has subject function and is not a first or second person pronoun term ... and B [not identical to term A] is a 'free' anaphorical term in a term position of which the selection matches the type of the subject term. If so, the anaphorical term inherits the Number, the Type, and the referential index from the subject term. If these conditions do not hold, the input predication remains unchanged." (UniGen, 104)

If the conditions do hold, there is a subsequent possibility that the co-indexed anaphorical term must be expressed as a reflexive pronoun, or requires zero expression, as in

> [refl] Who_i would kill himself_i for such a reason ? (UniGen, 87)
> [equi] John_k wanted $[]_k$ to go to the city. (UniGen, 106)

If the conditions do not hold, nothing is changed. Admittedly,

> " [t]he treatment of anaphora is rather rudimentary in the present program. ... ... The last point implies that there will be 'unresolved' anaphorical terms in many underlying structures." (UniGen, 104)

but there seems to be a way out of this embarrassing inconvenience :

> " [A]n anaphorical term which has no antecedent in the same clause may have one in some preceding clause in the discourse. Resolution of such discourse anaphora is not yet possible within this program." (UniGen, 104)

> " The 'unresolved' anaphorical term is therefore expressed as 'nn', awaiting further development of the program." (UniGen, 117)

The notion of anaphora

This treatment of anaphora, indeed the use of the word *resolution*, together with the subsequent steps of *reflexive* and *equi*, bears a strong resemblance to transformational approaches in general, and especially to the school known as *interpretive semantics*, because of its marking of basic (that is, lexically given) personal pronouns and later adjustment of referential indices, and because of the operation of equi-substitution that will lead to zero-expression. It seems to flatly contradict the semantic-pragmatic-functional foundations of Functional Grammar, where in fact we would expect such phenomena as coreferentiality, reflexivity, and 'equi' to be given as informational content (*ab initio*), rather than to result afterwards from an operation of anaphora resolution.

Anaphora resolution means finding an antecedent for a given anaphorical term such that the two of them can be co-indexed, that is, can be linked syntactically and interpreted semantically as coreferential. Anaphora resolution thus is part of analysis and interpretation, and as such completely different from anaphora generation, where an 'antecedent' is given, instead of being looked for.

In generation, reference is intentional, and therefore co-reference is intended, and in that case co-indexing is required rather than a possibility. Expression of coreferentiality, however, is to a large extent determined by the linear order of words. The use of a personal pronoun as an anaphorical term at a certain place in the sentence depends on whether it can be interpreted (in other words, understood by a Hearer) as coreferential with the proper antecedent. If it can, it may be expressed in that way, but otherwise the Speaker should use another form to express the intended information, or else a misunderstanding results.

In the given implementation antecedents are required to bear subject function, and anaphoric pronouns to occupy a position in the same extended predication, in order for them to be able to be co-indexed. But if their Type and Selection do not mutually match, the unresolved pronoun may have an antecedent in the wider discourse structure, or so it is hoped. The fear is legitimate that such a yet-to-be-developed discourse resolution will not always succeed either. The only possibility which remains, then, is to consider the pronoun at issue to be a deictic element. This is again an interpretive, not very satisfactory approach.

The approach to anaphora as realised in ProfGlot may have other problematic consequences with respect to processes executed in between the two moments of term insertion of pronouns and possible adjustment of indices afterwards in resolution. Constraints on the assignment of syntactic and pragmatic functions, for instance, may become rather complex when under certain circumstances it is less desirable that a personal pronoun gets subject or focus function (unless it were eventually interpreted once again *via* the above escape rule as deictic). This drawback does not arise in a treatment of anaphora which is not based on an early insertion of arbitrarily indexed basic pronouns followed by resolution in combination with a transformation-like adjustment of indices later on.

An alternative approach to anaphora

To summarise very briefly, in generation we have an entity to which we want to refer more than once, and in the linguistic expression we have to do so with the help of the proper morphosyntactic means, that is, sometimes we may use an anaphoric pronoun. But if we start with a given anaphoric pronoun and we are looking for an antecedent with which to properly co-index it, we speak of resolution, and then it is analysis and interpretation that we are involved in.

Apart from the ideological point, there is a methodological side as well. When reference is made to the same entity at two different positions in an underlying structure, it is not known in advance which of the two terms will be expressed first. Kwee (1987, 1988b, 1994, to appear) argues that anaphoric pronouns are merely surface phenomena. Any use of pronouns other than deictic ones will depend on word order, relative distance, and various other conditions (such as syntactic and pragmatic functions) which cannot be known yet at the moment of term insertion. Therefore such pronouns are out of the question in an abstract underlying structure. In the publications mentioned, an alternative method for term insertion, indexing, and coreferentiality is presented which, although it is a computationally motivated contrivance without any theoretical claims whatever, yet renders more justice to the functional point of view that in generation semantic and pragmatic contents in the underlying structure have priority over morphosyntactic expression in the surface string. It is based on the use of an auxiliary structure (a global register on which terms are put as soon as they are formed and have got a (new) index) and two computationally motivated tools (or programmer's tricks), pointers and flags, or markers. For the shortest exposition of this method, illustrated with an example, readers are referred to Kwee (to appear). Here just the barest essentials can be sketched.

Only shadow terms are used for insertion at (first order) term positions. Each shadow term, however, is co-indexed with some registered (first order) term. Thus, the possibility arises to loosen the strict ties between term formation and insertion, which could lead to a more discourse-oriented view on generation in Functional Grammar. Shadow terms are nothing but placeholders, or pointers to registered terms. The global register can be seen as representing the domain of discourse referents. Each registered term is provided with a binary on-off marker (or 'flag') which serves as an indication whether the term has already been expressed in the course of the linguistic expression, or has not yet been. Various strategies based on this flag can be followed during the application of the expression rules, for instance the principle called *first-come-first-served.* When a shadow term at some position in an underlying structure is going to be expressed, its associated, co-indexed, registered term is inspected, and if that term has not been expressed before, it must be expressed at this instance, while at the same time its marker is switched off. Any shadow term with the same index possibly met later on in the course of expression, finds the marker 'off' and is expressed as a pronoun. This is only a first approximation to anaphora. It needs refinement, but it is hoped that it is a step in the right direction.

Remarks on reflexive and equi

The remaining phenomena which are treated in ProfGlot in connection with anaphora and personal pronouns are reflexive-marking and equi-substitution. In my opinion the two of them are also more a matter of semantic content than of morphosyntactic form. They could better be handled at a much earlier stage in the overall process of generation as well.

Maybe the best way of handling reflexive, and also reciprocal, pronouns is by derivation rules during predicate formation, similar to the treatment of verbal reflexives described by Dik (1983).

The same holds for the equi-marking operation as defined in ProfGlot, which strongly resembles a transformational treatment. One gets the impression that an approach in which the following two sentences are produced along the same way but for the last step of equi-marking, which is obligatorily applied in the first case but cannot be applied in the second case, is rather questionable. The more so where equi-marking needs a preliminary and obligatory application of subject-to-object-raising in function assignment, while the latter operation is only optional for the second sentence.

[equi] John_k wanted $[]_k$ to go to the city. (UniGen, 106)
John_k expects $[\text{Bill}]_j$ to go to the city.

The equi-construction is probably restricted to a small class of matrix verbs. Such verbs could as well undergo a rule of predicate formation in which they would then receive the desired complex predicate frame with the appropriate argument positions, similar to the treatment of Dutch causative and permissive constructions described by Dik (1980b).

A last remark on random term-indexing

A last remark should be made in relation to anaphora, namely with regard to the process of term indexing. As for the arbitrary referential index that every term gets at the time of formation and insertion, in UniGen this arbitrariness is realised in a random process that produces a positive integer below a certain pre-established ceiling, say, 25 (UniGen, 86-90). It should be noted that it is quite possible in the given version of ProfGlot for two or more terms (while not being anaphorical elements) to unintendedly get the same referential index. This does not depend at all on the height of the maximum value but is inherent to the chosen procedure of random number generation, known by its technical name of 'drawing with replacement'.

In the alternative approach mentioned above, with a global register for terms and shadow terms inserted at argument and satellite positions, term indices are assigned to terms, at the moment of their registration, in the stepwise growing sequence of natural numbers, and we thus have a quite natural enumeration of newly formed terms, without the risk of unwanted, accidental co-indexing.

3.3. Preparing expression

Surface form of verbal predicates

Various surface phenomena connected to the form of the (verbal) predicate have crept into the specification procedure that is assumed to be the finishing touch to the abstract underlying structure, such as voice, and agreement, and the introduction of a copular verb as a support to a non-verbal (either nominal or adjectival) predicate.

With regard to verb agreement, the author of ProfGlot at least recognises the possibility of an alternative approach (cited in the summary in section 2.3) :

> " There are different ways in which such agreement could be captured. One way is to leave it to the expression rules to find the relevant agreement parameters in the clause structure." (UniGen, 103)

In my view, that is exactly the way it should be done. The abstract underlying structure should not be overloaded with information which is already present in some way or another, even if it is only implied. Redundancy of course is a very nice and useful property to have for messages in general, and *a fortiori* for linguistic expressions in particular, but as a matter of principle in abstract underlying structures it is, literally, superfluous.

Voice

The addition of Voice to the verbal predicate during *subj-obj-assignment*, for instance in the form of either one of the markers 'act' or 'pass', is redundant, because voice is implied by whether the first argument term has or does not have subject function.

Verb-agreement

Likewise, verb agreement does not need any markers either. It can be derived from information that is already present in the abstract underlying structure, and therefore it should properly be dealt with in the expression component, the more so since verb agreement is, to a very large extent, language specific.

In French, for instance, the verb not only has to agree with the subject term in Person, Number and Gender, as has been implemented in ProfGlot (UniGen, 103) — note, however, in the following example (apart from the less well-chosen preposition in the expression of the Source), the oversight with respect to the third feature of this subject-verb-agreement rule :

> cette voiture a été acheté [*sic*] de [*sic*] Pierre (So) par Jean (Ag). (UniExp, 142)

— but it must also agree, under certain circumstances, in Number and Gender with the object term (apparently completely overlooked in ProfGlot), as in :

> les lettres que je t'avais écrites, les as-tu jamais reçues ?

Copula-support

With copula support we have arrived at the third case of preparation for the expression component. The same reasoning as above applies here as well. It is not only redundant, the need for copula introduction being derivable from the category of the predicate, it is also language specific, for not in all languages is a copular verb required, and in several languages which do require such a verb in general, it may not always be required under all circumstances.

Dik's (1989:166) theoretical description of the rule of copula support is not wholly conclusive, and rather confusing as to its status, as if the author hasn't yet definitively made up his mind about it. On the one hand, the copular verb figures in underlying structure, on the other hand, it is also introduced by an expression rule. When Dik compares copula support to the rule of *do*-support in English grammar, that would indeed indicate that copular verbs are absent from abstract underlying structures and appear only at the surface level of the concrete linguistic expression. Such, then, is the general conclusion arrived at by Kwee (1994:147-148) :

> " Supportive elements are morphosyntactic devices that have no semantic or pragmatic content by themselves."

In his impressive work on non-verbal predication, Hengeveld (1992:33-34) likewise doesn't hesitate at all about the theoretical status of copular verbs, but decidedly considers them to be introduced by an expression rule.

Minimum redundancy and minimum language dependency

In summary, as far as the borderline between an abstract underlying structure and its surface linguistic expression is concerned, two guiding principles can be identified.

The first principle is that redundancy should be avoided as much as possible in an abstract underlying structure, because there is really no need at all for it to be there.

The second principle is that observable morphosyntactic features which differ from language to language should equally be avoided as much as possible in an abstract underlying structure.

The language-specific parts in a Functional Grammar comprise the lexicon of basic predicate frames and terms, the derivation rules for predicate frames in the fund, and the expression rules, but the construal of an abstract underlying structure, with the help of the fund (including the lexicon), is the task of the universal part of the generation module (UniGen). This universal part should, to the largest possible extent, have a language-independent character, and leave all language-dependent elements of a specific morphosyntactic nature to the language-specific expression component.

3.4. Function assignment

As was explained before, in UniGen a fully specified clause is produced in two stages, first the construal of a complete clause structure, then the specification of various details at the level of its extended predication. The latter action is recursively applied to predications that are possibly embedded in satellite and argument terms of the extended predication in question, and only after that to the extended predication itself, at its own local level.

As the ProfGlot system is now designed, this specification action gives rise to a variety of questions. Why is it only applied to extended predications ? And why does it inspect argument and satellite terms within extended predications only ? And why, if it is applied to extended predications, is it invoked only at the very end, at the highest level, that of the clause structure, instead of at the relevant moment and level of construal, that of the extended predication ?

If we assume for the moment that specification should indeed be invoked at the level of the clause structure, the first two questions affect the layers above the extended predication. The last question concerns fundamental design decisions.

Function assignment involves pragmatic functions too, and we expect those to be assigned at the interpersonal level, at the highest layer, the clause structure itself. These functions are as yet not accounted for in ProfGlot, however. As soon as they will be in a further extension of the system, the local specification should be split into a part that applies at the level of the extended predication, and another part that applies at the level of the clause. Although this extension can be accomplished in a simple and straightforward way in the present design it would have been wiser to anticipate this development, as a kind of reminder, by defining the procedure at issue, specification, already as applicable to the clause structure *in toto*, instead of as applying only to the extended predication that is contained within the clause.

The next question seems to pertain to a more serious omission. In the present version of ProfGlot, at least, it looks as if embedded predications can occur at most as second order (predicational) satellites, since all third and fourth order (modal, attitudinal, and illocutionary) satellites ('probably', and 'frankly') are given as such in the lexicon. The fact that the procedure of term-specification is performed on argument and satellite terms within an extended predication and cannot reach such third or fourth order satellites thus has no consequences for them. This is completely accidental, however, for it is not very difficult to adduce examples of predicational terms as satellites of propositional or higher order, such as 'as far as I know', 'as we have understood', or Dik's (1989:259) very own examples 'since you are interested' and 'in case you haven't heard'. As soon as satellites of this kind are allowed for (and this is less far-reaching an extension to the system than the one referred to above), they should also be specified for syntactic and pragmatic functions, and in that case the procedure in question must be made applicable to the entire clause structure.

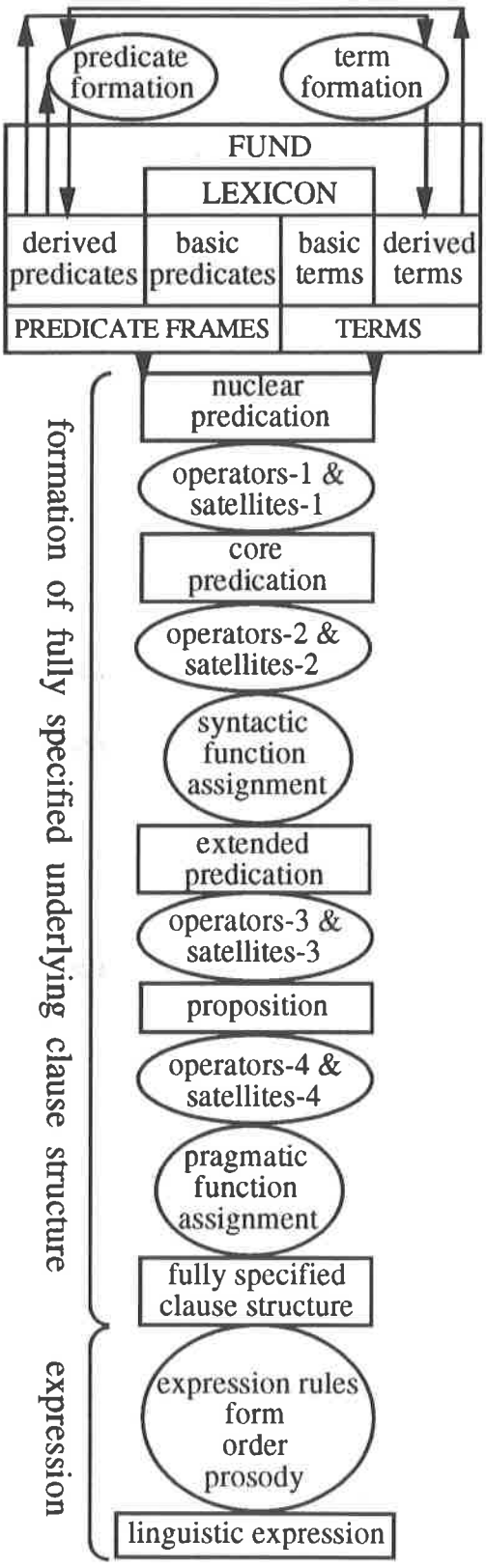
Program design and theory design

It should be noted that the observation made in the previous paragraph as to the possibility of predicational terms as satellites of third or fourth order, now has a further, and rather unexpected, consequence concerning the procedure of term insertion that is applied at the level of the extended predication schema (see section 2.2) in the same way as the recursive part of specification, hidden in what is called term-specification, is applied to an extended predication (see section 2.3). In ProfGlot, once the existence of predicational terms as satellites of third or fourth order is recognised, their formation and insertion cannot be handled as such, due to the design of term insertion in UniGen.

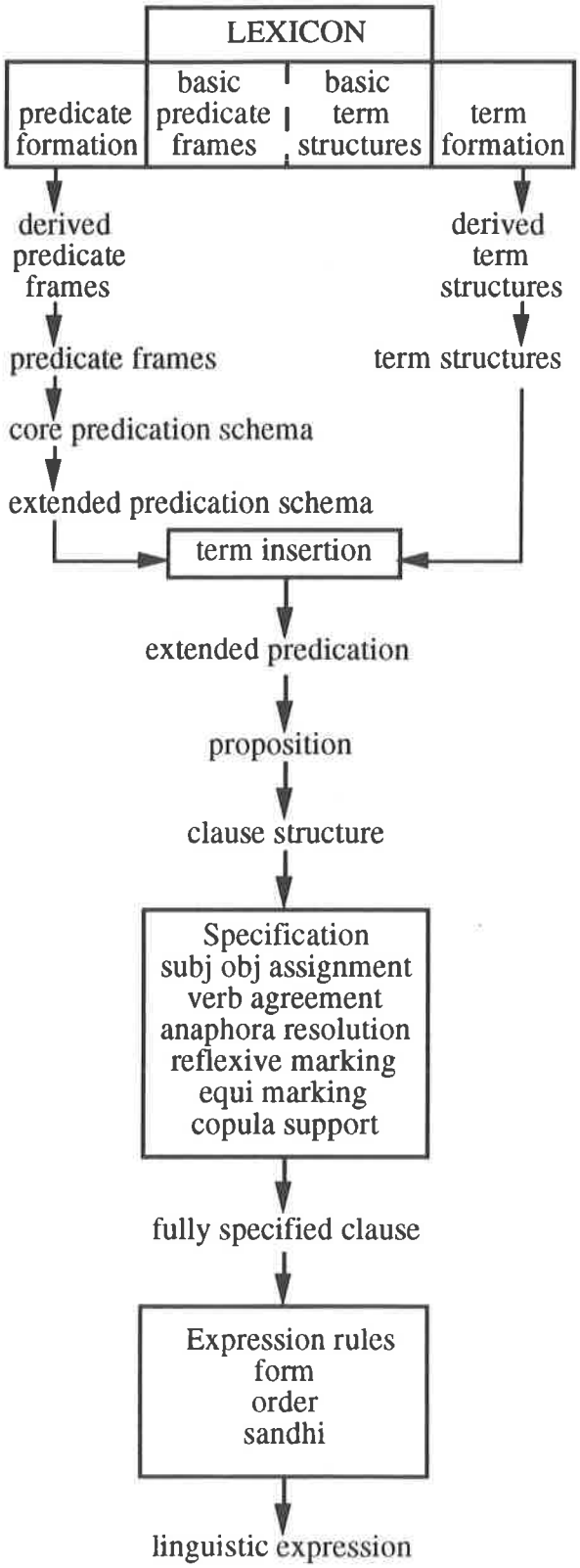
As announced before, the third and last question ('why is specification invoked at such a late stage, and only at the level of the complete clause structure') is more related to decisions taken in program design than to grammatical theory. It is interesting, though, to look at some minor but subtle differences between two versions of a diagram that is well-known to Functional Grammarians. In comparing Dik's (1989:53) *Outline of the Functional Grammar model* with Dik's (1992:20) *Lay-out of the Functional Grammar generator*, we wonder not so much about the absence of pragmatic function assignment (for that is excused by the state of development of the ProfGlot system, see section 2.3), as about what exactly would have caused syntactic function assignment to be moved from the completion of an extended predication to the completion of a clause structure, and about the significance that this change would have for the linguistic theory.

It is especially here that we may get confused because grammatical theory and computer program are not explicitly and clearly kept apart, far more than was the case with the procedures related to preparing expression. There at least the possibility of an alternative approach is recognised as far as verb agreement is concerned, but here no such hint is given. Is it another example of a rule that 'can be formulated in different ways and according to different algorithms' or would it be one of the points where it is claimed that 'programming has led to modifications, simplifications, and substantial improvements in the Functional Grammar formalism' (see section 1.1) ? In other words, is it a matter of a less felicitous program design in UniGen that, however, leaves the original theory intact, or do we have here a genuine example of a principled program design that has led to a substantially improved theory redesign ? In the former case we could easily think of an alternative program design, for instance, one in which the operation of subj-obj-assignment (but without voice-marking) must immediately be invoked at the level of the extended predication, irrespective of any later embedding into a propositional schema or a term structure. But in the latter case no clear motivation is provided for such a change, and no other one can be found than just particular programming convenience.

Programming aspects are relegated to the last part of this paper, where design and code are briefly discussed in a slightly more technical vein.



Outline of the FG model
(Dik 1989:53, Diagram 1)
(minor omissions corrected)



Lay-out of the FG generator
(Dik 1992:20, Figure 1)
(details not treated left out)

4. Summary of UniPar

4.1. Theoretical background

Like practically all linguistic theories or formalisms, Functional Grammar has been presented, from its inception onwards, in the 'generative' (or productive) mode. In the ProfGlot system this is reflected in the central role played by the generator. One of the very few moments, if not the only one, when parsing, or analysis, is referred to in the theory is in the context of translation. On that occasion, a few preliminary ideas on a desirable procedure are summarised, and followed by an important final observation (Dik 1979b:4, 7) :

> " The most obvious lacuna for implementing this translation scheme is obviously the analytic procedure which will reconstruct the underlying [structure] for a given surface sentence. ... . For an analytic procedure ... we thus need the following subprocedures (at least) :
>
> a. find verb
> b. reconstruct predicate
> c. find predicate-frame
> d. conflate information from (b) and (c)
> e. find terms
> f. reconstruct term structure
> g. insert terms
>
> This procedure would then have the character of a sort of analysis-by-synthesis : there is a to-and-fro between the surface of the sentence to be analysed, and the different parts of the grammar where, given some item from the surface sentence, we can find information about what sort of structure to expect in the rest of the sentence."

Analysis is discussed by more authors in the context of Functional Grammar (see Connolly & Dik 1989). Kwee (1989) explicitly tries to sketch some first steps on the path indicated in the above ideas, with the help of two well-known parsing tools from computational linguistics, the augmented transition network mechanism based on surface order of constituents, and the chart parser which uses the formalism of rewriting rules.

As regards UniPar, however, it is claimed 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. ... in this way the parser provides immediate access to the rich store of semantic information contained in Functional Grammar clause structures." (Overall structure of ProfGlot, 31)

It is also noteworthy as an attempt at a genuine 'analysis-by-synthesis', in that

> " [it] makes essential use of the generator, in the sense that most rules of the parser take the following form :
>
> 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." (Overall structure of ProfGlot, 31-33)

4.2. The practice of UniPar

Although it is claimed that in the module UniPar of the ProfGlot system

> " [t]wo parsing strategies have been implemented : 'true-parse' reconstructs the fully specified clause underlying the input sentence ; 'deep-parse' directly [*sic*] reconstructs the underlying structure." (Overall structure of ProfGlot, 31)

it will turn out, after a careful comparison of the procedures involved, that the second strategy in essence boils down to the first one plus the final procedure *remove-voice*. It will therefore suffice to look at that first one, which, as may be noted here, should better be called 'full-parse', in order to avoid confusion (one could argue that a 'deep' structure is, in a certain sense, more 'true' than a 'surface' structure where the active or passive voice is an additional feature).

Such remarks aside, let us now look at the details of the procedure *true-parse*. It should first read an input string of surface word forms up to and including the final punctuation, keeping it in what is called an input-list, and then fill a standard preset structure which is referred to as a skeleton representation of a clause structure schema, but will be called here a full-clause skeleton (adding 'full', and leaving out 'schema', for reasons of transparency and convenience).

A full-clause skeleton exactly looks like an underlying clause structure, but it provides only variables and temporarily empty markers at the usual positions for the elements in its predicate frame, for the set of possible argument terms, and for the sets of all possible operators and satellites at the different layers. The fully specified clause is to be reconstructed by gradually processing the input-list of various forms up to, but this time not including, the last element, which after that is used to find the illocution. We thus have (cf. UniPar, 192) :

> applying *parse* to an as yet unknown input sentence is defined as
> - performing *read-in* in order to get an input-list,
> - setting a full-clause skeleton schema, (cf. UniPar, 172)
> - applying *pass-list* to the input-list until exactly one element is left,
> - applying *find-illo* to the only element left.

It is clear that the essence of *parse* is hidden in the kernel procedure *pass-list* which carries the burden not only of distinguishing, or recognising, the right constituents, but also of using them in the right way in the reconstruction. As for the recognition of constituents (an exhaustive enumeration of what counts as such is given shortly), it is here that the claim of 'analysis-by-synthesis' by way of an inverted generator may be justified, although in comparison to what has been stated earlier (see before : Overall structure of ProfGlot, 31-33), it is rather a particular part of the generator, for the parser is now said to be

> " based on an inverted application of the expression rules. It works as follows : you may reconstruct an underlying structure UC for a form F found in the input ... if there is a rule of the expression component which generates F from UC." (UniPar, 173)

The 'parser-as-inverted-generator' claim is further discussed in section 5.

4.3. The essence of UniPar : pass

Pass, in particular find-constituent

As will have become clear, the parsing process in ProfGlot essentially consists of a sequence of passes in each of which one constituent part not only must be recognised but also put into the preset full-clause skeleton structure, such that at the end a fully specified underlying clause structure is arrived at. At each individual pass an appropriate initial part of the dynamically decreasing input list of word forms must be recognised as a constituent of one of six categories, as listed : terms, full terms, predicate complex, satellites, negation, illocution. With regard to these six constituent categories, the following can be observed :

• an illocution can only be recognised after the input list has been reduced to length one (cf. UniPar, 192) ;

• negation is recognised if the lexical negator of the language at hand is found, after which the corresponding slot in the skeleton, namely the one for polarity (a specific predication operator), must be instantiated (*ibid.* 195) ;

• a satellite is found either lexically, as such, or as a full term (*ibid.* 183-184) ;

• a predicate complex either is recognised if a verbal lemma can be identified after which the relevant (finite) tense is retrieved, or must be recontructed if an auxiliary verb form is recognised and a 'rest' can be found, where a 'rest' consists of an infinitive, or a present participle, or a past participle (possibly followed by a past or a present participle) (*ibid.* 184-189, 194) ; it should be remarked that the parser cannot yet recognise term predicates (with additional copular verbs), although such expressions can be produced by the generator ;

• a full term is recognised if a preposition is found which expresses a certain (semantic) function, and if subsequently a term is recognised (*ibid.* 182) ;

• the remaining category, term, requires more details than can be handled here but we will proceed on the assumption that the recognition of such constituents is feasible as well (*ibid.* 177-183) ; note that this category includes embedded structures such as propositional terms, which implies a recursive application of the parsing process, but it should be remarked that these embedded forms can only be recognised in clause-final position, due to the particular definition of the procedure *pass-list* in the ProfGlot system (*ibid.* 182, 192, cf. also 203).

It is clear that a single successful pass will start with any one of the procedures find-term, find-full-term, find-pred, find-sat-n-x (for n from 1 to 4, and for all different specific values of x), or with immediately recognising the lexical negator. One last procedure, find-illo, is only applied to the final element of the input list. This final punctuation is obligatory for a successful parse, for :

> " The parsing task is completed if (a) the parser has 'used up' all the material in the input list, and (b) it has arrived at a well-formed underlying clause structure which contains no more variables." (UniPar, 172)

Pass, in particular match-terms

The distinctive feature of UniPar is not the (morphosyntactic) recognition of constituents that belong to the above mentioned categories, however, since that is something every parser should be capable of, but it is the reconstruction of the underlying clause by way of matching them with positions in the skeleton.

Matching the predicate or a satellite in the pre-set skeleton is relatively simple, so the interesting task of the parser comes down to matching (full) terms with the argument positions that are associated with the (verbal) predicate frame ; but the predicate complex is not always the first constituent to be found, and therefore terms which are recognised before the (main) verb is known, must be kept on hold until that is the case.

In conclusion, the concise characterisation of the parsing process in ProfGlot as given at the beginning of this section can now be elaborated as follows. It is a sequence of passes such that at the end, after the final punctuation has been reached, a fully specified underlying clause structure is arrived at. In each of these passes a constituent which belongs to one of five categories as specified is recognised and put into the preset full-clause skeleton structure. However, if the position for the verbal predicate in the structure has not yet been filled and a (full) term is found, this (full) term cannot yet be placed, so it is temporarily put on hold, in a buffer (UniPar, 193). As soon as the predicate is recognised, all (full) terms in the buffer are put into their proper positions, by a matching procedure. Once the verbal predicate is known, any (full) term recognised is immediately put into its proper position, by the same matching procedure.

The procedure *match-terms* will use all semantic information available in the selection restrictions of the argument positions and the terms. Since this is not a deterministic process, in the sense that there may be more positions at which a recognised (full) term would fit, it is quite possible that decisions as to which position it is put into lead to a dead end in the matching process with respect to a term that is found later on. In that case earlier matchings have to be revoked and the parsing process should be resumed with another match for the earlier term in question. At the end, though, every (full) term found should have been allocated some appropriate argument position (that is, unless it is a satellite).

In this summary, the term 'matching' is rather informally used, in its general sense, applicable to all constituents alike (readers acquainted with Prolog may think of its technical sense in that programming language). In the terminology of ProfGlot proper, the procedure *match-terms* is only applied to (full) terms associated with argument positions, not to a verbal predicate or to a satellite.

In section 5 we will not only discuss the 'parser-as-inverted-generator' claim but also investigate another one, namely that the UniPar parsing strategy may be useful to computational linguistics because it is a novelty in the field (Dik, personal communication ; cf. also the second statement cited in section 1.1).

4.4. Survey of procedures

Pages referred to are from UniPar.

% Reminder — Prolog notation has been simplified and adapted for this survey.
% Commentary annotations are preceded by a special marker, the % sign.

UniPar general set-up

% Note. A single procedure or Prolog-rule can be defined in several Prolog-clauses which
% depend on conditions and thus apply as alternatives (see below, pass-list and find-illo).

% Remark : deep-parse and true-parse, and corresponding 'deep' and 'true' versions of
% such procedures as pass-list, pass, match-terms, and find-proposition
% only differ in specification of subject, object, voice, and agreement (Num, Pers, Gender).

deep-parse = % result is underlying clause structure (not : fully specified) (191)
    read-in, skeleton-schema, % same as in true-parse
    pass-list, % first difference, 'deep' version of pass (disregard verb agreement)
    find-illo, % same as in true-parse
    remove-voice. % second difference, cancel Voice of verbal predicate in nucleus

true-parse = % result is fully specified underlying clause structure (192)
    read-in, % read input string of symbols up till and inclusive of final punctuation
    skeleton-schema, % prepare an 'empty' clause structure (cf. UniPar 172)
    pass-list, % 'true' pass = handle list of input symbols until last single symbol
    find-illo. % handle last remaining single symbol in input list

% stop conditions for pass-list (192) :
pass-list([],[]). % stop when list is empty
pass-list([.],[.]). % stop when list contains single element which is a period
pass-list([?],[?]). % stop when list contains single element which is a question mark

pass-list(Z,Z2) = % piecemeal recognition of constituents in list (193), by
    pass(Z,Z1), % recognising and handling some relevant initial part of list
    pass-list(Z1,Z2). % continuing process with remaining part of list

find-illo([.],decl). % reconstruct illocution as declarative (189)
find-illo([?],interr). % reconstruct illocution as interrogative (189)

find-proposition = % (192)
    skeleton-schema,
    pass-list, % either 'deep' or 'true' version
    remove-voice. % only in case of 'deep' version

% Remark : find-proposition is essential for find-propositional-pterm
% (see later, in particular steps, the procedure find-term) ;
% it differs from parse (see above) at its start (no read-in) and at its end (no find-illo),
% and it is also intended to differ from parse in that it should find any complete proposition
% in some appropriate initial part (which is not known in advance) of the input list —
% however, due to the definition of pass-list and its stop conditions,
% only propositions in clause-final position can be found in this way.

UniPar particular steps

% Note 1. A single procedure or Prolog-rule can be defined in several Prolog-clauses which
% depend on conditions and thus apply as alternatives (see below, pass and find-pred).

% Note 2. A disjunction within a single Prolog-clause is written here as "either ... or ... or ... "

```
pass = find-some-specific-satellite-of-some-specific-order. % (183-184)
pass = find-negation. % if lexical negator element is found, integrate it (195)
pass = % when Voice is empty, that is, before predicate is found (193)
    either find-full-term or find-term,
    add-term-at-end-of-buffer. % keep on hold until predicate is found
pass = % when Voice is empty, that is, before predicate is found (194)
    find-pred, % if predicate is found,
    match-terms. % integrate terms from buffer into argument positions (196-200)
pass =% when Voice is not empty, that is, after predicate is found (195)
    either find-full-term or find-term,
    match-terms. % integrate term into an argument position (196-200)
pass = % when Voice is empty but no continuous verbal complex can be found (194)
    find-aux, not(find-rest), pass, % keep on hold and skip ; find intervening constituent
    add-aux-at-head-of-remaining-part-of-list. % and restore skipped material

find-rest = % (194)
    either find-past-participle
    or find-present-participle
    or find-infinitive.

find-pred = % in case of finite lexical verbs (187)
    identify-lemma, ex-tense, add-features.
find-pred = % in case of continuous verbal complex (187-189)
    find-aux, % the following is written here as a disjunction, but in the book as alternatives
    either find-present-participle
    or find-infinitive
    or find-past-participle
    or find-past-participle, find-past-participle
    or find-past-participle, find-present-participle.

find-full-term = find-prep, find-term. % (182)

find-term = % (180)
    either find-'basic'-bterm % proper-name or pattern '(Det) (Adj) Noun' (181)
    or find-'modified'-mterm
    or find-propositional-pterm.

find-'modified'-mterm =
    find-'basic'-bterm, find-full-term. % NP = NP1 + PP (183)

find-propositional-pterm = % (182)
    find-subordinator, find-proposition. % (192) — see before, in general set-up
```

5. Comments on some aspects of UniPar

5.1. A classic parsing technique : transition networks

In a first attempt to elaborate Dik's (1979b) ideas on parsing along the lines of Functional Grammar, Kwee (1989) applies two well-known classic techniques from computational linguistics literature. An Augmented Transition Network is proposed after a few introductory remarks. That introduction is somewhat extended here, and the technique at hand is presented in three stages, by way of a short tutorial.

Modern theories of grammar have almost exclusively been adopting ways of describing linguistic constructions that, although often explicitly claimed to be neutral as to hearer or speaker, from a computational point of view are much better suited to generation rather than to analysis. The very name of the first such modern theory already has been a sign of this trend. As a result, analysis or parsing, usually defined as delinearisation of input, is seen as re-creation or re-construction of structure. A very extensive computational tradition exists in analysing what is usually called surface structure patterns. Almost all parsing systems have been influenced in some way or another by theories in which the order of constituents plays an important role. No wonder, since those theories (such as Transformational Grammar, Generalised Phrase Structure Grammar, Lexical Functional Grammar, Systemic Functional Grammar) have basically been modelled after natural language phenomena in English. It is, obviously, also more convenient in a computational system to handle an input string of words from front to end (for Western languages : from left to right) than to jump to and fro from main verb to (surface, or even possibly, deep) subject and from there to direct and oblique objects, and adjuncts, and so on.

Augmented Transition Network Grammar is among the oldest of the modern techniques still in use today. It arose in the late sixties and early seventies and gained immense popularity (especially due to Woods (1970, 1973)) in strong connection to the transformational generative paradigm of those days. In the Augmented Transition Network (henceforth ATN) technique, along with the left-to-right processing of surface structure, a tentative deep structure is built up that can be changed gradually as new information is encountered *en route*. The fundamental authority in the field, with extensive guidelines for practice, is Bates (1978, see also Bates 1994). Although the ATN formalism is claimed by experts to be very flexible with respect to the various theoretical points of view, it was onty the merit of Winograd's (1983) unsurpassed classroom text to have severed the ties that linked ATNs with Transformational Grammar, and to have demonstrated to a larger audience that it is indeed quite possible and even relatively simple to use an ATN for other theoretical models as well. The relevant chapter 5 in that book is particularly recommended for a first but thorough introduction to the basic working of a complete system written in this formalism.

The formal definition of structures such as transition networks belongs to the branch of pure mathematics called Graph Theory. In a graph, two classes of entities are distinguished, one could say : points and connecting line segments ; or more technically : vertices and edges ; or : nodes and arcs (it turns out that the tree diagrams that linguists are so fond of form a particular subset of the universe of graphs, and linguists then call these entities : nodes and branches).

Every line segment connects two points (which are not necessarily different) and may, but need not, have a direction. A graph in which all line segments do have a direction (in other words, are directed) is called a directed graph. Such is the case with transition networks, and here we also often speak of states and transitions rather than of points and directed line segments, or nodes and arcs.

States must have names or labels by which they can be distinguished from each other, and the same holds for transitions. As a result, more than one transition is possible between any two states, because a transition will be characterised by its departure state, its arrival state, and its label. States in a transition network, furthermore, belong to one of four kinds : initial, final, both initial and final, or neither initial nor final. This can be described by specifying two subsets of the set S of states, I and F, which are not necessarily disjunct (in other words, the sets I and F may have elements in common).

The formalism of transition networks can be used both for producing and for accepting (that is, recognising as 'grammatically correct', according to given rules) certain strings of symbols (such strings are also often called patterns). In either procedure the goal is to try to walk through the network from some initial state to some final state, while producing or recognising an appropriate symbol (sometimes, a substring of symbols) at each transition, corresponding to the label of the transition taken (in recognising, these symbols should be in the order as given by the input string).

The simplest version of transition networks is called FTN, or finite transition network, also known as a finite-state automaton. This version is equivalent to regular grammars, the lowest class (that is, type 3) in the Chomsky hierarchy of formal grammars. All versions of transition networks, however, and not only the simplest one, have a finite number of states. The finiteness of FTNs concerns the lack of recursion in this version, a phenomenon that is introduced only in the second version, that of a recursive transition network, or RTN, equivalent to context-free phrase structure grammars that constitute the next higher class (that is, type 2) of Chomsky grammars. The extension of RTN to ATN, or augmented transition network, however, is a giant leap which does not correspond at all to the step from Chomsky type 2 to Chomsky type 1, but rather to a direct jump to transformational grammars. As has been said above, for a long time ATN has been associated with this particular syntactic theory, but the formalism is extremely flexible, even to such an extent that it can be adopted for analysis or parsing (as opposed to simple recognition) within any theoretical model whatever.

A more detailed, although very brief, introduction to each of these three types of transition network now follows in the next few subsections. In the diagrams that illustrate them, states are represented by circles, and transitions by lines with an arrowhead, while initial states are pointed at by a single arrowhead in bold, and final states are shaded (elsewhere also often represented by double circles). Transition labels are either a literal (that is, a specific word, or string of words), or else a category, either a lexical one (such as Adj or Noun) or a compound one (phrasal categories such as NP or PP), in which case the word (or word group) involved should belong to the category in question.

Finite transition networks

The simplest kind, the finite transition network or FTN, does not have any compound categories yet. Two examples are shown in the Figures below. The first (Figure 5.1.a) is a very rigid one, since it can produce or recognise only strings that follow a single fixed pattern, namely 'Det Noun Verb Det Noun'. The second one (Figure 5.1.b) offers more variation : some transitions bear a composite label ('Proper-noun or Pronoun'), which is just a shorthand in the diagram, an abbreviation for more transitions, with different labels, between the same states (one 'Proper-noun', the other 'Pronoun') ; the net also allows for an arbitrary number (possibly nil) of Adjectives preceding a Noun, and it has not only two final states but also two initial states ; another phenomenon is, that it is possible to pass from state 3 with a Verb either to state 4 or to state 5. Sample sentences produced or accepted by the first or second FTN are :

Both Fig. 5.1.a and 5.1.b :

A boy kissed a girl.
A boy kissed the girl.
A boy kissed the girls.
The boy kissed a girl.
The boys kissed the girls.
A girl kissed the boys.
The girl kissed a boy.
The girls kissed the boy.

Only Fig. 5.1.b :

Smart tall girls kissed a nice little kid.
A friendly host welcomed the guests.
The strong girls rode high bicycles.
She bought a square paper.
The sailors walked.
Mary loved John.
They knew it.
Dogs barked.

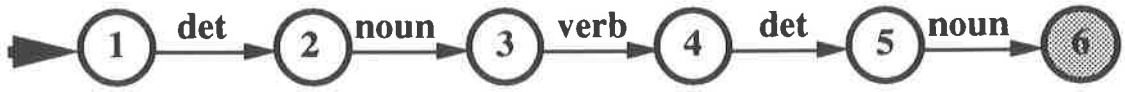


Figure 5.1.a. First example of a finite transition network (FTN)

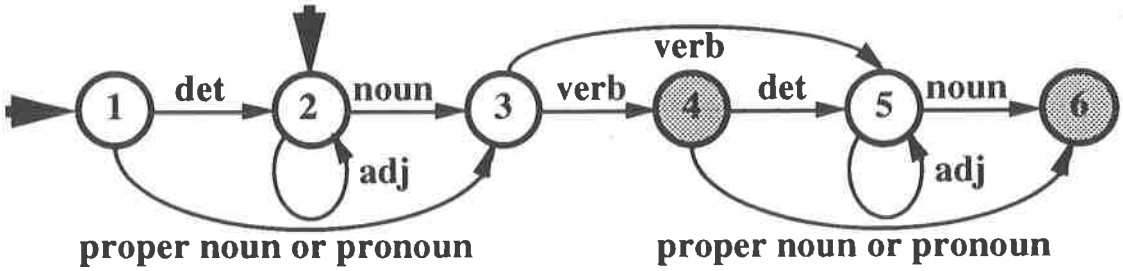


Figure 5.1.b. Second example of a finite transition network (FTN)

Of the features mentioned for the second example, the last two — more initial states, and more transitions from state 3 with the same category Verb — are linked to the notion of nondeterminism or free choice, that is, one is permitted to decide which of the given alternative paths to follow. In general, different paths may lead to different results, however. Recall that one should try to find a walk through the network from some initial state to some final state. If, with a given string, a certain choice has been made and no such successful walk can be performed, this does not necessarily mean that the string is not acceptable, for it might be possible to reach the goal by choosing another alternative. For instance, when starting in Figure 5.1.b at state 1, it is not possible to produce (or to recognise) the first or the last sentence in the second column above, but when starting at state 2 it will be possible, provided a felicitous choice is made at state 3 with the corresponding Verb (in both cases to state 4, not to state 5).

Another type of decision does not depend so much on the given network as on the input words. Take, for example, the fourth sentence in the second column above, "She bought a square paper." After having recognised the first three words as corresponding to the pattern Pronoun Verb Det, we arrive at state 5, but the next word, "square", may be an Adj or a Noun. If the last possibility is chosen, we arrive at state 6, and then we cannot proceed any further although there are still words left, so we should choose Adj and stay at state 5. The next word, "paper", is again lexically ambiguous, but if we decide in the same way as before, staying at state 5, we have no input left any more while not having reached one of the final states yet. This time, therefore, we should have chosen Noun, and arrive at state 6. This is a final state, and at the same time the input is terminated, so the string is recognised as a legitimate one.

There is a strong resemblance between transiting nondeterministic networks and maze traversals. From a static, declarative point of view, a transition will be possible if there exists some path from some initial state to some final state, in one way or another. This does not give us a cue, however, as to how to find such a successful transition, or even to establish if any exists at all. Now, from a dynamic, procedural point of view, two methods are available. Either always follow all alternatives at the same time, in parallel, or always check them one after another systematically, in a 'trial and error' fashion, whenever there is a choice. In technical jargon, the last method is also called 'backtracking'. It has been proved that the class of nondeterministic finite networks does not exceed that of deterministic ones, that is, for each nondeterministic FTN an equivalent deterministic FTN can be construed. Here, of course, equivalence is defined in terms of producing or recognising the same collection of strings. For the FTN of Figure 5.1.b, this set consists of the strings that can be represented as shown in the pattern below. The notational conventions that apply here are the same as those traditionally used in linguistics, curly brackets for obligatory choice of one of the alternatives separated by commas, parentheses for optionality, and a postfix superscript asterisk for an arbitrary number of repetitions.

{(Det) Adj* Noun, Proper-noun, Pronoun} Verb ({(Det) Adj* Noun, Proper-noun, Pronoun})

Recursive transition networks

The next kind is the recursive transition network or RTN. This one does have compound categories, and for each of these it has a separate individual subnet. A complete system is illustrated in Figure 5.2. Some sample sentences are :

The boy kisses the girl in the garden.
The professor gave the book to the girl.
The lady walked to the station.
The sailor kicked the boy in the garden in the city.
The boy in the city kissed the lady.

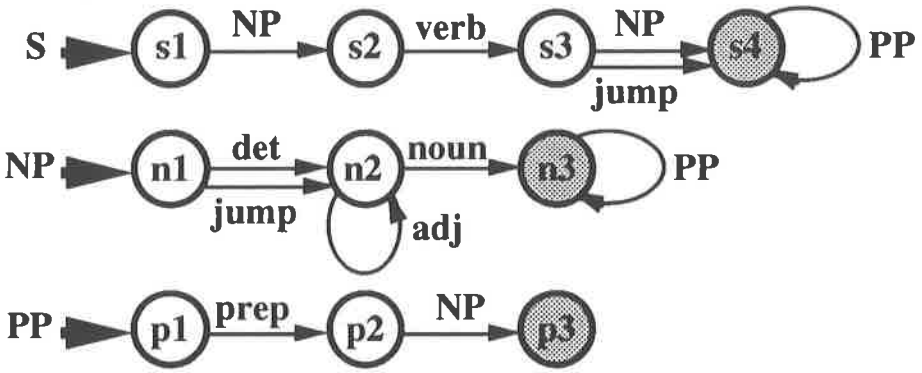


Figure 5.2. An example of a recursive transition network (RTN)

We see the usual syntactic categories NP and PP, but note that there is no VP in this system. A walk through a subnet is the same as a walk through a FTN, except that a transition over an arc with a compound category label succeeds only if the corresponding subnet has been run through. Such crossovers, for instance from state p2 to state n1, and, after the complete NP walk to the final state n3, from that state back to state p3, are called, in technical jargon, 'push (down)' and 'pop (up)', respectively. The words 'up' and 'down' here refer to the direction of embedding, not to the direction in the diagram as displayed.

One feature is new with respect to the previous Figures, but not at all essential for the difference between FTN and RTN, since it can appear also in an FTN. Certain arcs in Figure 5.2 are labelled Jump. This name indicates a free ride, that is, an empty transition which does not produce or recognise any symbol (be it a word or, in an RTN, some constituent). The effect of a Jump arc can always be reached by other means, in the first subnet by a Verb arc from state s2 to state s4, and in the second subnet by defining state n2 as an initial state. Another feature already occurred in the previous Figures, but is conspicuously present here, namely the fact that our walk need not be finished as soon as we have arrived at a final state (see also state 4 in Figure 5.1.b). As a result of the PP arc from s4 to s4 in combination with the PP arc from n3 to n3, however, the RTN of Figure 5.2 is already capable of signalling structural ambiguities with regard to PP-adjunction which can take place either at the level of S or at that of NP. Without the arc (PP, s4—s4), structural ambiguities are restricted to the opposition between [NP Prep NP Prep NP] (which may be considered as a shorthand for [[NP Prep NP] Prep NP]) and [NP Prep [NP Prep NP]].

5.2. Augmented transition networks

As the reader may already have noticed, the previous example networks FTN and RTN also accept, next to the intended grammatical sentences, many other ungrammatical ones. Trying to put constraints on this while staying within the limits of FTN and RTN is a difficult task, involving endless subcategorisation of arc labels. With an augmented transition network or ATN, this task seems to be feasible, and, yet more importantly, in a way that is better motivated by theoretical linguistic concepts (as opposed to strictly formal notions).

This last type of transition networks is obtained by a powerful enrichment of the previous one. Each subnet is provided with a memory, in the form of a notebook that is sometimes called a register structure (Winograd 1983). With each transition arc, furthermore, conditions and actions are associated which pertain to the contents of the appropriate register structure. Entering a subnet (called a 'push') entails the action of initialising a new register, from which the immediately higher one can always be accessed for inspection, and exiting a subnet (called a 'pop') leads to the integration, in some way or other, of the lower register into the upper one. Conditions can also apply to specific details of the arc to be taken, and actions comprise any changes in the contents of the register structure. When a recursive network is thus enriched (or augmented), the possibilities are practically without limits. A register can contain a phrase structure tree with annotations and all, or a system of features such as used in systemic grammars, or any other structure linguists might think of.

As an illustration of an elaborated ATN, Figure 5.3 shows part of the system such as treated by Winograd (1983). Only the subnet for S is displayed here, in three separate instalments which should be superposed one upon another — a single diagram would be far too complicated — the 'backbone' formed by the states 1 to 5 being the same in the three partial diagrams. Figure 5.3 is not the complete subnet for S as treated in the book, for certain other extensions, which handle imperatives and relative clauses, have been omitted here. The subnets for NP and PP are not shown here either. No conditions and actions nor further explanations are given now but the following brief remarks (the extensive illustrative example of a step-by-step analysis process that is given below, however, will mention such details whenever it will be necessary).

Transitions are not only labelled but also numbered, in order to distinguish, for instance, the three arcs from state 5 to state 5 labelled PP which differ in their associated conditions and actions. In such a case, one arc could also have got three differently numbered labels, by way of a shorthand. For embedded clauses, other initial states than state 1 may be needed (see the qualified 'push' at each of the transitions 16, 17, 18). A 'pop' from a final state is indicated by a Send arc, not only for the sake of clarity or uniformity, but in particular to enable the specification of additional terminal conditions and actions. There is yet one more tool (not treated here), a list of global hold registers, especially useful for handling long-distance dependencies (Winograd 1983:232-243).

Recall this characteristic property of ATN, that the contents of a register can be changed (as opposed to just updated) during (or rather after) any transition when the corresponding action so requires. To illustrate this, we sketch the analysis of a passive sentence along the lines of a certain theoretical approach, assuming that the structure aimed at belongs to a kind of deep case grammar. At the same time, the example illustrates the systematic trial and error method that was referred to a short while ago by its technical name of 'backtracking'. Only the states and arcs in the first instalment will be used in this example. Let the sentence be "Mary was given a book by John." (cf. also UniPar 200, (43)). The subsequent actions then develop as follows. After arc 1, "Mary" is seen as Subject. After arc 2, "was" becomes Main Verb. After arc 3 (by convention alternatives are handled in numerical order), "given" becomes Main Verb and "was" Auxiliary Verb. After arc 5, "a book" becomes Direct Object. After the jump over arc 7, the PP "by John" takes arc 9 and becomes a Modifier.

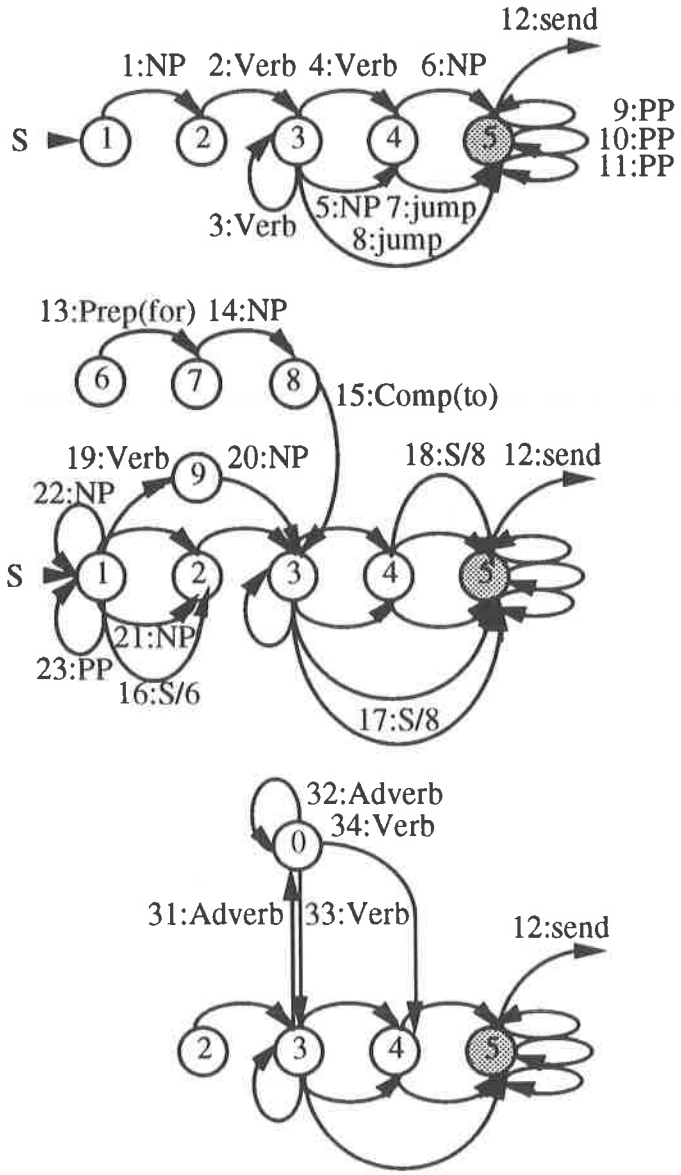


Figure 5.3. An example of an augmented transition network (ATN)
– main subnet in 3 overlapping instalments, not complete – (Winograd 1983)

This is all straightforward, and rather naive. To this point, the pattern wholly corresponds to that of sentences such as "Mary is visiting friends in Italy." The Send arc 12 now puts certain transitivity conditions on the register that are not met in our case. The Main Verb "give" is Bitransitive, not just Transitive, but there is no Indirect Object. Therefore, another alternative should be tried for the choice which has been taken last. PP arc 10 requires the Preposition to be "by" and Subject to be a dummy NP. While the former is indeed the case, the latter is not, so the next alternative, PP arc 11, is tried. As its conditions are not met either, and no more arcs are left, we have to go back as far as the last choice before state 5. It was made in state 3 when the Verb arc 3 was chosen.

Arc 4 offers an alternative, and its conditions are fulfilled, that is, the Form of the word being processed ("given") is Past-Part, and the Type of Main Verb (which, at this moment, is "was") is Be. The corresponding actions prescribe that "was" is to become Auxiliary Verb, and "given" Main Verb, as before, but also that Voice is now set to Passive, and "Mary" becomes Direct Object, while a dummy value must be set for Subject. After arc 6, "the book" replaces "Mary" as Direct Object and "Mary" shifts to Indirect Object. In state 5, the PP arc 9 is tried first, with the same outcome as above, that is, "by John" as a Modifier. This pattern corresponds to that of "Mary was given a book at the desk." The transitivity conditions on Send arc 12 are met (Indirect Object not empty, and Transitivity feature of Main Verb is Bitransitive), so the result is accepted. Most readers would prefer another structure, though, the one found if PP arc 10 were tried first instead of arc 9. Processing the PP "by John" on arc 10 would lead to "John" being Subject, and the transitivity conditions on Send arc 12 would be met as well. This preference can be realised by changing the order of the arcs (with possible consequences for other input sentences).

An attempt to apply ATN to parsing in Functional Grammar

Figure 5.4 is the diagram of Kwee's (1989) proposal to use ATNs for English expressions that conform to the functional pattern as given. It was a tentative idea, advanced in a first attempt at a more specific realisation of Dik's (1979b) general guidelines. A second diagram (which is not reproduced here) was proposed for Term structures, but no conditions and actions were elaborated, pending further development of the theoretical ideas.

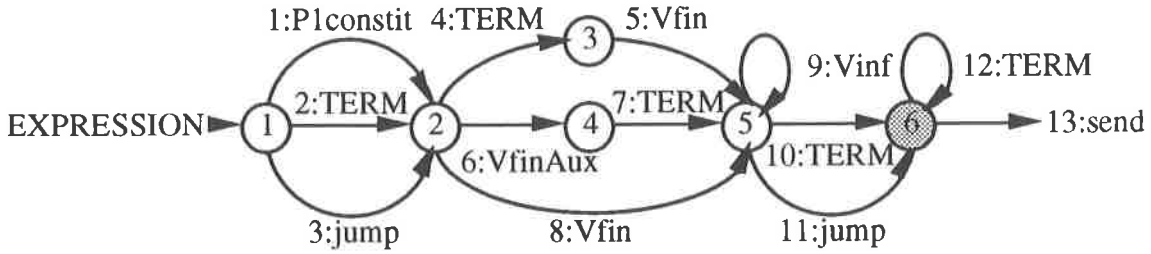


Figure 5.4. Main part of proposed augmented transition network (ATN) for linguistic expressions in an English Functional Grammar with the pattern (P1) {Subj Vfin, VfinAuxQ Subj, VfinImp} Vinf* (Obj) X* (Kwee 1989)

5.3. The UniPar strategy is identical to an ATN

The reader is now completely prepared for the proof that the UniPar strategy is identical to an augmented transition network, and hence not a novelty at all in the field of computational linguistics. It is very straightforward, and after the long and careful introduction given in the previous subsections it can best be represented here in a diagram such as Figure 5.6.

The procedure *parse*, it is explained in section 4, essentially consists of a chain of passes in each of which a single constituent of some category is recognised and put at its proper place in a preset full-clause skeleton structure. Pre-setting this skeleton is shown as the Initialisation action at state 1.

The categories to which a single constituent possibly belongs, are : (full-)term, satellite, predicate, negation, and illocution. The position of the lexical negator in the linear input string of word forms is, for the time being, left free, and in order to simplify the diagram, find-negation arcs are, accordingly, not shown here (it can occur everywhere without further constraints being put on it).

As the illocution must be found at the very end (recall that the procedure *pass-list* as defined in the book does not stop until the input list contains exactly one element), we are left with finding the predicate complex, and (full-)terms and satellites, with the subsequent conditions and actions.

A satellite should immediately be put at its proper place, but the treatment of a (full-)term depends on the predicate's having been found or not. Only when a predicate is known, are its predicate frame and argument positions available for any (full-)term to be put at an appropriate place. Conditions and actions on arc transitions in the diagram have been put at the arrival states of the arcs in question. Those at state 5 serve at the same time as conditions and actions on the final Send arc which, however, is not shown here.

A last note on composite labels is called for. For the sake of convenience, in Figure 5.6 two transitions with their respective arrival states and conditions and actions have been collapsed into one transition plus its arrival state, with corresponding compositions (it happens twice, and it keeps things transparent). Figure 5.5 below explains how this shorthand should be interpreted.

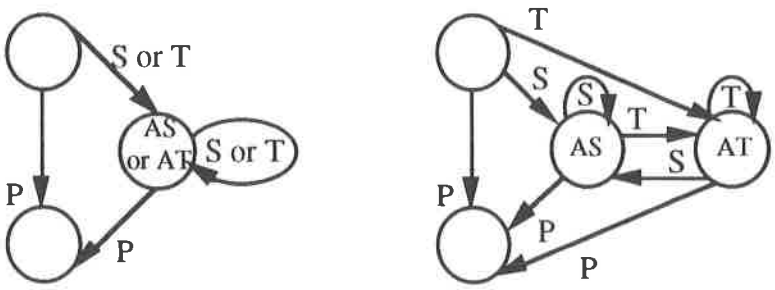


Figure 5.5.

Shorthand diagram (left) and its equivalent (right)

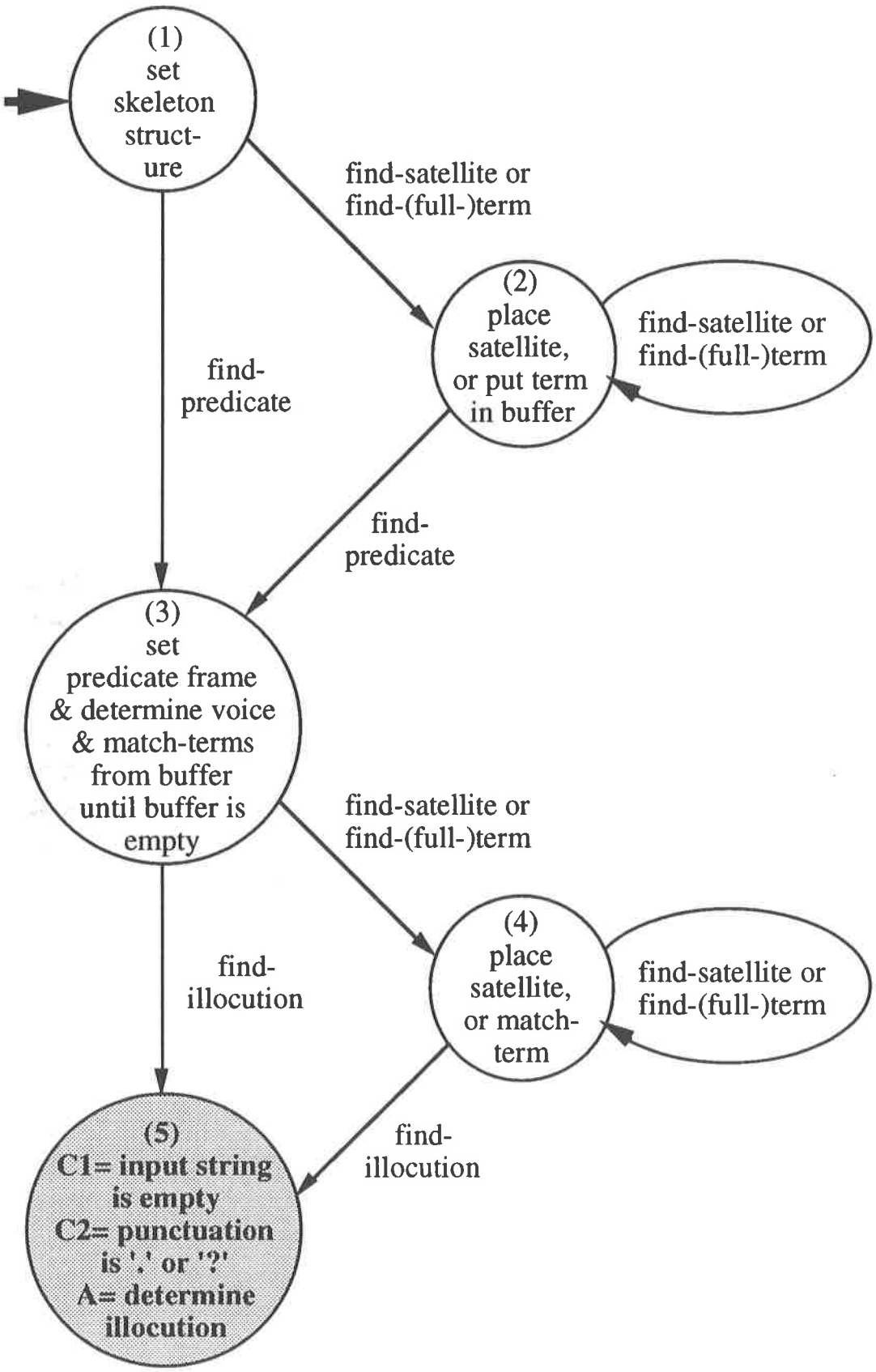


Figure 5.6. UniPar's true-parse
is an augmented transition network (ATN)

5.4. Parser as inverted generator

Even if the parsing strategy of UniPar with its identification of various kinds of constituents and reconstruction of a fully underlying clause is not so novel, another claim is yet to be examined, the one which is put forward in the form of 'the parser as a kind of inverted generator.'

It should be observed that whereas the result of *true-parse* in UniPar is a fully specified clause that could just as well have been produced in UniGen by a call to the procedure *fully-specified-clause* (a statement that is to be qualified in a while), the question would indeed be justified whether the result of *deep-parse* in UniPar really is exactly the same structure such as one produced in UniGen by a call to the procedure *clause-structure* (although it can be used as input to the procedure *specification*). The suggestion that a fully specified clause and a clause structure just differ in so far as Subject and Object are specified or not (UniPar, 170-171) is rather strong but nonetheless incorrect, not only from a theoretical point of view but also in the context of ProfGlot.

The 'inverted-generator' claim is doubtful if for no other reason than that the procedure *deep-parse* is longer than the procedure *true-parse*, and not shorter, as it would have been in case of a genuine 'analysis-by-synthesis'. The former takes a deviation *via* the latter :

> " Voice ([act] or [pass]) is removed because it is irrelevant to the level of 'deep' clause structure ; it is removed at the end because it *is* relevant to the reconstruction process." (UniPar, 192)

It turns out, furthermore, that the actions of *specification* other than subj-obj-assignment (see section 2.3) do not play any part yet in the difference between a 'deep' clause structure and a 'true' fully specified clause, simply because at the present stage of development of UniPar various constituents, in particular personal pronouns as terms and copular verbs as predicates, are still excluded from recognition in the specific alternative elaborations of the find-constituent procedure, notwithstanding the fact that they can be generated in UniGen.

The claim that UniPar is an inverted generator becomes even more doubtful, since

> " ProfGlot can analyse or parse an extension of a subset of the constructions which it can generate," (Introducing ProfGlot, 1)

which in the case at hand not only means that not all generated expressions can (yet) be parsed, as we saw just above (hence the 'subset'), but also that not all parsable strings can (yet) be generated (as implied by the 'extension'), for it is stated that

> " [t]he parser as it stands has some interesting performance features which in some respects make it more powerful than the generator on which it relies ('comprehension exceeds production'), but which in some conditions may also produce incorrect parses (by the side of correct ones)." (UniPar, 201)

Now, this is not quite compatible with the claim currently under discussion as cited in section 4.1, which is somewhat mitigated in the second statement cited in section 4.2, but nonetheless explicitly reformulated as in the third citation below (with a cautious proviso 'in principle'), and again as in the fourth one :

> " The parser makes essential use of the generator, in the sense that most rules of the parser take the following form : 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." (Overall structure of ProfGlot, 31-33)

> " [The parsing strategy] is based on an inverted application of the expression rules. It works as follows : you may reconstruct an underlying structure UC for a form F found in the input sentence if there is a rule of the expression component which generates F from UC." (UniPar, 173)

> " The input to the present parser must in principle be a sentence in L of which the underlying structure can be generated by the L generator." (UniPar, 169)

> " The correctness test for the parse is that it should be possible to regenerate the input sentence ... from the reconstructed underlying clause structure." (UniPar, 172)

Clearly, there are two separate issues here, if we take for granted that existing generator rules guarantee that certain underlying structures produce, or are expressed as, certain forms. In the first place, the 'subset' restriction suggests that in some cases the parser cannot find the relevant rules of the generator, or at least, that it is too difficult to find the appropriate rules through which the given forms can be generated. In the second place, the 'extension' suggests that in some cases an underlying structure is found, in some way or another, which the generator cannot yet express as the given form, because it lacks the proper rules. Neither issue can be made consistent with the claim in question, in any of the formulations cited above, apart from the question whether it is just the form that should be generated from an underlying structure, or the underlying structure itself that should be generated in the first place by the generator.

There is yet another related issue, namely that some incorrect, ungrammatical sentences are in fact successfully parsed (although we may wonder how they could possibly be re-generated). It is even put forward as a positive property :

> " [S]equences ungrammatical in L can nevertheless get a correct parse in L. Unless the parser is used to check the correctness of the input sequence, this is an advantage rather than a disadvantage, since ordinary speech may contain such ungrammatical sequences without understanding being necessarily impaired." (UniPar, 191)

It should be noted that, while with respect to the above claim it is already an important deficiency that UniGen cannot yet produce all sentences parsable by UniPar, it is yet more important, even essential, that no future version of the generator should ever be able to produce any ungrammatical sequences. Such strings, therefore, should not be parsable at all, unless the theoretical claim of a reversible grammar (Strzalkowski 1994) implied by 'the parser as a kind of inverted generator' is abandoned. If it is not so, it is completely unwarranted.

The crux of the question seems to reside in the fact that order of constituents is hardly relevant, or even not relevant at all except at the level of expression, that is, in the module UniExp. In fact, recall the characteristics of the parser :

> " [W]e could formulate a rule to the effect that : you find a clause structure if : you find a term ; you find a predicate complex ; you find a term ; you find a satellite ; you find an illocution ; in this order. ... ... [but] I have implemented a much more flexible parsing strategy at the clause level [which] relies on the following considerations : as far as the parsing capacity of the present parser goes, every parsable sentence can be exhaustively divided into : terms, full terms, predicate complex, satellites, negation, illocution. Let us now define a 'pass' on an arbitrary input string Z as the successful interpretation of the [*sic*] initial part of Z as any of the items listed [above], leaving a rest Z1. ... ... We can thus define the parsing process as a successful series of 'passes' through the input sequence. Each 'pass' reconstructs the structure which can be retrieved from the [*sic*] initial part of that sequence, and defines which action can be taken towards integrating that structure into the underlying schema of the clause. Then, it hands over the rest of the input sequence to the next 'pass' trial." (Unipar, 189-191)

As a consequence of which

> " all permutations of the ... units will receive the same interpretation" (Unipar, 191)

since

> " [t]he parsing strategy is not strictly tied to a fixed order of constituents, except in the following cases : (1) Auxiliaries ... . (2) Determiners, adjectives, nouns, and nominal modifiers must have a fixed order ... . But this order is in fact largely fixed ... . (3) The order of terms which are not overtly marked by a preposition must be fixed, as in : ... . In actual fact, this order is indeed fixed in most cases." (UniPar, 201)

Of these cases, the first two pertain to recognising a single constituent, and the last one pertains to reconstructing the specified clause, in *match-terms*.

As for the reconstruction, the standard format of the full-clause skeleton may remind us of UniGen, but none of the procedures of that module is used here, so at this point the 'parser-as-inverted-generator' claim cannot be maintained in its 'wide' form as cited in section 4.1 and again above.

As for the recognition of constituents, all retrieval rules with names such as *find-x* indeed contain one or more rules with names such as *ex-y* that can be found as specific elaborations in the procedure *formally-ex-clause* of UniExp. On closer investigation, however, it turns out that for retrieval rules for terms and predicates we can only find expression rules at the lowest level, namely : ex-det, ex-restr2 (for adj), ex-noun, ex-present-participle, ex-past-participle, ex-number, ex-fun, and ex-tense, but not ex-(full-)term or ex-verbal-complex, so here the 'parser-as-inverted-generator' claim in its 'narrow' form as cited in section 4.2 and again above can hardly be maintained either, because the constituents that are involved here are only very trivial underlying structures for the given forms.

Some more comments on the parser as it is actually coded and its consequences as regards linguistic aspects will follow in the final section.

6. Comments on some aspects of design and code

6.1. Linguistic aspects

Structural ambiguities related to (full) terms

A notorious problem in linguistic analysis is the case of PP-adjunction and its associated structural ambiguities (see, for instance, Figure 5.2 in section 5.1). A prepositional phrase PP can modify a sentence S but also a noun phrase NP, or in Functional Grammar terminology, a full term (that is, a preposition plus a term) expresses a satellite or an attributive adpositional term (for the sake of simplicity we will disregard here the case of adpositional predicates such as in 'the books are for mary'). Examples of attributive adpositional term modifiers are explicitly mentioned throughout the book, at several occasions :

> " ProfGlot produces ... attributive adpositional term modifiers : *john's book, the book of john, the man in the garden, ... the man with the book.* Note that what we have here is a term embedded within another term." (Overall structure of ProfGlot, 35-36)
>
> " The third restrictor position can be taken by attributive adpositional terms ... Such a term may be formed by placing a concrete term into a possessor frame with semantic function 'poss' ... This results in restrictors such as *this man's, of this man.* ... Idem, but now for locative restrictors such as *in the garden, in the city.* ... Idem, but now for 'associative' restrictors such as *with the book, with the book of the boy in the garden,* etc." (UniGen, 90-91)
>
> " the potential restrictor 3 is realized as a postmodifier ... *the professor of the girl, the book of the lady, the man in the garden, the lady with the roses.*" (UniExp, 116)
>
> " As for the power of the present parser : it can handle ... ... all such adpositional modifiers as can be generated by the generator, and therefore complex terms such as *the boy in the city, the ladies with the books of the professor.*" (UniPar, 176)
>
> " An 'mterm' (modified term) is found in a situation such as : *the boy in the garden. the boy* is interpreted as a 'bterm', *in the garden* ... as a 'full term' ... entered into the R3 position of the bterm *[sic].* Since the rule for 'mterm' is formulated recursively, it will also correctly identify and reconstruct the following strings as possible 'mterms' : *the boy of the professor in the city , the book of the lady with the roses for the painter.* Note that such a sentence can also be interpreted as a term followed by a satellite, so that we get two analyses for a sentence such as : *John kicked the boy in the garden.* and multiple analyses for : *John kicked the boy in the garden in the city.*" (UniPar, 183)

The structural ambiguity between sentence modifier and noun phrase modifier is duly indicated and accordingly analysed in the system. Because constituent order is relatively unimportant, undesirable parses admittedly are got as well :

> " In certain cases the parse gives incorrect parses. Consider the following example : ...
> a. [the boy in the city] [kissed] [the lady] (correct)
> b. [the boy] [in the city] [kissed] [the lady] (incorrect)" (UniPar, 202)

There is, however, another and slightly more subtle ambiguity which seems to have been completely ignored, although in fact it should have been noticed in the context of such remarks as 'term embedded within another term' and 'the rule for 'mterm' is formulated recursively'. What is not correctly identified as a possible legal term structure is the second item in the following pair of noun phrases, as opposed to the first one, nor can the second structure be produced :

a. books with roses by the famous painter = books with [roses by the famous painter]
b. books with roses by the famous printer = [books with roses] by the famous printer

It may be noted in this connection that Kwee (1988a, 1994) points out that the theory lacks the tools to distinguish a similar structural ambiguity in adverbial subordinate clauses, [S1 + AdvS=[S2 + AdvS3]] *versus* [[S1 + AdvS2] + AdvS3].

Expressing embedded propositions or predications

Some term positions are not filled by a term structure but by a predication or a proposition. The expression of such terms usually starts with a subordinating conjunction that depends on the semantic function of the position in question, for instance, 'that' for English sentential complements. However, in English

> " we can also express an embedded proposition by nominalizing it ... . We nominalize a proposition in English by marking the subject term with the marker 'gen' ('genitive') and then doing 'nominalized clause expression' to it. ... This will produce :
> *john deplored mary's cheating.* [and] *mary's cheating was deplored by john.*
> [Also,] [i]n English the complement of a verb like *want* must be expressed in the form of an 'accusative cum infinitivo' ... ; a finite subordinate clause ... gives bad results :
> *John wanted Mary to cheat.* [but not] **John wanted that Mary cheated.*
> This pattern is achieved by marking the subject as 'ssubj' (which will yield the object form ...) and doing infinitival expression to the embedded predication. Note that I have provisionally solved this problem in the expression component, without doing any 'deeper' Raising operation. This is a matter for further consideration." (UniPar, 120)

The warning issued in the last lines is very appropriate and justified. Even so, it would have been better if the two procedures were left out altogether. They confuse the interested but un-initiated readers who do not know Dik (1979a, 1980a). The note is not a hidden caution, but it is not given emphasis either. It is easily overlooked amidst the surrounding Prolog code which requires a lot of going back and forth between definitions on different pages involved in the expression of these constructions. My chief objection to *infinitival-expression* and *nominalisation, qua* processes, apart from the fact that they are defined in a rather discouragingly intricate way (see subsection 6.2), and that the passive counterpart of the *Acc-cum-Inf* is not covered (that is, the *Nom-cum-Inf* such as in 'Mary is required by John to cheat'), is that they are not theory-driven at all, only result-driven, with an un-principled, strong transformational flavour.

Recognising embedded propositions or predications

It would be very surprising if subordinate clauses in nominalised or infinitival form were parsed in UniPar. They can't be. Nor can relative clauses. Other subordinates, regardless whether they are satellites (adverbials) or argument terms (complements), can be parsed only if in sentence-final position, which is not noticed as such in the book, let alone explained. Only an example is given :

> " Certain correct parses are not delivered. Consider the following example :
> [john] [deplored] [that mary cheated yesterday] (correctly parsed)
> [john] [deplored] [that mary cheated] [yesterday] (he deplored it yesterday - not parsed)
> Obviously, these ... require further development of the parser." (UniPar, 202-203)

6.2. Computational aspects

Elsewhere I have expressed my view on the proper use of Prolog predicates (or functors), and of data-structuring facilities in Prolog in general, and for the representation of theoretical linguistic notions from Functional Grammar in particular (Kwee 1994, section 7.1.2). In addition, a very brief tutorial on programming, and especially on programming in Prolog, can be found there (*ibid.* appendix 0.1). Those sources may be useful for linguists as a supplement and a counterpart to chapter 2 of Dik's book, *Some elements of Prolog*. Here, as in the previous subsection, a handful of disparate topics will be handled.

Expressing embedded propositions or predications

A survey of the procedures relevant to expressing sentential complements in English is provided for the readers' convenience. Pages are from UniExp.

ex-clause = fully-specified-clause, ex2-clause. 143
ex2-clause = formally-ex-clause, full-place. 143
formally-ex-clause = ex-verbal-complex, ex-full-terms, ex-sats(-1-2-3-4). 130
ex-verbal-complex = ex-voice, ex-progr, ex-asp, ex-att, ex-pol, ex-illo, ex-tense, flatten. 124

ex-full-term = ex-term, ex-fun. 119, 121

ex-full-term(prop) = ex-emb-prop, ex-fun, sub1(that). 119
ex-emb-prop = formally-ex-emb-prop, emb-place. 143
formally-ex-emb-prop = ex-verbal-complex, ex-full-terms, ex-sats(-1-2-3-4). 131

ex-full-term(prop,fact) = either (ex-emb-prop, ex-fun, fact-m) or (nominalisation, ex-fun). 119
nominalisation = substit(subj,subj-gen-s), ex-nom-clause. 120
ex-nom-clause = formally-ex-nom-clause, emb-place. 144
formally-ex-nom-clause = ex-nom-verbal-complex, ex-full-terms, ex-sats(-1-2-3-4). 131
ex-nom-verbal-complex = ex-voice, ex-asp, ex-nom =present-participle 130, ex-nom-pol. 125

ex-full-term(ext-pred) = substit(subj,ssubj), infinitival-expression, ex-fun. 120
infinitival-expression = formally-ex-inf, emb-place. 144
formally-ex-inf = ex-inf-complex, ex-full-terms, ex-sats(-1-2). 131
ex-inf-complex = ex-voice, ex-asp, ex-inf =to-plus-inf-string 130, ex-nom-pol. 125

Recognising embedded propositions or predications

Picking up where the previous subsection 6.1 ended : the book fails to connect the example of the missed parse with the definition that is its only explanation, ten pages earlier (see also the survey of procedures at the end of section 4) :

> " Finding a proposition analogously works through a part of the input sequence to see whether it can reconstruct the propositional part of an underlying clause structure (deep) or of an underlying 'fully specified clause' (true). These procedures are used to interpret propositions ... such as *that mary cheated* in a sequence such as [john, deplored, that, mary, cheated, .] *[note the period]* Passing through a list is defined recursively in the following clauses. Stop conditions are [] (no more material in the list), or [.] or [?] (end of the list reached). ... . Otherwise, we can pass through a list in mode DT (deep/true) by doing a pass on the *[sic]* initial subsequence of the list, and then list-passing the rest of the list (until one of the stop conditions is reached)." (UniPar, 192-193)

Basic term, bterm, find-bterm, and related details

Choice of terminology is not always felicitous or transparent, and sometimes it is a possible source of annoying confusion, especially for readers who are not fluent in the Functional Grammar jargon. To mention a single example, in the theory a basic term (proper name or pronoun) is given as such in the lexicon, but in the system the functor *bterm* applies only to abstract representations of personal pronouns. Basic question terms have a functor *bqterm*, relative terms have no functor, proper names, or proper nouns, are treated as basic predicate frames instead of basic terms (as an alternative to what is stated in the theory). What is meant by the 'bterm' in the parsing procedure *find-bterm*, however, is a 'basic', or rather an elementary, term structure that conforms to the pattern (Det) (Adj) Noun. Note that the more general, but equally elementary, pattern (Det) Adj* Noun (section 5.2) cannot be generated nor analysed in the system.

In this connection, it should further be remarked that in the system a predicate frame is a triple consisting of a linguistic form, a type specification, and a list of descriptions of argument positions, whereas its category can be found in the particular Prolog functor, varying from *bpredv*, *bpredvm*, *bpredn*, to *bpreda*, for basic (lexical) frames that are verbal, matrixverbal, nominal, or adjectival.

I disapprove of Prolog functors for the representation of a feature or property such as the lexical category of a predicate, but would recommend their use, on the other hand, to distinguish data types. The distinction between entities (and hence terms) of different orders, is a good example. This distinction is made in the formulae by means of the variables x, e, X, E, with a subscripted index. Because of case sensitiveness, X and E are encoded as xx and ee, but they can be omitted altogether, in fact, because different types can be distinguished by different functors, for instance, *term-str*, *extpred-str*, *prop-str*, and *clause-str*.

The indexed variable x_i of a Functional Grammar term structure is encoded as an additional element, an operator called the referent identifier, or referential index, at least for first order entities (UniGen, 87, 90). For higher orders, it is the propositional (*ibid.* 95) or illocutionary (*ibid.* 98) referent. This operator has the form of an ordered pair consisting of the type-distinguishing atom (x, e, xx, ee), and the index proper, being an arbitrary numerical value. I already explained in section 3.2 why the use of a random number generator (*ibid.* 87, 89-90) is completely out of order here, as in this way two terms are liable to receive the same numerical index, although they are not intended to do so. It may be suspected that it happens quite often (in a row of two random numbers less than or equal to 25 the probability of two or more equals is 4%, in a row of three 11%, of four 22%, of five 34%, of six 47%), but it will never show, because numerical indices are not visible at the level of linguistic expressions. The only part term indices seem to play is related to co-indexing in *anaphora-resolution*, when a 'free' anaphorical term drops its provisional index *same*, and inherits the [x,N] from the subject term. An 'unresolved' anaphorical term is expressed as *nn* (UniExp, 117), which presumably stands for *nomen nescio*.

Blind alleys remaining invisible, backtracking after failure

I just mentioned a strong suspicion of mine that unintended undesirable results of a certain kind would never show up although they might often be produced. A similar suspicion arises in relation to another type of undesired results, such as can occur, for example, in relative clause formation. Kwee (1981) describes how relativisation can be achieved in a proper and effective way. Formation of an arbitrary one-open predication is not effective as the basis for a relative clause, nor is the unrestricted choice of a predicate frame a good start, for it is not always possible for the given head noun to meet the selection restrictions associated with the available position(s). Repeated choices after initial failures, until a suitable frame is found, will not guarantee effectiveness either. In the ProfGlot system, we are told, when a term structure is being built,

> " [the fourth restrictor] R4 may be empty ([]) or, if gambling on 'rel' succeeds, it may be a 'relative' in relation to the head noun R1 and the number specification of the term (these relations are established in order to guarantee selection of 'appropriate' relative clause structures)." (UniGen, 89)

but the procedure *relative*, with the given head R1 and the specification Num, starts with an unqualified call to the procedure *ext-pred-schema*, without any relevant parameter (precisely here reference is made to the above paper, in a footnote).

> " Restrictors-4, underlying relative clauses, can be formed as follows ... . The relative structure is formed by taking an extended predication schema, relativizing one term position appropriate to the head noun R1, and inserting terms into all the remaining term positions. Note the parallelism with Q-word questioning ... ." (*ibid.* 91)

The parallelism with Q-word questioning, however, is only very partial, since the latter operation is not constrained by a specific head noun as an antecedent. Readers should carefully compare the definitions of *relative* and *perform(rel)* (*ibid.* 91, 93), with those of *qext-pred* and *perform(que)* (*ibid.* 85, 92).

The two parameters R1 and Num do play a part in the steps *do-one-term(rel)*, *do-one(rel)*, and *perform(rel)*.

> " Relativization of some term position ... within a predication must be formulated in relation to the R1 (the head noun) of the term ... , if we want to avoid such output as : *the ball which laughs* (the first argument of *laughs* requires an animate term ; the head noun is inanimate)." (*ibid.* 93)

We may wonder what happens if we have a noun 'ball', and the random choice gives a frame 'laugh'. Relativisation fails on all term positions (arguments and satellites), and the backtracking mechanism makes the system look for another path, undoing some results, possibly the decision to have relativisation, maybe even 'ball' as the head of a term-to-be-formed. Anyway, the unintended blind alley (unintended, as it was in fact decided to have a relative clause) remains forever invisible and undetected under the usual demonstration circumstances (unless trace facilities are switched on for inspection purposes).

The issue of backtracking brings me to the last couple of remarks on the code. Don't rely on Gricean maxims when you read that the top rule 'go' continues

" repeating ... until no more sentences can be formed." (UniExp 144, cf. section 2)

That is, don't expect the 'go'-cycle to eventually stop, it will go on till the cows come home. If it is meant to stop, *repeat-fail* as coded is a mistake (consult any good Prolog book). This technicality aside, there is another misapprehension : *arbitrary-pred* never exhausts the fund, except when the lexicon is empty and no sentence is formed at all. In the marginal case of there being no more than just one verb, the system will always choose that predicate. It will produce an infinite row of sentences, not necessarily all different from each other, such as

a walking walker walked. walkers had been walking. the walker that is walking walks.

Processes such as rules related to arbitrary choices (*choose-random*, UniGen 71), to decisions to realise certain linguistic complexities (*gamble*, Chapter 5 : 'Basic facilities'), or to term indices (the class *rf*, *prop-referent*, *illo-referent*, as discussed before), all involve a special built-in LPA-Prolog procedure that randomly generates a positive integer below a specified ceiling, and is called *irand(M)*, apparently an abbreviation for 'integer-random'. Their definitions show trivial but annoying mistakes and inconsistencies related to the result of *irand(M)*. Thus, the last element of a list is never chosen, gamble(p) will never succeed for p(1) but may fail for p(100), an explicit definition for irand(0) is lacking, and I also have my doubts about the range, as it should be up to and including the maximum (that is, not 'under' but 'less than or equal to').

Backtracking is related to the *gamble* procedure in the context of alternatives and disjunctions. Recall that case distinctions, called *if-then-else*-constructions, can be coded in Prolog in two ways. Alternatives are different elaborations of a Prolog-rule in different Prolog-clauses, usually depending on conditions that may be specified in the parameters. A disjunction is a case distinction within one Prolog-clause, correspondingly formulated as *either-or-or-* (examples can be found in the survey of procedures in section 4.4). At the present stage of Prolog implementations on deterministic machines, the order of the possible cases is relevant, for they are inspected one after another in the order as given in the code, until one case is successfully executed and completely terminated. It is not clear why case distinctions with *gamble* should be ordered as shown in the examples below. I would have expected the reverse order. What happens if *gamble* fails ? When is *gamble* reached anyway ? I think I must have seen some results during a demo session, but I don't quite see in the book how they are produced. During backtracking after some failure further down the preceding branch ? After an explicit 'fail', or unintentionally ? It is all very mysterious.

```
clause-structure = ext-pred, ..., ..., ... .
clause-structure = gamble(que), ext-pred, ..., ..., ... . (UniGen, 98)

term = ..., ..., ..., (either R2=[] or restr2),
    (either R3=[] or (gamble(modf), restr3)),
    (either R4=[] or (gamble(rel), relative)). (UniGen, 88)
```

6.3. Epilogue (summary and conclusion)

As is observed at the beginning of this paper, it is not so easy to distinguish, in Dik's presentation of his system, the linguistic theory from the computational implementation, as a result of which it is hard to exactly recognise on which points either of the two sides might have influenced the other side. Where a linguistic theory and a computer program are so intimately entangled, it will also be very difficult to discern essential aspects from accidental ones.

In the main part of this paper, the critical assessment concentrated around two particular points in UniGen and in UniPar. There are various reasons for this choice. First of all, the two subsystems clearly are fundamental for the system as a whole. In the second place, from a theoretical point of view Functional Logic and Functional Translation are so rudimentary that they cannot compete with what has been achieved elsewhere in the domains of logic and translation, two fields that, moreover, extend so far beyond the conventional boundaries of linguistics that, for an adequate discussion of either issue, author as well as readers need to have more than just plain elementary knowledge of the subject matter. Thirdly, generation and analysis are the branches which, presumably, the intended (linguistic or computational) readership of this paper will be most knowledgeable about, and, also, I am most acquainted with.

In the discussion, familiarity with certain topics has led to frequent references to and comparisons with work I have done in modelling Functional Grammar on computer. In that research I carefully distinguish the linguistic theory as such from the actual programming efforts. Inconsistencies, vagueness, weak spots, and other questionable points in the theory are brought to light when the output is unintended and undesirable, that is, not descriptively adequate, but only on this condition that the grammar is modelled as truly and as faithfully as possible. Solutions are never proposed with a view to theoretical linguistic claims, but always presented as a computationally motivated approach which might suggest a way for the linguist to look at. In that sense, it is addressed to a specialised audience, in an attempt to establish a dialogue between functional linguists involved in developing their grammatical theory, and programmers who, with a sympathetic but critical eye, are trying to model it on computer.

Dik's book is aimed at a different readership. His theory is to be presented to relative outsiders, and the exercise of writing linguistic computer programs is to be demonstrated to yet another class of relative outsiders. Both goals suffer from the mix between them. Readers interested in the grammatical theory risk getting a slanted impression of Functional Grammar and should, therefore, not judge it by this book alone. The same holds for the Prolog programming side. Elsewhere I expressed my view on Prolog style, and I shall not repeat it here. A large part of the code, moreover, is spent, unfortunately and regrettably, on simple morphological details, nothing but elementary programming exercises in string manipulation, not interesting from a general theoretical point of view and certainly not typically functional from a particular linguistic point of view.

References

Bates, Madeleine 1978. The theory and practice of augmented transition network grammars. In : L.Bolc ed., Natural language communication with computers. Berlin : Springer. 191-259.

— 1994. Transition Network Models. In : R.E.Asher & J.M.Y.Simpson eds, the Encyclopedia of Language and Linguistics (10 vols). Oxford : Pergamon Press. 4666-4670.

Connolly, John H. 1989. Functional Grammar and Artificial Intelligence. In : J.H.Connolly & S.C.Dik eds, Functional Grammar and the computer. Dordrecht : Foris. 217-228.

Connolly, John H. ; Dik, Simon C. eds. 1989. Functional Grammar and the computer. Functional Grammar Series 10. Dordrecht : Foris.

Dik, Simon C. 1979a. Raising in a functional grammar. Lingua 47:2/3. 119-140.

— 1979b. How to arrive at a procedure for mechanical translation (MT) in terms of Functional Grammar (FG). Amsterdam : Institute for General Linguistics, University of Amsterdam.

— 1980a. Seventeen sentences - basic principles and applications of Functional Grammar. In : E.A.Moravcsik & J.R.Wirth eds, Current approaches to syntax. Syntax and Semantics 13. New York : Academic Press. 45-75.

— 1980b. Studies in Functional Grammar New York : Academic Press.

— 1983. On the status of verbal reflexives. Communication and Cognition 16:1/2. 39-63.

— 1989. The theory of Functional Grammar. Part I : The structure of the clause. Dordrecht : Foris.

— 1992. Functional Grammar in Prolog - an integrated implementation for English, French, and Dutch. Natural Language Processing 2. Berlin : Mouton de Gruyter.

Hengeveld, Kees 1989. Layers and operators. Journal of Linguistics 25:1. 127-157.

— 1992. Non-verbal predication - Theory, typology, diachrony. Functional Grammar Series 15. Berlin : Mouton de Gruyter.

Kobsa, Alfred 1987. What is explained by AI models ? In : R.Born ed., Artificial Intelligence - the case against. London : Croom Helm. 174-189.

Kwee Tjoe Liong 1981. In search of an appropriate relative clause. In : T.Hoekstra ; H.van der Hulst ; M.Moortgat eds, Perspectives on Functional Grammar. Dordrecht : Foris. 175-189.

— 1987. A computer model of Functional Grammar. In : G.Kempen ed., Natural language generation - new results in artificial intelligence, psychology and linguistics. Dordrecht : Martinus Nijhoff. 315-331.

— 1988a. Natural language generation - one individual implementer's experience. In : M.Zock & G.Sabah eds, Advances in natural language generation - an interdisciplinary perspective (2 vols). London : Pinter Publishers. volume 2, 98-120.

— 1988b. Computational Theoretical Linguistics - generating (English) sentences in (Dik's) Functional Grammar. In : [D.Bojadžiev ; P.Tancig ; D.Vitas eds,] Proceedings of the IV-th [federal Yugoslav] Conference [on] Computer Processing of Language Data. Ljubljana : Institut Jožef Stefan [and] Society of Applied Linguistics of Slovenija. 75-90.

— 1989. An ATN parser for English FG ? Or maybe an active chart ? In : J.H.Connolly & S.C.Dik eds, Functional Grammar and the computer. Dordrecht : Foris. 93-107.

— 1994. Programmer's reflections on Functional Grammar - more exercises in computational theoretical linguistics. Studies in Language and Language Use 8. Amsterdam : IFOTT.

— to appear. On generation in Functional Grammar - towards an alternative, more discourse oriented approach. In : J.H.Connolly & R.Gatward eds, A Fund of Ideas (provisional title) (papers from the 6th International Conference on Functional Grammar, York 1994).

Ritchie, Graeme 1980. Computational Grammar - an artificial intelligence approach to linguistic description. Brighton : Harvester ; Totowa NJ : Barnes and Noble.

Strzalkowski, Tomek ed. 1994. Reversible grammar in natural language processing. Dordrecht : Kluwer.

Winograd, Terry 1983. Language as a cognitive process. Volume I : Syntax. Reading : Addison-Wesley.

Winston, Patrick H. 1977. Artificial Intelligence. Reading : Addison-Wesley.

Woods, William A. 1970. Transition network grammars for natural language analysis. Comm. ACM 13:10. 591-606. Also in : B.J.Grosz ; K.Sparck Jones ; B.L.Webber eds, Readings in Natural Language Processing. Los Altos : Morgan Kaufmann, 1986. 71-87.

— 1973. An experimental parsing system for transition network grammars. In : R.Rustin ed., Natural Language Processing. New York : Algorithmics Press. 111-154.