COGNITIVE SCIENCE 11, 201-258 (1987)

An Incremental Procedural Grammar for Sentence Formulation

GERARD KEMPEN AND EDWARD HOENKAMP

Department of Psychology
University of Nijmegen, The Netherlands

This paper presents a theory of the syntactic aspects of human sentence production. An important characteristic of unprepared speech is that overt pronunciation of a sentence can be initiated before the speaker has completely worked out the meaning content he or she is going to express in that sentence. Apparently, the speaker is able to build up a syntactically coherent utterance out of a series of syntactic fragments each rendering a new part of the meaning content. This incremental, left-to-right mode of sentence production is the central capability of the proposed Incremental Procedural Grammar (IPG). Certain other properties of spontaneous speech, as derivable from speech errors, hesitations, self-repairs, and language pathology, are accounted for as well.

The psychological plausibility thus gained by the grammar appears compatible with a satisfactory level of linguistic plausibility in that sentences receive structural descriptions which are in line with current theories of grammar. More importantly, an explanation for the existence of configurational conditions on transformations and other linguistics rules is proposed.

The basic design feature of IPG which gives rise to these psychologically and linguistically desirable properties, is the "Procedures + Stack" concept. Sentences are built not by a central constructing agency which overlooks the whole process but by a team of syntactic procedures (modules) which work—in parallel—on small parts of the sentence, have only a limited overview, and whose sole communication channel is a stack.

IPG covers object complement constructions, interrogatives, and word order in main and subordinate clauses. It handles unbounded dependencies, cross-serial dependencies and coordination phenomena such as gapping and conjunction reduction. It is also capable of generating self-repairs and elliptical answers to questions. IPG has been implemented as an incremental Dutch sentence generator written in LISP.

The cognitive processes underlying sentence production are usually categorized under the headings of content, form, and sound. One group of activ-

The research presented in this paper has been supported by grants from the Netherlands Organization for Pure Scientific Research (Z.W.O.). We are indebted to William Levelt, Herman Kolk, Carel van Wijk, Koenraad De Smedt, and Patrick Hudson for their comments on earlier versions of this paper, and to Peter Desain and Henk Schotel for their collaboration in setting up a computer implementation of Incremental Procedural Grammar. Most of all we thank Mary-Louise Kean for scrutinizing the prefinal manuscript.

Correspondence and requests for reprints should be addressed to Gerard Kempen at the Department of Psychology, University of Nijmegen, Montessorilaan 3, 6525 HR NIJMEGEN.

ities is concerned with planning the conceptual (semantic) content for language utterances. They select to-be-verbalized conceptual structures in such a way as to be "digestible" for the listener, that is, comprehensible, interesting, not too redundant, and so forth. A conceptual structure is linearized by splitting it up into a sequence of messages each of which is expressible in a complete or partial sentence. These and related activities may be termed *conceptualizing*. A second group of processes takes care of translating meaning content into sentence form. This we call *formulating*. Finally, syntactic and morphological structures built by the formulator system are handed over to the mechanisms of speech for overt *articulation* (Fromkin, 1971; Kempen, 1977; Levelt, 1982).

This paper is concerned with sentence formulation. It proposes a sentence construction device, termed "Incremental Procedural Grammar" (IPG), which aims at both psychological and linguistic plausibility. By psychological plausibility we mean that all sorts of psychological data on how speakers assemble natural language utterances during spontaneous speech are taken into account, so that the device may be said to simulate human sentence production processes as closely as possible. The goal of linguistic plausibility implies that we try to incorporate into the device grammatical (syntactic, lexical, morphological) rules which a linguist would not qualify as ad hoc, that is, which cover a range of grammatical phenomena as broad as is possible by current standards of linguistic research. In particular, the device should incorporate an optimal solution to what has become one of the central issues in the theory of syntax: conditions or constraints on the applicatioan of rules (transformational and others).

What properties are desirable or necessary for a sentence construction device to qualify as psychologically plausible? Sections 1 and 2 provide an answer to this question which, in contrast to extensive discussions on linguistic adequacy of grammars, has not received much attention in the literature. The core of the paper is a detailed description of the formulator system we have worked out. After an overview of the workings of IPG in Section 3, we present analyses of some important syntactic constructions of the Dutch language (Section 4). Finally, in Sections 5 through 7 we will return to the issue of psychological plausibility of the proposed grammar.

1. PSYCHOLOGICAL CONSTRAINTS

A most remarkable property of the human sentence production system is the high level of output fluency it is able to attain. The primary factor conducive to fluency derives from the temporal alignment of the three subprocesses of speaking: conceptualizing, formulating, and articulating. The traditional view, implicitly held by many students of sentence production, is that they are ordered strictly serially in time. First, the conceptual content is fully specified by the conceptualization process. Next, the syntactic structure is built for the whole utterance. Finally, this structure is realized phonetically (cf. Figure 1a). This serial model implies that hesitations *within* sentences cannot have a conceptual or syntactic origin. This is not only empirically wrong (cf. Goldman-Eisler, 1968) but it is also contradicted by the following introspective observation.

Speakers often experience situations where they initiate overt speech production after having worked out only a fragment of the conceptual content of the resulting utterance. They also find it very easy to take up the thread of a broken-off sentence spoken by someone else and bring it to a syntactically impeccable end. Such phenomena force us to give up the strictly serial view in favor of the position that the three subprocesses run parallel to each other (Kempen, 1977). As soon as a fragment of conceptual content has been computed it is passed over to the formulator which attempts to translate it into a sentence fragment that is then articulated (Figure 1b). In the meantime, work on further conceptual and syntactic fragments continues. Figure 1b also shows that the order of conceptual fragments does not always correspond to the order of utterance fragments. This is caused by rules of syntax which may call for a reversal. As a matter of fact, we make the assumption that the "conceptualizer" system has no syntactic knowledge whatsoever. The order in which it delivers its conceptual fragments will therefore, in principle, be uncorrelated with the order of the corresponding utterance fragments in the spoken sentence. In reality, however, the correlation will be positive because the formulator will try to match them.

The mode of sentence production intended here we will term *incremental* or *piecemeal*. Its usefulness undoubtedly relates to the more efficient management of the processing capacities of working memory and other mental machinery involved in formulating and articulating. It prevents these mechanisms from having to operate in a very irregular fashion, that is, according

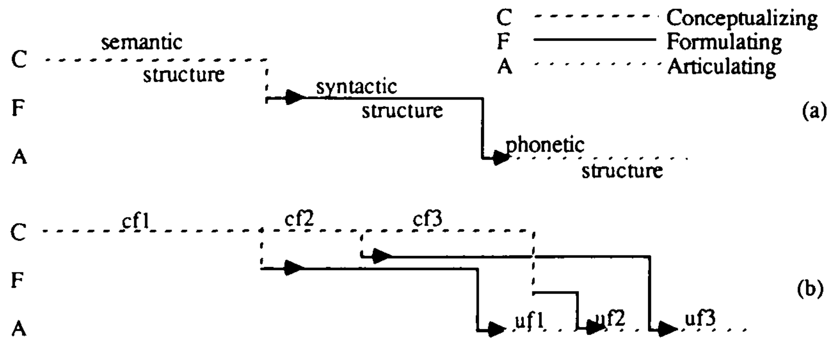


Figure 1. Two theoretically possible alignments of conceptualizing, formulating and articulating processes (cf=conceptual fragment, uf=utterance fragment).

to a schedule where long periods of idling (as long as the conceptualizer is busy) alternate with bursts of hectic work (when an entire conceptual structure has to be converted into a complete sentence). A different argument for the existence of incremental sentence production derives from the assumed independence of conceptualizer and formulator: The former has no way of knowing in advance whether a conceptual fragment it is about to release has the right size to fit exactly into a sentence.

The foregoing analysis of the sentence production process in human speakers imposes some important constraints on the shape of possible mechanisms for building syntactic structures (see also Kempen & Hoenkamp, 1982). Let us start out from the customary assumption that syntactic structures can be represented by tree-shaped diagrams where nodes stand for constituents (e.g., sentence, noun phrase, and prepositional phrase) and arcs indicate membership relationships between phrases (constituents). Left-to-right order of nodes is immaterial for the moment.

The first constraint derives from the fact that it is conceptual structures which serve as input to the tree formation process. In the linguistic literature a great deal of attention is given to the problem of mapping from syntactic structures into logical form. The converse problem—mapping from logical into sentence form—is not a very active area of research. This situation is paralleled in Artificial Intelligence where language parsing and understanding are intensely studied, in contrast with the field of language generation which has only recently begun to gain systematic interest (Mann, 1982). The approach we have taken consists of designing a tree formation module which is sensitive to

(a) properties of the input conceptual structure representing the to-be-expressed meaning, and
(b) properties of the lexical items rendering this meaning.

That indeed both these factors must be taken into account is borne out by examples like (1a–e).

(1a) John wanted to hit Peter.
(1b) want(actor: John)(object: hit(actor: John)(patient: Peter))
(1c) John knew he hit Peter
(1d) know(actor: John)(object: hit(actor: John)(patient: Peter))
(1e) *John wanted he hit Peter.

Consider sentence (1a) and its semantic representation (1b), which expresses the fact that "John" is actor to both "want" and "hit." Any notational scheme for representing sentence meanings must be capable of bringing out such coreferentialities. Now suppose that "want" is replaced by "know," and look at the sentence which expresses the altered meaning (1c–d). Notice that the two complement clauses have different shapes. The verb *want* cannot take a finite complement under the described coreferentiality conditions (see (1e)). If the tree formation component would only have access to conceptual representations, so that the differing lexical properties of the English verbs *want* and *know* were out of reach, then there would be no basis for deciding between finite versus infinitival complement clauses. On the other hand, if only the information stored in the lexical entries for *want, hit, John, Peter,* and so forth, could be accessed, then it would be impossible to assign these words their correct syntactic functions (subject, direct object, etc.) because these functions depend on conceptual roles.

However, can't such decisions be postponed? Why couldn't one construct such provisional (sub)trees in parallel and throw away those which, for whatever reason, turn out to be unsuitable? Or why not arbitrarily choose one possibility and take the risk of having to revise that choice at a later point in time? The answer is that both these proposals violate the *Determinism Hypothesis* in the sense of Marcus (1980). There is no evidence—introspective or behavioral—that speakers necessarily engage in backtracking or multiple tree formation when planning utterances such as (1a–b). For example, it seems unlikely that speakers first build trees corresponding to (1a) as well as (1e) and only afterwards decide that one of them is wrong, or that they start with the alternative corresponding to (1e) and subsequently transform it into—or replace it by—the correct one. For reasons of parsimony we prefer a theoretical approach that invokes the assumption of multiple tree formation or backtracking only when empirical evidence to that effect is conclusive (e.g., as witnessed by the introspection of speakers, by speech errors of the "fusion" or "blending" variety, or by overt self-corrections).

Examples like (1a–e) in conjunction with the Determinism Hypothesis force us to conclude that the tree formation component is *both conceptually and lexically guided.*[1] Categorial Grammar is the only linguistic model of grammar which may be qualified as lexically guided. However, Categorial Grammars do not generate incrementally. And tree formation systems which do have some capacity for incremental production (Augmented Transition Networks, McDonald's (1980) MUMBLE program) fail to meet the requirements of strict determinism or lexical guidance. We are not aware of any type of grammar which satisfies all basic criteria developed in this section. In as much as other models of grammar have aimed at "psychological reality," they have been preoccupied with issues of learnability or parsability in circumstances characteristic of human language learners or language users

[1] The idea that the sentence construction process is at least partially guided by grammatical properties of individual lexical items, is certainly not new. For instance, see the following quote from Miller & Chomsky (1963): "There is no reason to think that a speaker must always select his major phrase types before the minor subphrases or his word categories before his words" (p. 474). An empirical defence of the thesis of lexical guidance of the sentence formulation process was recently given by Harley (1984).

(Kaplan & Bresnan's (1982) Lexical-Functional Grammar; Gazdar's (1981) Generalized Phrase Structure Grammar). Whether the IPG model which we propose here leads to parsable and learnable grammars remains to be investigated.

The two final constraints we wish to discuss in this paragraph are less basic because they are implied by the previous ones. We have assumed that the order of conceptual fragments delivered by the conceptualizer does not depend on the order of the corresponding syntactic fragments. With the possible exception of languages with extremely flexible word order, grammar rules do not always permit a new syntactic fragment to be simply appended to the right-hand side of the current tree. Other spatial arrangements of the new fragment with respect to the current syntactic tree are possible, depending on the word order rules of the grammar. Sometimes, these rules even require the presence of other elements between the current tree and a newly computed syntactic fragment. A clear example is provided by the position of verbs in main clauses of Dutch and German. Subject noun phrases and adverbial phrases cannot follow each other at the beginning of a main clause. The finite main verb or auxiliary is always inbetween: either NP-V-AdvP or AdvP-V-NP, but not NP-AdvP-V or AdvP-NP-V. Grammars which use some version of traditional phrase-structure rules capture such regularities in a rather implicit manner. In fact, they do not keep constituent order apart from phrase membership (constituent hierarchy). It has been proposed to disentangle the two aspects by means of a split between rules which generate "mobiles" and rules which impose a left-to-right order on branches. So far, this alternative has not proven to be superior. However, the phenomenon of incremental sentence production creates a new situation. For example, consider the following phrase-structure rules which express the above word order contingencies:

$S \rightarrow NP + V + AdvP$
$S \rightarrow AdvP + V + NP$

Now, suppose that the formulator is processing a conceptual fragment which lexicalizes into a verb and applies the first rule which says, among other things, that the verb needs an NP at its left-hand side. In the meantime, a new conceptual fragment has arrived which receives the syntactic shape of an AdvP. The first rule does have an AdvP slot, but not to the left of the verb. This implies the formulator has to wait for a third conceptual fragment which can be worded in the form of an NP. At that point the formulator can deliver its first output: an NP-V-AdvP utterance. The waiting time, that is, the period between onset of (conceputal) input and onset of (syntactic) output, would have been shorter, had the formulator picked the second phrase-structure rule. Then, output could already have begun after the second conceptual fragment ("AdvP-V. . .") and closed off grammatically with ". . .NP". Because the order of conceptual fragments is unknown in advance, the formulator can never be sure of having made the best choice between rules. This problem does not arise in a rule system which allows word order to be computed independently of phrase membership. We conclude, therefore, that in an incremental sentence formulator it is desirable to have separate components for tree (or rather "mobile") formation and for word order.

The last constraint follows from the previous one. Suppose the tree formation module applies the rule

VP→V NP NP

which leaves left-to-right order of constituents undefined. Some distinguishing features on the two NPs will then be required in order to assign them correct word order and/or correct morphological case. An obvious possibility is to introduce *functional* notions, for example, to label them direct and indirect object, respectively. We have opted for this solution, thereby committing ourselves to syntactic structures which are somewhat similar to those proposed within the framework of Functional Grammar (Dik, 1978), Lexical-Functional Grammar (Bresnan, 1982), and Relational Grammar (Cole & Sadock, 1977).

2. SENTENCE FORMULATION IN TWO STAGES

On the basis of an extensive study of speech errors, Garrett (1975, 1980) has developed a two-stage model of the sentence formulation process. He had made some important observations on speech errors of the exchange type: word exchanges (2a–b) and combined-form exchanges (2c–d).

(2a) She donated the LIBRARY to the BOOK.
(2b) . . . read the newspapers, WATCH the radio, and LISTEN TO t.v.
(2c) I'm not in the READ for MOODing.
(2d) She's already TRUNKed two PACKs.
(2e) *She's already packS two trunkED.

The former type of exchanges affects full words, including their inflectional morphemes (e.g., plural ending). The latter type is also termed "stranding errors" because the inflectional morphemes are left behind. Exchanges of inflectional morphemes, as in the hypothetical case (2e), were conspicuously absent.

Garrett's observations were the following. The interchanged elements in *word* exchanges (2a–b)

(a) nearly always are members of the same word class,
(b) have similar syntactic functions in the sentence, and
(c) may be far apart in the surface tree, sometimes even belonging to different clauses.

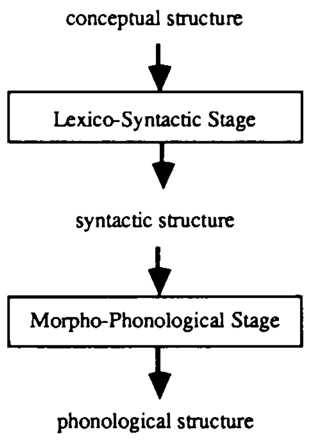
None of these properties apply to combined-form exchanges. Assuming that "computational simultaneity" is a general condition for an interchange between elements, Garrett hypothesized that word exchanges "represent interactions of elements at a level of processing for which functional relations are the determinant of 'computational simultaneity' ", whereas combined-form exchanges "represent interactions at a level of processing for which the serial order of the elements of an intended utterance is the determinant of computational simultaneity" (1975, p. 154).

The next step was to postulate two successive processing stages, called Functional and Positional, respectively, corresponding to the "levels of processing." During the first stage, the syntactic skeleton for an utterance is constructed specifying hierarchical and functional relationships among constituents. The syntactic skeleton does not contain any closed-class lexical material (function words, inflectional morphemes), and word order is still open. The Functional Stage works on all constituents more or less simultaneously. The Positional Stage assigns the constituents a left-to-right order and enriches them with closed-class items, traversing the sequence of constituents from left to right.

In our IPG model we have adopted the essentials of Garrett's proposal. The only deviation concerns the stage which is responsible for inserting function words (i.e., those closed-class items which have word status) and for computing word order. We have allotted these (syntactically interrelated) tasks to the first, Functional Stage rather than to the second, Positional Stage. Our reason derives, among other things, from the observation that exchanged words often carry along dependent function words. For instance, the preposition *to* in (2b) was not stranded but moved along with *listen* (cf. (68) in Section 7).

In diagram (3) we summarize the resulting make-up of the sentence formulation process, using our own terminology.

(3)



3. RULES AND MECHANISMS OF INCREMENTAL PROCEDURAL GRAMMAR

This Section describes in detail the machinery which IPG deploys in constructing sentences expressing a speaker's intention. The Lexico-Syntactic Stage will receive most attention.[2] The Morpho-Phonological Stage will be briefly discussed at the end.

3.1 Preliminaries

In the linguistic and psycholinguistic literature it is commonly agreed that the notions of "syntax" and "syntactic processor" should be kept carefully apart. The difference is usually construed as an instance of the prototypical "database" versus "processor" distinction. The database contains rules of syntax which the syntactic processor can access and utilize for the purpose of computing correct sentence forms. The distinction has been invoked in attempts to explain why linguistic operations as defined in existing grammar types have been unsuccessful in accounting for language performance data. It encourages linguists to claim that their grammatical models only concern the database (knowledge of the language). Psychologists can use it as an excuse to concentrate on processing issues and loose interest in grammar. The drawback is that we are left with two disparate partial theories of the human language faculty whose relationship is not easy to understand.

We take a different perspective. We will describe a sentence construction model which integrates assumptions about data (rules of syntax) with assumptions about the processor manipulating the data. This combined strategy offers an important additional advantage. It opens up the possibility of accounting for linguistic phenomena not in terms of grammar rules but in terms of structure and functioning of the syntactic processor. In Section 4.4, for example, we will argue that syntactic Locality Constraints follow from the normal, independently motivated functioning of the syntactic processor and need not be specified explicitly as an addition to the grammar rules (cf. Marcus, 1980, for a similar approach in the context of syntactic parsing). This position entails a deviation from the well-known *competence hypothesis* which holds that, in language user's cognitive system, there is a grammar representing the (i.e., all) tacit knowledge of his/her language. (For a recent formulation and defense of the competence hypothesis, see Bresnan & Kaplan, 1982.) Any model which articulates—preferably empirically grounded—assumptions about *both* format of grammar rules *and* structure and functioning of the syntactic processor, we call a *procedural*

[2] We have worked out an IPG fragment for Dutch covering many of the constructions discussed in this paper. The grammar fragment has been implemented in a LISP program running on a VAX11/780 computer (dialect Franz Lisp). A completely new Common Lisp implementation running on the Symbolics Lisp Machine is being written by Koenraad De Smedt.

grammar. The specific variety put forward in this paper we have termed "Incremental Procedural Grammar" or IPG because it features incremental sentence production as explained in Section 1.

3.2 Syntactic Procedures

All procedural grammars for sentence production that exist to date have been developed in predominantly nonpsychological contexts. Well-known examples are artificial sentence generators such as the ones proposed by Yngve (1960) and McDonald (1980), and those based on Augmented Transition Networks (ATNs; e.g., Simmons & Slocum, 1972). These models contain a centrally controlled processor which grows syntactic trees in a depth-first, left-to-right manner, at every node consulting the rules of the database. However, this processing schedule entails temporal properties which are at odds with the speech error phenomena discovered by Garrett (1975). He explains word exchanges as exemplified by (2a-b) in terms of "computational simultaneity" between direct object and modifier phrases (2a) or between the verbs in two successive coordinate clauses (2b). Production models which operate left-to-right certainly do not process such constituents simultaneously since the interchanged words may be at considerable distance in the utterance. A model operating *breadth-first* and left-to-right probably fares somewhat better (see Kempen, 1978), but the ultimate solution clearly requires machinery for growing branches of a syntactic tree *in parallel*. Actually, the notion of parallelism is quite congenial with the grammar rules themselves. For example, there is nothing in the rules for generating the direct and indirect object of a clause to suggest that either constituent should be constructed before the other. So the motivation for choosing any particular order must stem from somewhere else.

The basic step towards a mechanism for parallel branch construction, is to view symbols such as NP, N, SUBJECT, OBJECT, and so forth, not as passive structural elements but as *active procedures* or *modules*. Each procedure is an "expert" specialized in assembling one type of syntactic constituent. For example, procedure NP knows how to build noun phrases; procedure PP can deliver prepositional phrases; procedure SUBJECT is responsible for the shape of subject phrases. Like procedures or routines in ordinary computer programs, syntactic procedures are permitted to call on each other as subprocedures ("subroutines"). Procedure S, for instance, may decide to delegate portions of its sentence formation job to SUBJECT and OBJECT as subprocedures. OBJECT need not necessarily wait for SUBJECT to finish: They can get started simultaneously and run in parallel. They, too, are free to call further subprocedures, a typical candidate being NP. Thus a hierarchy of procedure calls arises which is conveniently (and conventionally) depicted as a tree. To illustrate, (4) is the procedure call hierarchy for Dutch sentence (5). The numerical subscripts serve to distinguish various *instantiations* of the same syntactic procedure in a hierarchy. We will always number instantiations consecutively in depth-first, left-to-right order. Differences in depth as expressed by numerical subscripts will be consequential for the tree formation process. Left-right differences never influence tree formation. (Subscripts will be dropped if this does not lead to confusion.)

(4)

$$\begin{tikzcd}
 & \text{S1} \arrow[dl] \arrow[d] \arrow[dr] & & & 1 \\
\text{Subj1} \arrow[d] & \text{VFin1} \arrow[d] & \text{Obj1} \arrow[d] & & 2 \\
\text{NP1} \arrow[d] & \text{V1} & \text{NP2} \arrow[dl] \arrow[dr] & & 3 \\
\text{NPHead1} \arrow[d] & \text{Det1} \arrow[d] & & \text{NPHead2} \arrow[d] & 4 \\
\text{N1} & \text{Art1} & & \text{N2} & 5
\end{tikzcd}$$

(5) Tonnie bakt een cake.
Tony bakes a cake.

Before explaining what syntactic procedures do and why they are stacked hierarchically, we wish to introduce a distinction between two groups of procedures: categorial procedures ("CPROC") and functional procedures ("FPROC"). Informally, CPROCs are capable of building structures of various syntactic shapes (NP, S, PP, etc.); FPROCs take care of the grammatical (functional) relations between such structures (e.g., subject, object, modifier). In (6a) we have listed the most important procedures along with indications of the constituents they deliver.

(6a) Categorial procedures (CPROCs):

S	clause
NP	noun phrase
PP	prepositional phrase
AP	adjectival or adverbial phrase[3]
V	main verb
Aux	auxiliary verb
N	noun
A	adjective or adverb
P	preposition
Art	article
Conj	subordinating conjunction

[3] In this paper we will treat adjectival and adverbial constituents as one "family of phrases."

Functional procedures (FPROCs):

VFin	finite verb
VInfin	infinitive verb
Subj	subject
Obj	object
IObj	indirect object
SMod	sentence modifier
Comp	complementizer
NPHead	head of noun phrase
NMod	noun phrase modifier
Det	determiner
PPHead	head of prepositional phrase
PObj	prepositional object
PMod	prepositional phrase modifier
APHead	head of adjectival or adverbial phrase
AMod	modifier in adjectival or adverbial phrase

Categorial procedures come in two varieties: *phrasal* CPROCs and *lexical* CPROCs. The latter correspond to the traditional parts of speech (V, Aux, N, A, etc.), the former to major phrase types as commonly distinguished in current linguistic practice: S, NP, PP, and AP. The columns of (6b) show that the functional and categorial procedures can be grouped into four nonoverlapping families (phrase types). The rows contain listings of (a) phrasal CPROCs, (b) functional procedures, and (c) lexical procedures for each family.

(6b)

	clauses	noun phrases	prepositional phrases	adjectival or adverbial phrases
(a)	S	NP	PP	AP
(b)	Subj, Obj, VFin, VInfin, IObj, SMod, Comp	NPHead, NMod, Det	PPHead, PMod, PObj	APHead, AMod
(c)	V, Aux, Conj	N, Art	P	A

Procedure call hierarchy (4) illustrates the alternation of CPROCs and FPROCs within branches. The even-numbered levels of the hierarchy are FPROCs, the odd-numbered ones are CPROCs. The terminal nodes are lexical procedures which never call any subprocedures. All other procedures call at least one subprocedure.

3.3 Lexicalization

In order to convey a global idea of the tree formation process in IPG we will sketch out how sentence (5) is generated from conceptual structure (7).

(7) bake(actor: Tony)(product: cake)

As for the conceptual structures serving as input to IPG's tree formation component, we use an informal case-frame notation similar to what one tends to find in the literature on semantic representation. Such structures contain slots or "regions" whose contents are accessible through *path functions*. A path function traces a path through a conceptual structure and returns the content encountered at the end of the path. When applied to structure (1b), the path "object-patient" leads to the concept "Peter" (i.e., "Peter is the patient of the object of (1b)"). In Section 4 we introduce some further notational conventions. Geurts (1984) has developed a logical calculus which is fully compatible with IPG. It appears that IPG does not thrust many constraints upon the representational system for specifying to-be-expressed meanings. This is an interesting consequence of the lexical guidedness of IPG's tree formation component. On the basis of arguments put forward in Section 1, we assume a *lexicalization system* whose task it is to inspect conceptual structures (often using path functions) and to look up from the mental lexicon words or expressions rendering the speaker's intention.[4] It is the "lexicalizer" which starts up the tree formation process. After that, the conceptual structures only play a minor role, namely, when it comes to inflectional computations and to the insertion of function words. In this respect, IPG differs considerably from Generative Semantics, where the semantic representation of a sentence is identified with its initial syntactic tree.

The standard format of a syntactic procedure call is

PROC(cp, <synspec>)

where

PROC	is the name of a categorial or functional procedure;
cp	("conceptual pointer") a variable or an expression evaluating to a conceptual structure; and
<synspec>	("syntactic specification") a list of zero or more calls to special functions which influence the shape of the constituent that PROC will build.

[4] How the lexicalization system works we will not discuss here, nor how it makes a choice between alternative (synonymous) words or idioms. For an example of a computational lexicalization system which operates on Schank's (1975) conceptual dependency structures, we refer to Goldman (1975). In the remainder of this paper we will take successful lexicalization of the input conceptual structure for granted.

Among the first actions taken by a procedure is lexicalization. The retrieved lexical entries are procedural in nature, that is, they consist of a list of one or more procedure calls. We will denote such entries by the term *lemma* to distinguish them from *lexemes,* a second type of entry to be introduced in the next paragraph. Examples of lemmata are given in (8)–(10).

```
(8)   V(nil, <Lex(bake)>)
      Subj(Path(actor:), < >)
      Obj(Path(product:), < >)

(9)   N(nil, <Lex(Tony)>)

(10)  N(nil, <Lex(cake)>)
```

Lemma (8) represents the English active transitive verb *bake.* It consists of calls to procedures V, Subj and Obj. The first argument of every procedure call, cp, evaluates to a conceptual structure (accessed via a path function) or to NIL (if there is nothing to lexicalize). This conceptual structure is the meaning that the associated procedure is going to express. In (8), the paths associated with the Subj and Obj calls lead to, respectively, actor and product of the baking activity. We assume that all procedure calls mentioned in a lemma are members of the same "family" in the sense of (6).

The second argument to procedure calls in lexical entries is a synspec list. A typical function to be found there is Lex which takes as its single argument a pointer to a *lexeme.* Lexemes are lexical entries which specify phonological shapes for words (e.g., *bake, Tony, cake*). Lex always occurs on the synspec list of lexical procedures (V, Aux, N, P, etc.).

Procedure call hierarchy (4), which leads to sentence (5), is composed as follows. The construction of main clauses is initiated by the standard procedure call

S(cp, <Main>)

where Main is a zero-argument synspec function which causes S to select main clause word order, that is, to place the finite verb in "verb-second" position. When called without this synspec function, S uses the default position for finite verbs, namely, "verb-final". (The mechanism for computing word order rules will be discussed in Section 3.5.) S's first argument now points to conceptual structure (7), which lexicalizes into the Dutch translation of lemma (8). The resulting situation is depicted in snapshot (11).

```
(11)  Lexicalizing Procedure:           S(cp, <main>)
                                       ______________________
                                      ⎧ V(nil, <Lex(bakken)>)
      Procedures listed in lemma:     ⎨ Subj(Path(actor:), < >)
                                      ⎩ Obj(Path(product:), < >)
```

Two procedures below the line refer to functions or roles played by major constituents with in a clause: Subj and Obj. The third procedure, V, first needs to be assigned a function within the clause. Eligible are the roles of finite verb or infinitive verb: VFin or VInfin. In this case, S selects VFin on the basis of so-called Appointment Rules (see Section 3.4). The partial hierarchy depicted in (12) is the result.

(12)

$$\begin{tikzcd}
 & \mathrm{S} \arrow[dl] \arrow[d] \arrow[dr] & \\
\mathrm{Subj} & \mathrm{VFin} \arrow[d] & \mathrm{Obj} \\
 & \mathrm{V} &
\end{tikzcd}$$

We define successful lexicalization as the retrieval of exactly one lemma covering at least part of the to-be-expressed meaning. (When fewer or more lemmas turn up, hesitations or speech errors such as word blending might ensue.) Any noncovered fragments of the conceptual structure are assigned to *modifier* procedures. To this purpose, the four phrasal CPROCs S, NP, PP, and AP have at their disposal the procedures SMod, Mod, PMod, and AMod, respectively. The cp arguments these FPROCs receive point to conceptual fragments which do not fit under the looked-up lemma. Consider sentence (13) and assume that the lexicalization process within S yields the *bakken* lemma which leaves unexpressed the meaning underlying *in een oven.*

(13) Tonnie bakt een cake in een oven.
Tony bakes a cake in an oven.

This left-over conceptual fragment, let us denote it by cp2, is then passed on to SMod:

SMod(cp2, < >)

How the SMod constituent is given the shape of a prepositional phrase is explained in Section 3.4. The relevant portion of the final procedure call hierarchy is given in (14).

(14)

$$\begin{tikzcd}
 & \mathrm{S} \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
\mathrm{Subj} \arrow[d] & \mathrm{VFin} \arrow[d] & \mathrm{Obj} \arrow[d] & \mathrm{SMod} \arrow[d] \\
\mathrm{NP} & \mathrm{V} & \mathrm{NP} & \mathrm{PP}
\end{tikzcd}$$

3.4 Appointment Rules and Functorization Rules

The construction of procedure call hierarchies is governed by a set of *Appointment Rules.* They specify possible shapes of such hierarchies by telling each procedure call contained in a retrieved lemma which role it is going to play within the context of the lexicalizing procedure. The general format of Appointment Rules is the following:

PROC1, PROC2, <cond*1*, cond*2*,..., cond*n*> → PROC3> >PROC2

PROC1 is the lexicalizing procedure, PROC2 is mentioned in a lemma (i.e., is seeking a role to play within PROC1). The third element of the left-hand side of an Appointment Rule is a list of zero or more conditions. The right-hand side prescribes which procedure will be PROC2's parent in the hierarchy. PROC3 may be identical with PROC1. The symbol "> >" means "is parent of." The examples we have just met are:

S, Subj, < > → S> >Subj
S, Obj, < > → S> >Obj
S, V, <No-other-VFin> → VFin> >V
S, VFin, < > → S> >VFin

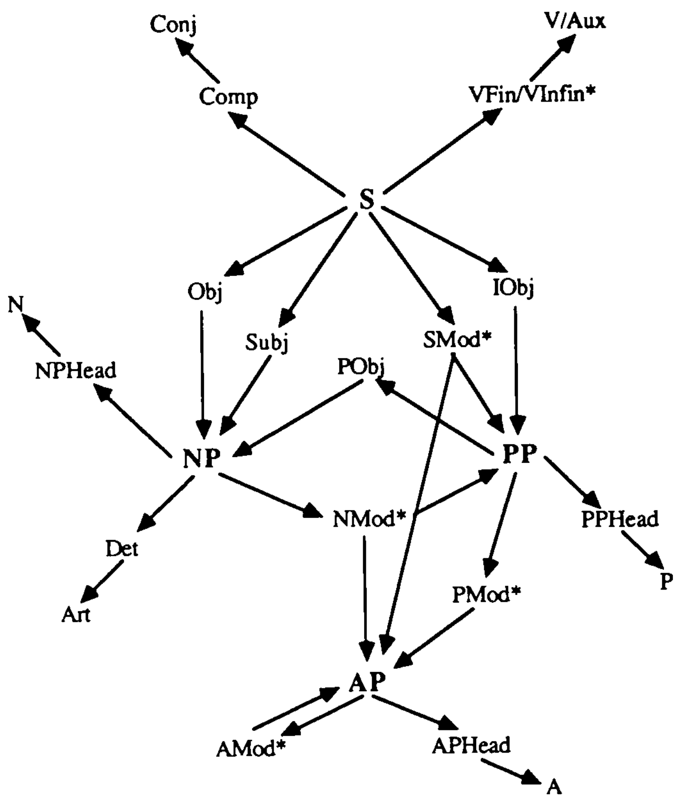
The first of these rules states that Subj is immediately acceptable as a daughter node to S. The third rule assigns VFin as V's parent, that is, appoints V in the function of the clause's finite verb. The last rule attaches VFin to S, thus establishing a link between lexicalization result V and lexicalizing procedure S.

Rather than listing a long set of Appointment Rules we present a graph which summarizes them in a convenient manner (Figure 2). Notice that the Appointment Rules sometimes lead to rather long chains of intercalated nodes. See, for example, the rules for dealing with <NP, A> and <S, N> pairs which are rendered here in abbreviated form.

NP, A, < > → NP> >NMod> >AP> >APHead> >A
S, N, <No-other-Subj> → S> >Subj> >NP> >NPHead> >N

The former rule assigns to A the role of head of an AP which is a modifier within NP. The latter rule tells S that N can fulfill the function of head of a subject noun phrase provided there is no other subject around. By tracing a route, in Figure 2, from a PROC1 to a PROC2, one finds the names of any procedures to be intercalated. For instance, <S, V> pairs are handled in the upper right-hand branch of the graph. If more than one route exists between a pair of procedure names, then select the shortest one; if several routes of equal lengths are possible, apply the relevant rule in the bottom left-hand corner. Some further restrictions are given in the legends to Figure 2.[5]

[5] The Appointment Rules rendered in Figure 2 are "default" rules in the sense that they can be superseded by more specific rules adduced from the lexicon. In Section 4.1, when discussing object complement constructions, we will see that certain lemmas specify local exceptions to the basic Appointment Rules. For example, the lemmas for verbs of perception (*see, hear, want, think*) indicate that not only NPs (the default) but also Ss can serve as object constituent. We implement this using an object-oriented extension of LISP with "inheritance".



Special Appointment Rules	
S, NP member	: if no Subj then Subj, otherwise SMod.
S, V/Aux	: if no VFin then VFin, otherwise VInfin.
S, PP member	: SMod

Figure 2. Appointment Rules for the syntactic procedures listed in (6). Asterisked FPROCs are permitted to occur more than once in a list of subprocedure calls. For example, S may issue several calls to SMod and VInfin.

Let us now continue with the construction of the subject and object branches of procedure call hierarchy (4). FPROCs Obj and Subj proceed by lexicalizing their portions of the input conceptual structure, that is, actor and product, respectively. In both cases, the lexicon suggests a lemma which only contains a call to procedure N (lexical entries (9) and (10)). Appoint-

Those who are familiar with Marcus's (1980) parser will have noticed that our Appointment Rules work much the same way as his node attachment rules. What we call a lexicalizing procedure is the bottom node on his active node stack, and the procedure calls in a lemma are comparable to the constituents which fill the slots of his buffer. For example, our way of inserting VFin inbetween S and V is similar to the way Marcus intercalates VP between S (active node stack) and V (buffer).

ment Rules know how to handle these situations: inspect Figure 2 for the <Subj, N> and <Obj, N> pairs.

The present example raises an important further issue, namely, how *function words* (articles, prepositions, auxiliaries, etc.) come into play. Their presence in an utterance is chiefly motivated on syntactic grounds, so they cannot be supposed to originate simply from lexicalization. The same conclusion follows from the well-known linguistic fact that function words are often in complementary distribution with *inflections*. For instance, in English as well as in Dutch, the present and past tenses of verbs are indicated by inflectional morphemes, whereas the future requires an auxiliary, a morpheme with word status. A convenient term covering both groups of syntactic morphemes is *functor,* and we propose the term *functorization* to denote the process of inserting functors.

Functorization is best characterized as refining the set of procedure calls contained by a lemma. This may happen in two different ways, corresponding to the distinction between inflections and function words. The refinement either affects the synspec list of a procedure call by inserting a new function there, or it supplements the current set of subprocedure calls with an additional member. In the former case, the synspec function will influence the inflectional shape of the resulting constituent; in the latter, a separate function word will emerge.

The notation we use for functorization rules is similar to the one we proposed above for Appointment Rules. Consider the following Functorization Rules:

S, V, <Time-past...> → V(nil, <...Tense-imperfect...>)
S, V, <Time-future...> → V(...)^Aux(nil,<Lex(will),Tense-present>)

The triples at the left-hand side are

(a) the name of the lexicalizing procedure,
(b) the name of a procedure mentioned in the lemma,
(c) a list of one or more conditions; in particular, conditions relating to the conceptual structure which the lexicalizing procedure is trying to express.

The right-hand side specifies the refinement. The first rule drops a Tense function into V's synspec list. The second rule leaves V untouched but adds a call to Aux complete with two synspec functions. The symbol "^" denotes the "sister procedure" relation. Functorization has to take place prior to application of any Appointment Rules because it sometimes leads to additional subprocedure calls which need to be assigned a role within the lexicalizing procedure.

We can now take up sentence (13) again. The conceptual structure bound to cp2 lexicalizes into noun lemma

N(nil, <Lex(oven)>)

The Appointment Rules of Figure 2 reveal that a fairly long chain of procedure calls needs to be intercalated between SMod and N:

SMod> >PP> >PObj> >NP> >NPHead> >N,

telling SMod to call in the help of subprocedure PP. How does PP deal with the N lemma? First of all, Functorization Rule (15) triggers in response to certain locative information present in cp2.

(15) PP, N, <Loc-inside> → N(...) ^ P(nil, <Lex(in)>)

Two Appointment Rules are applicable now for <PP, N> and <PP, P>, respectively. On the basis of Figure 2, P and N are given the roles of PPHead and PObj. This entails procedure call hierarchy (16).

(16)

$$\begin{tikzcd}
 & \text{SMod} \arrow[d] & \\
 & \textbf{PP} \arrow[dl] \arrow[dr] & \\
\text{PPHead} & & \text{PObj}
\end{tikzcd}$$

PPHead is permitted to call lexical procedure P, which will terminate one branch of (16). Within the other branch, one further Functorization Rule triggers at the level of NP:

(17) NP, N, <Ref-indefinite, Number-singular>
→
N(...) ^ Art(nil,<Lex(een)>)

It inserts the indefinite article in case the conceptual structure refers to an indefinite singular entity. NP proceeds by applying Appointment Rules <NP, N> and <NP, Art>. The complete procedure call hierarchy dominated by SMod finally looks like (18).

(18)

$$\begin{tikzcd}
 & \text{SMod} \arrow[d] & & \\
 & \textbf{PP} \arrow[dl] \arrow[dr] & & \\
\text{PPHead} \arrow[d] & & \text{PObj} \arrow[d] & \\
\begin{array}{c}\textbf{P}\\ \textit{in}\end{array} & & \textbf{NP} \arrow[dl] \arrow[dr] & \\
 & \text{Det} \arrow[d] & & \text{NPHead} \arrow[d] \\
 & \begin{array}{c}\textbf{Art}\\ \textit{een}\end{array} & & \begin{array}{c}\textbf{N}\\ \textit{oven}\end{array}
\end{tikzcd}$$

Although functorization clearly is a different kind of process than lexicalization, the division of labor between them—activation of function words and of content words, respectively—is less clear. Both English and Dutch have many words which according to their grammatical class are to be regarded as function words but whose meaning is so salient that they could justifiably be labeled content word. An example is the preposition *zonder* (Eng. *without*). The converse case occurs as well: words whose grammatical class grants them the status of content word although they are interchangeable with a function word in many syntactic contexts, for example, the Dutch adjective *zeker* (Eng. *certain*). Such observations on the vagueness of the boundary between function and content words force us to devise normal lexical entries (i.e., lemmata) for prepositions like *without* on one hand, and Functorization Rules which lead to inserting adjectives like *zeker* on the other.

(19) P(nil, <Lex(without)>)
PObj (Path(. . .), < >)

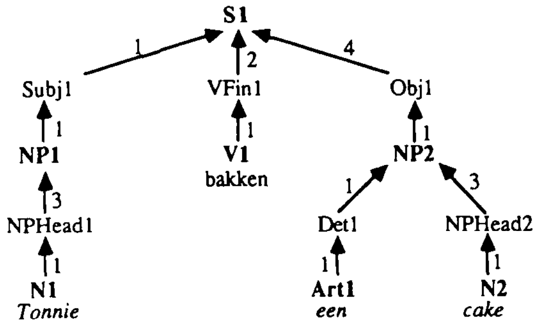
(20) Tony baked a cake without an oven.

The construction of sentence (20) differs from that of (13) in the way PP gets hold of its preposition. The lexicalization process provides SMod with lemma (19). The procedure calls listed there lead to intercalating subprocedure PP which assigns P the role of PPHead via an Appointment Rule. So, P is placed in position through lexicalization rather than through functorization. The noun lemma *oven* is retrieved by PObj rather than by SMod as was the case in sentence (13). Notice, however, that the eventual procedure call hierarchies for both sentences are identical. The proposed "mixed" treatment of prepositions corresponds to the distinction between prepositions which are clitics (*of, by, on, in*) and those which aren't (*without, under, after*).

3.5 Combining and Communicating Subtrees

Apart from assembling a list of zero or more subprocedure calls and putting all of them to work simultaneously, a syntactic procedure also has the duty of processing the subtrees they return as their *values*. Top procedure S1 in example (4) receives values representing the subject, finite verb, and object constituents. How does S1 combine these subtrees into a single, grammatical clause? A procedure, we assume, creates a data structure, called *holder*, containing a sequence of numbered positions P1, P2,. . .Pn. Each of these slots can serve as a receptacle for subtrees delivered by a subprocedure. Most types of holder have just one slot. Only holders created by procedures S, NP, PP and AP (the four phrasal CPROCs) contain more than one slot, namely 6, 4, 3, and 2, respectively. Upon receipt of a value (subtree) computed by one of its daughters, a procedure deposits it into a holder slot. (This operation is the IPG version of what is usually called node attachment.) For example, S deposits the subject-np subtree into P1 of the holder it created, the finite verb into P2 and the object-np into P4. These slots are chosen on the basis of a set of *Word Order* Rules which we explain now in terms of the *value return hierarchy* shown in (21).

(21)



The upward arrows denote the operation of returning a value. Their numerical labels refer to slots. For example, the subject-np has been deposited into slot P1 of the holder created by S1 and VFin was assigned second position in the sentence.

For ease of survey we have organized the essential Word Order Rules of Dutch into one large decision tree. The nonterminal nodes of Figure 3 spe-

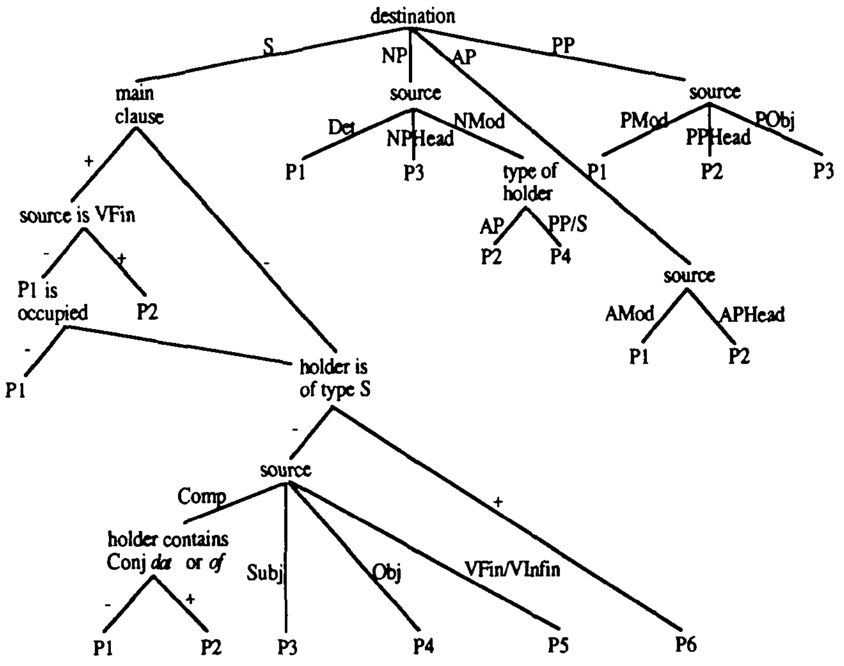


Figure 3. Word Order Rules for categorial procedures S, NP, PP and AP.

cify decisions which relate to properties of the procedure receiving a value ("destination"), or of the value itself ("source"). Arc labels refer to possible outcomes of such decisions. Terminal nodes stand for positions (slots) to be selected by the receiving procedure.

VFin's position was determined through the following sequence of decisions:

type of destination: S
main clause: +
source is VFin: +

which leads to terminal node P2. The subject branch received initial position P1 because, presumably, Subj was the first subprocedure within S1 to return a value.

type of destination: S
main clause: +
source is VFin: – (because it was Subj)
P1 is occupied: – (no other value has occupied P1 yet).

The object branch went to P4 as follows:

type of destination: S
main clause: +
source is VFin: – (Obj instead)
P1 is occupied: + (namely, by the subject-np)
value is of type S: – (NP instead)
source: Obj.

In the foregoing, we have taken for granted that the output value delivered by a procedure consists of the holder created by that procedure together with its contents. Like in ordinary hierarchical computer programs, it is the lowest (innermost, deepest) subprocedure which is first to deliver its output value. In our procedure call hierarchies this is always a lexical procedure. It delivers its one-slot holder after filling it with a pointer to a lexeme. (In Section 3.6 we explain how such pointers are processed by the Morpho-Phonological Stage. In value return hierarchy (21) we have simply substituted Dutch words for them.) The destination selected by all lexical procedures is the parent. For instance, V sends its output value to VFin, N to NPHead, as shown in (21). Actually, all categorial procedures return their value to their parent. This is not true of functional procedures, though.

In Section 4 we will heavily exploit the IPG equivalent of "movement transformations." This is a mechanism which causes procedure call hierarchies to build nonisomorphic value return hierarchies. The resulting syntactic trees are less deep than the procedure call hierarchies which put them together. The mechanism is essentially a set of rules for FPROCs to choose a destination other than their parent (and usually located higher up in the procedure call hierarchy). When computing destinations for their output values, FPROCs utilize the following system for referencing holders created by other procedures. Immediately upon being called, syntactic procedures (both functional and categorial ones) declare a variable whose name consists of the character string "var" prefixed with the procedure's own name. For instance, the variables declared by S, NP and V are s-var, np-var, and v-var, respectively. The value assigned to such a variable is the name of an instantiated procedure (e.g., S1, NP2). The *Destination* Rules used by FPROCs are phrased in terms of such variables. For example, Obj seeks s-var as its destination. This means it climbs the procedure call hierarchy until it hits upon an occurrence of "s-var." Obj then ascertains the name of the instantiated procedure bound to that variable, and sends its value to that address.

In (22) we summarize the Destination Rules discussed so far. We will introduce a few more in Section 4.2.

(22)

Source	Destination
CPROC:	Parent procedure
FPROC	Instantiated procedure bound to:
of S-family:	s-var
of NP-family:	np-var
of PP-family:	pp-var
of AP-family:	ap-var

Under the influence of lexical information, s-var is sometimes given a different value than the name of the S instantiation which declared the variable (see Section 4.1).

We conclude this Section with an overview of the main activities syntactic procedures have to perform:

(23) A. Declare and initialize variables.
B. Create a holder.
C. Evaluate cp and synspec arguments.
D. Lexicalize cp.
E. Apply Functorization Rules.
F. Apply Appointment Rules.
G. Run subprocedures in parallel.
H. Apply Word Order Rules to receive subtrees.
I. Apply Destination Rules.
J. Return holder with contents to destination.
K. Exit.

Terminal (lexical) procedures skip steps D through I.

3.6 The Morpho-Phonological Stage

The output value computed by a *terminal* procedure contains a "lexical pointer" which serves to locate a lexeme in the mental lexicon. A lexeme is a phonological specification of a to-be-uttered word. However, the final shape of the word still awaits the application of inflection rules and of various sound rules which belong to the domain of articulation rather than formulation. The Morpho-Phonological Stage converts syntactic trees delivered by the Lexico-syntactic Stage (more precisely, trees returned by the top member of a procedure call hierarchy) into phonological structures.

As part of their normal work, syntactic procedures compute all information needed by rules of inflection. In Section 3.4 we introduced some of the relevant computations in the context of Functorization Rules. Remember that some of these rules enrich a procedure call with special synspec functions (e.g., "Tense(present)"). Without going into computational details we assume that these functions fit holders with special instructions to be executed by the Morpho-Phonological Stage. During the construction of noun phrases, for instance, information about number and gender is transferred from head to modifiers and article. Synspec functions activated within procedures NP, NPHead, Det, NMod, and so forth, take up this duty. And after NP has passed its output on to, say, procedure Subj, the latter will add instructions to select nominative case. Subj's output value is finally delivered at the holder created by S and thus made accessible to VFin: subject-verb agreement.

Nonstandard case can be assigned to noun phrases by means of synspec functions. A good example is the indirect object of the German verb *fragen* (*ask*) which does not govern dative but accusative case. We could devise a function "Acc" to be put on the synspec list of IObj in the *fragen*-lemma:

IObj(Path(...), <Acc>)

Acc will mark the NP-subtree returned to IObj for accusative case. The Morpho-Phonological Stage will then attach accusative rather than dative inflectional endings to the indirect object noun phrase.

These brief remarks about the workings of the Morpho-Phonological Stage suffice in the present context. In various other papers we have worked out further details on the basis of new experimental psycholinguistic evidence (Kempen & Huijbers, 1983; van Wijk & Kempen, 1987). We also have considered the problem how intonation contours get woven into an utterance. Van Wijk & Kempen (1985) argue that the Morpho-Phonological Stage has an important role to play there and describe the computational system they developed for automatically generating Dutch intonation contours for syntactic structures delivered by the Lexico-Syntactic Stage.

4. AN IPG ANALYSIS OF SOME SYNTACTIC STRUCTURES

In order to illustrate the capabilities of the machinery developed in the previous section, we will present an account of three complex constructions in Dutch: object complement clauses, interrogatives, and coordinate structures.

4.1 Object Complement Clauses

Sentences (24)-(27) contain verbs which take object complements: *willen* (*want*) and *zien* (*see*). Their lemmata are shown in (28).

(24) Tonnie wil een cake bakken.
(Tony wants a cake bake)
Tony want to bake a cake.
want(actor: Tony)(object: bake(actor: Tony)(product: cake))

(25) Tonnie wil dat Marietje een cake bakt.
(Tony wants that Mary a cake bakes)
Tony wants Mary to bake a cake.
want(actor: Tony)(object: bake(actor: Mary)(product: cake))

(26) Tonnie ziet Marietje een cake bakken.
(Tony sees Mary a cake bake)
Tony sees Mary bake a cake.
see(actor: Tony)(object: bake(actor: Mary)(product: cake))

(27) Tonnie ziet dat Marietje een cake bakt.
(Tony sees that Mary a cake bakes)
Tony sees that Mary bakes a cake.
see(actor: Tony)(object: bake(actor: Mary)(product: cake))

(28a) V(nil, <Lex(willen)>)
Subj(Path(actor:), < >)
Obj(Path(object:), < >)
ObjComp1

(28b) V(nil, <Lex(zien)>)
Subj(Path(actor:), < >)
Obj(Path(object:), < >)
ObjComp2

New elements in these lemmata are *Lemma Functions* whose duty it is to refine the list of procedure calls contained by the lemma. In fact, ObjComp1 and ObjComp2 serve to build object complement clauses in two slightly different ways. They do so, given that Obj's cp argument is lexicalized into a V-type lemma like (8). In such a case the standard Appointment Rules of Figure 2 fail to apply since they only provide for NP-shaped object constituents.

(8) V(nil, <Lex(bake)>)
Subj(Path(actor:), < >)
Obj(Path(product:), < >)

ObjCompl overrules this limitation and allows Obj to call S as a subprocedure. Moreover, ObjCompl tells the new S-instantiation (labeled S2 in Figure 4) to carry out three synspec functions listed in (29). These cause S2 to behave in a somewhat anomalous fashion:

(29) *ObjCompl* (infinitival clause):
- (a) the V-lemma is stripped of its call to Subj;
- (b) V is assigned the role of VInfin instead of VFin;
- (c) the s-var declared by the Obj-S is given a nonstandard value: s-var is initialized not to the procedure's own name but to the value of the other s-var that is within reach (by climbing the procedure call hierarchy).

The relevant portion of the computational process is shown in Figure 4. Both s-var variables point to "S1." Accessing its destination address "s-var," VInfin therefore skips the embedded S2 and is "raised" to the level of S1, the matrix S. The same is true of the deepest instantiation of Obj. The value return hierarchy consequently is less deep than the procedure call hierarchy. See Figure 5 for the complete hierarchies underlying sentence (24). Also notice that the (a)- and (c)-parts of ObjCompl accomplish effects that are usually termed *Equi-NP-Deletion* and *Clause Union,* respectively.

In (25), ObjCompl has chosen to develop the object clause into a full-blown subordinate clause. That is, Obj called S without the three synspec functions that are responsible for infinitival clauses. The subprocedures within S2 (see Figure 6) are the result of lexicalization and of the application

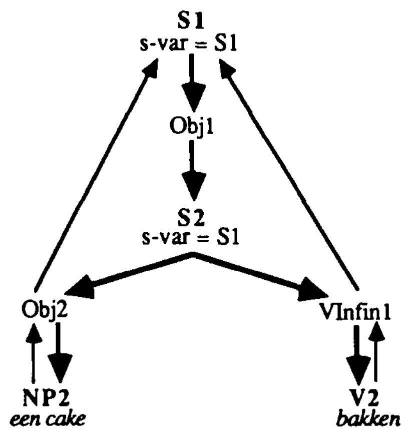


Figure 4. Value return within the OBJ branch of (24).

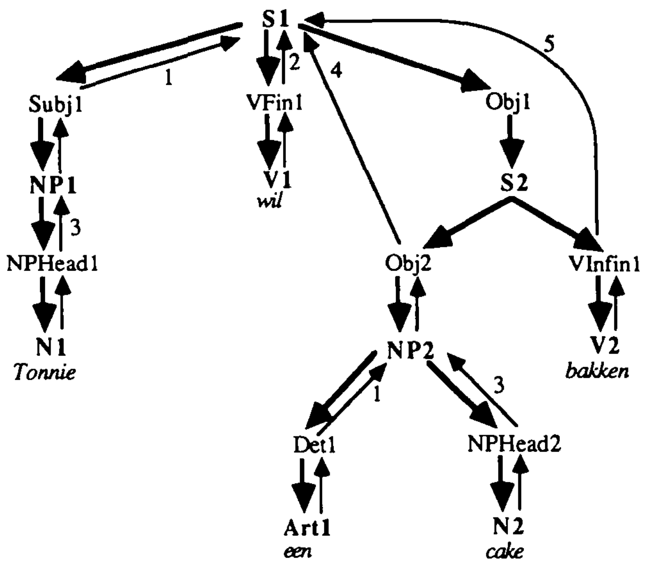


Figure 5. Construction of sentence (24).

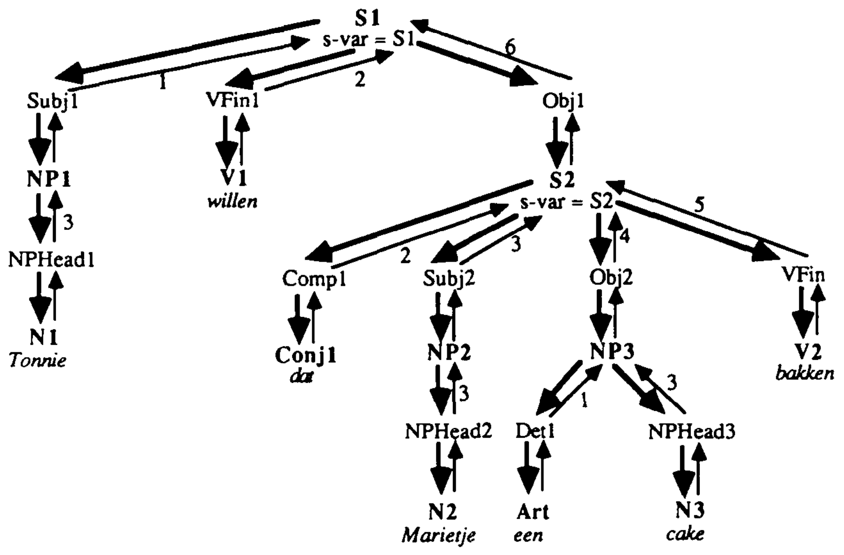


Figure 6. Construction of sentence (25).

of standard Appointment Rules. Subprocedures Comp and Conj were stuck in by a Functorization Rule (whose details we will not go into). Procedures Comp, Subj, Obj, and VFin running within the embedded S2 deliver their values at the holder created by S2 because this is the value of "s-var" from their point of view. The topmost s-var is beyond their *scope* and therefore inaccessible as a destination. On the other hand, the top-level instantiations of Subj, VFin and even Obj are completely screened off from what happens inside of the embedded S and therefore could never select the value of the deeper token of "s-var" as their destination.

Lemma functions ObjCompl and ObjComp2 differ in two respects. First of all, they choose between infinitival and finite object clauses on different grounds. ObjCompl (for *willen*) selects the infinitival variety if the cp arguments of the top-level Subj and of the embedded (deleted) Subj point to the same conceptual structure. In terms of (24), since the path functions in the Subj calls to lexical entries (28a) and (8) delivered the same conceptual structure as to-be-expressed meaning, ObjCompl chose an infinitival object complement. This condition of "coreferentiality" being violated in (25), ObjCompl decided to construct a finite complement clause. (Checking the coreferentiality of two "subject contents" makes part of ObjCompl's duties, we assume.) ObjComp2 (for *zien*) certainly does not use a coreferentiality check. In fact, we do not know which criteria ObjComp2 applies when deciding between finite and infinitival complement clauses. Very often, though, this choice seems rather arbitrary: (26) and (27) are virtually synonymous.

The second difference between ObjCompl and ObjComp2 lies in the way they treat the V-lemma to be used within the Obj branch. This lemma—(8) is an example—always specifies a call to Subj including a cp argument in the form of a Path function. ObjComp2 composes an additional call to Obj which contains a copy of this Path function as its cp argument, and adds it to the procedure calls which are already listed on the lemma of the complement taking verb. The consequence is that *two* Obj procedures will run in parallel. The effect of the whole operation, shown in Figure 7, is comparable with *Subject-to-Object Raising*. The main actions of ObjComp2 are summarized in (30).

(30) *ObjComp2* (infinitival clause):
(a) the V lemma is stripped of its call to Subj; the Path function occurring therein is copied into a new call to Obj which is added to the procedure calls listed in the lemma of the complement taking verb;
(b) see the (b)-part of (29);
(c) see the (c)-part of (29).

What needs an explanation yet is the left-to-right order of the two object NPs that end up in position P4 of the holder of the matrix S. The Word Order Rules of Figure 3 provide no basis for ordering them. The extra rule in (31) makes NP3 follow NP2.

(31) An object NP which stems from a more deeply embedded instantiation of Obj is lined up to the right of a less deep object NP.

In Section 3.2 we introduced a top-down system for numbering instantiations of syntactic procedures (cf. (4)). The system guarantees that, within the

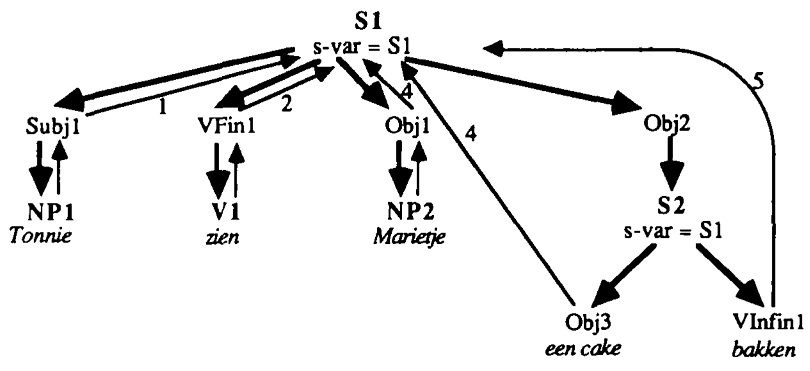


Figure 7. Construction of sentence (26).

same branch of a procedure call hierarchy, the procedure with the highest instantiation number always refers to deepest one. *Mutatis mutandis,* (31) also applies to the finite and infinitive verbs which occupy position P5 of an S-holder: Deeper verbs trail behind less deep ones.[6] These extra Word Order Rules enable a complete analysis of the more complicated cases (32) and (33) containing both a *willen* and a *zien* verb, as the reader may work out for himself.

```
(32)           P4           P5
          ⏞‾‾‾‾‾‾‾‾‾‾‾‾⏞  ⏞‾‾‾‾‾‾‾⏞
      Tonnie wil Marietje een cake zien bakken.
      Tonnie wants Mary    a  cake see  bake)
      (Tony wants to see Mary bake a cake.
      want(actor: Tony)
          (object: see(actor: Tony)
                  (object: bake(actor: Mary)
                               (product: cake)))

(33)           P4           P5
          ⏞‾‾‾‾‾‾‾‾‾‾‾‾⏞  ⏞‾‾‾‾‾‾‾‾‾⏞
      Tonnie ziet Marietje een cake willen bakken.
      (Tony sees Mary    a  cake  want   bake)
      (Tony sees Mary want(ing) to bake a cake.
      see(actor: Tony)
         (object: want(actor: Mary)
                  (object: bake(actor: Mary)
                               (product: cake)))
```

[6] This vinfin variant of (31) is too strict, though, because it rules out certain orders of infinitive verbs which are (marginally) acceptable, for example,

Tonnie heeft een cake bakken willen.
(Tony has a cake bake want).
Tony has wanted to bake a cake.

Our analysis provides a complete account of the Cross-serial Dependencies in Dutch as described by Bresnan, Kaplan, Peters, & Zaenen (1982). Example (34a) has been taken from their paper. The horizontal brackets indicate dependency relationsips between NPs and main verbs. We have added German and English translation equivalents because, despite their widely different dependency configurations, they will receive very similar IPG analyses.

(34a) ... dat Jan Piet Marie zag helpen zwemmen

(34b) ... dass Jan Piet Marie schwimmen helfen sah

(34c) ... that Jan saw Piet help Mary swim

All three structures are generated by the procedure call hierarchy depicted in Figure 8 (downward arrows). The differences between Dutch and German derive solely from the Word Order Rules which control the sequencing of finite and infinitive verbs in position P5 of S-holders. As explained in the context of (31) above, the Dutch rule tells deeper verbs to follow less deep ones. The German rule prescribes just the opposite: deeper verbs have to precede less deep ones. The nested dependency configuration of (34b) is the result. The contrasts between Dutch and English relate to basic Word Order Rules (English is treated as an SVO rather than as an SOV language) on one hand, and to the absence of Clause Union on the other. The latter boils down to the assumption that the lemmata of English complement taking verbs such as *see, want,* and *help* do not contain lemma functions ObjCompl

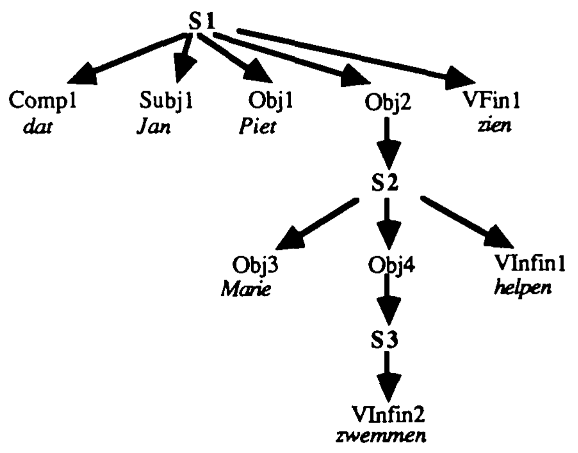


Figure 8. Construction of sentence (34a).

or ObjComp2, but a simpler variant which is lacking the (c)-part of (29). Then, the value of an s-var variable declared by an S-instantiation will always be initialized to the name of that instantiation itself rather than to the name of a higher instantiation. This, in turn, causes the destination selected by Subj, Obj, VFin, VInfin, etc., to be their parent S-instantiation (rather than one higher up in the procedure call hierarchy).

4.2 Interrogatives

The analysis of interrogatives we are going to present concentrates on word order phenomena as observable in Dutch (and, for that matter, German). We will first consider yes/no questions, then wh-questions.

As explained in Section 3.3, main sentences are constructed by executing procedure call

S(cp, <Main>).

Main is a synspec function which puts up a flag signaling S to select main clause word order. It causes the finite verb to be deposited into slot P2 of the S-holder. We assume that S will deliver a yes/no-question if cp points to a conceptual structure tagged with the symbol Q ("Query"). The presence of Q can be detected by Main which responds by putting a question mark ("?") into P1 of S-holder. From the point of view of the Word Order Rules of Figure 3, slot P1 henceforth counts as occupied and can no longer receive any output values returned by procedures lower in the hierarchy (except for wh-constituents, as we will see below). In particular, the subject constituent delivered by Subj is diverted to P3 instead of P1. This causes inversion of subject and finite verb, typical of Dutch and German interrogative *main* clauses (but not of interrogative *subordinate* clauses which are constructed without the intervention of Main). A call to S without synspec function Main yields word order of subordinate clauses. That is, the finite verb will be located at P5 of S-holder ("verb final").

A simple example of subject-verb inversion is sentence (35) which comes about as depicted in Figure 9.

(35) Bakt Tonnie een cake?
(Bakes Tony a cake)
Does Tony bake a cake?
Q<bake(actor: Tony)(product: cake)>

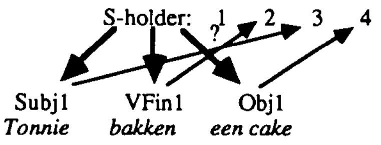


Figure 9. Construction of sentence (35).

Another effect of the Q tag is to trigger certain Functorization Rules. In (36a), the conceptual structure underlying (35) has been worded as object complement of *zien*.

(36a)
Marietje wil zien $\left\{\begin{array}{c}\text{of}\\ \\ \text{dat}\end{array}\right\}$ Tonnie een cake bakt.
(36b)

Mary wants to see $\left\{\begin{array}{c}\text{if}\\ \\ \text{that}\end{array}\right\}$ Tony bakes a cake.

We hypothesize that the choice of subordinating conjunction *of* (*if*) instead of *dat* (*that*) has resulted from a functorization rule which is sensitive to the Q tag.

We now proceed to wh-questions which come into play when S is given a conceptual argument of the following shape:

?X<...X...>.

For example, (37a) is an informal notation for the meaning that underlies (37b): for which X is it the case that X bakes a cake?

(37a) ?X<bake (actor: X) (product: a cake)>
(37b) Wie bakt een cake?
Who bakes a cake?

The discovery of an ?X tag attached to the conceptual structure it is trying to express, causes procedure S to engage in several special actions. First of all, S deposits a question mark within the P1 slot of the holder it created. (Remember the Q tag engenders the same effect.) This measure secures subject-verb inversion in main wh-clauses. The second action prepares for wh-movement. We assume that *every* S declares a variable whdest ("special destination for wh-constituents"). The default initialization value for whdest is NIL, but in the presence of the ?X tag it receives the same value as s-var, so that s-var and whdest both refer to the name of an S-instantiation. A conceptual structure tagged with ?X contains a gap where some piece of information is missing, at the location marked by X. The lexicalization process responds to a gap by activating a lemma whose only procedure call is one to lexical procedure Wh. We leave aside the details of how Appointment and Functorization Rules can fit Wh into the prevailing syntactic context. For instance, if NP hits upon the gap, then a functorization rule may reveal that the interrogative pronoun *who* is appropriate, and Wh will be enveloped in NPHead by an Appointment Rule:

NPHead(nil, <Wh(nil, <Lex(who)>)>).

Alternatively, X may be localized within the "specifier" region of a conceptual structure given to NP. Then, NP will accommodate Wh under Det:

Det(nil, < Wh(nil, < Lex(which) >) >)

Wh-fronting is accomplished through Destination Rule (38a) in conjunction with Word Order Rule (38b), which are to be added to (22) and Figure 3, respectively.

(38a) Destination Rule for FPROCs whose standard destination (as specified in (22)) is s-var:

Select whdest as destination if the following four conditions are fulfilled:

(1) the current value of whdest is not NIL (but the name of an S-instantiation);
(2) FPROC's own output value is not a clause (but an AP, NP or PP);
(3) FROC's own output value is a wh-constituent (i.e., contains a subtree delivered by procedure Wh);
(4) position of P1 of whdest is empty (no other constituent has yet been deposited there).

Otherwise, apply FPROC's standard Destination Rule.

(38b) Word Order Rule for FPROCs who have selected "whdest" as their destination:

Deposit the output value into slot P1 of the holder created by the S-instantiation whdest is referring to.

Wh-constituents will be fronted only in the presence of the ?X tag. Without this tag, the standard Word Order Rules of Figure 3 are applied, as in (39).

(39) Tonnie bakt wat?
Tony bakes what?
bake(actor: Tony)(product: X)

The construction of sentences (37b) and (40) is illustrated in Figures 10 and 11.

(40) Welke cake bakt Tonnie?
Which cake does Tony bake?

Variable whdest may look superfluous because its value is either NIL or identical to that of s-var: so why should FPROCs use whdest instead of s-var for destination? The answer has to do with embedded clauses.

In the context of certain verbs, a wh-constituent is allowed to escape from the confines of the S-instantiation it was constructed by. A case in point is (41), where the interrogative pronoun *wat* has moved out of *zien*'s object complement clause and occupied initial position in the main clause, thus also leaving *willen*'s complement behind.

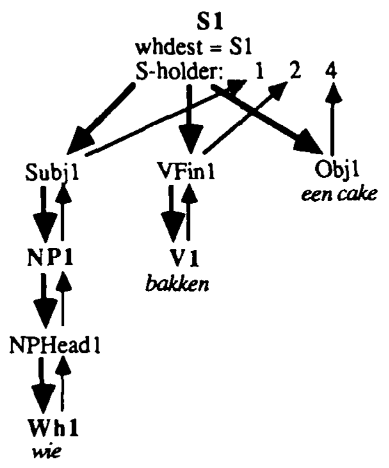


Figure 10. Construction of sentence (37b). Subj selects P1 of whdest through rules (38a–b).

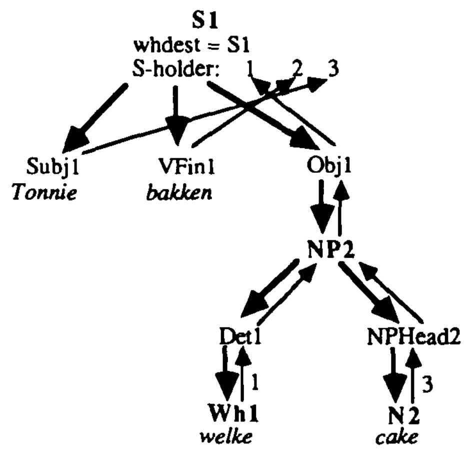


Figure 11. Construction of sentence (40). Obj selects P1 of whdest through rules (38a–b) and pushes Subj into P3 of S-holder.

```
(41) Wat will Marietje dat Tonnie ziet dat zij bakt?
     (What wants Mary that Tony sees that she bakes)
     What does Mary want Tony to see that she bakes?
     ?X< want(actor: Mary)
            (object: (see(actor: Tony)
                        (object: (bake(actor: Mary)
                                       (product: X)))))>
     For which X is it true that Mary wants Tony to see that she bakes X?
```

In fact, both *willen* and *zien* belong to the fairly small class of complement taking verbs which, under certain conditions, permit wh-movement across clauses. The conditions include, among other things, that the wh-constituent was constructed within the complement clause, not within a relative clause belonging to subject or object NP, for example.

Our treatment of cross-clause wh-movement involves a new synspec whose action overrules the standard initialization of whdest. Rather than being set to NIL or to the current value of s-var, whdest is given the same value as the other whdest that is within reach (i.e., is hit upon by climbing the procedure call hierarchy). The effect of this synspec function—let us call it *Copywhdest* —is exemplified by Figure 12 and (41). The lexical entries for *willen* and *zien* have to be adapted in such a way that Copywhdest will turn up on the synspec list of the correct S-instantiation. But we will skip these bookkeeping details.

An object complement verb which does not allow of cross-clause wh-fronting is *weten* (*know*). See (42a) and the snapshot of the computational process in Figure 13.

(42a) *Wie weet Tonnie dat een cake bakt?
(Who knows Tony that a cake bakes)
Who does Tony know that bakes a cake?

(42b) Tonnie weet dat wie een cake bakt?
Tonnie knows that who bakes a cake?
?X< know(actor: Tony)
(object: (bake(actor: X)
(product: cake)))>
For which X is it the case that Tony knows that X bakes a cake?

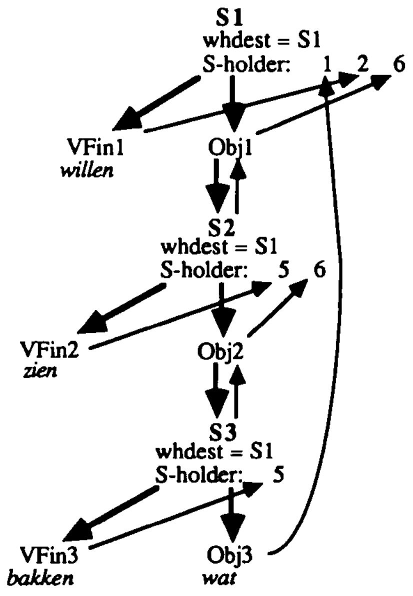


Figure 12. Partial construction of sentence (41).

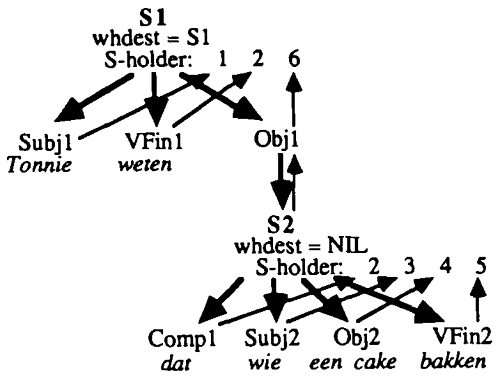


Figure 13. Construction of sentence (42a).

We assume the *weten*-lemma does not mention Copywhdest as a synspec function to be applied by Obj. This makes (42b) the grammatical version of (42a). *Wie* (*who*) is occupying its normal subject position, that is, P3, because it found the first condition in rule (38a) violated: whdest = NIL.

The machinery developed so far works for direct as well as indirect wh-questions. Sentence (43) is generated when the ?X tag is discovered by the embedded object clause. *Wie* goes to P1 of the holder created by the embedded S2. See Figure 14.

(43) Tonnie weet wie een cake bakt.
Tony sees who bakes a cake.
Know(actor: Tony)
(object: ?X< bake(actor: X)
(product: cake)>)
Tony knows for which X it is true that X bakes a cake.

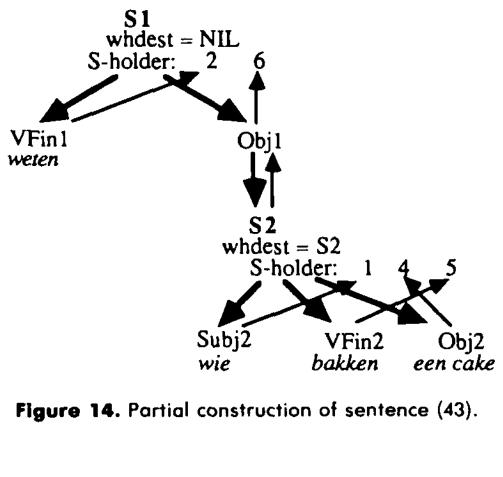


Figure 14. Partial construction of sentence (43).

(44) Tonnie ziet wie een cake bakt.
Tony sees who bakes a cake.

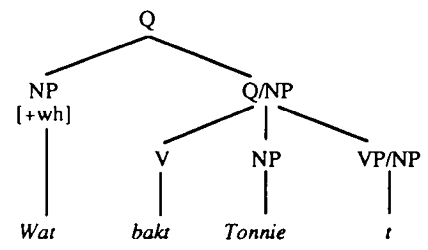
Example (44), however, proves that Copywhdest contains a bug. Since *zien* is one of the verbs injecting Copywhdest into its complement clause, the interrogative pronoun *wie* finds the topmost rather than the embedded S-instantiation as the value of whdest. The fact that *wie* is left within the embedded S2 implies that Copywhdest remains inactive when the complement clause is a wh-question. The following refinement of Copywhdest's operation will do the job:

(45) *Copywhdest:*
Copy into whdest the value of the first whdest up the procedure call hierarchy, unless the cp-argument of the current S-instantiation (i.e., the object clause) is tagged with ?X.

A point of similarity between our IPG analysis of wh-movement and Gazdar's (1981) treatment within the framework of Generalized Phrase Structure Grammar (GPSG) should not go unnoticed. Gazdar assigns interrogatives such as (46) a structure like (47).

(46) Wat bakt Tonnie?
What does Tony bake?

(47)



The derived categories Q/NP and VP/NP serve to remember, in a sense, that near the root of the tree an NP has been generated (the one marked " + wh") which later on must be left out (the empty constituent "t"). This resembles the IPG mechanism by which, through the values of whdest variables, possible attachment points for wh-constituents are carried down a procedure call hierarchy. One might wonder why IPG does not follow the GPSG solution to the end, that is, generate the wh-constituent immediately at the node where it will be attached rather than moving it there from a lower position. A proposal along these lines would be doomed to failure, though, being in conflict with the basic principle of lexically guided tree formation which we postulated in Section 1. This is so because the final place of a wh-constituent depends on properties of other lemmata figuring in the procedure call hierarchy. Remember, for example, the difference between *zien* and *weten* with respect to cross-clause extraction of wh-constituents. It is for this reason that we opt for a "movement" solution which is reminiscent of certain versions of Transformational Grammar.

4.3 Coordinate Structures

In this Section we outline our treatment of Coordination and two related phenomena: Conjunction Reduction and Gapping. (For a more detailed account of Coordination in Dutch, also including Right Node Raising, see Pijls & Kempen, 1986.) We start with some assumptions about the shape of conceptual structures underlying coordinate structures.

At the conceptual level, we assume, logical conjunction is expressed by the presence of AND, OR, BUT, and so forth, inbetween "conjuncts" (i.e., conjoined concepts or conceptual structures). In some of the examples in (48) we use square brackets to highlight conjoined structures.

(48a) bake (actor: Tony AND Mary AND Peter)(product: cake)
(48b) Tonnie, Marietje en Peter bakken een cake.
Tony, Mary and Peter bake a cake.
(48c) [bake (actor: Tony)(product: cake)] AND
[bake (actor: Mary)(product: tart)]
(48d) Tonnie bakt een cake en Marietje bakt een taart.
(48e) [bake (actor: Tony)(product: cake)] AND
[" (actor: Mary)(product: tart)]
(48f) Tonnie bakt een cake en Marietje een taart. (Gapping)
Tony bakes a cake and Mary a tart.
(48g) [bake (actor: Tony)(product: cake)] AND
[sell (actor: ") (possession: ")]
(48h) Tonnie bakt een cake en verkoopt hem. (Conjunction Reduction)
Tony bakes a cake and sells it.
(48i) ...dat Tonnie een cake bakt en verkoopt. (idem)
...that Tony bakes a cake and sells it.

In the context of Gapping and Conjunction Reduction we introduce a special "quotation mark convention." Many concepts mentioned in conjoined structures are repetitions of a concept which already figured in an earlier conjunct. For example, in (48c) there are two tokens of the element "bake." Following a well-known typographical convention, we will not spell out second and subsequent tokens but use double quotes instead, as in (48e) and (48g). Intuitively, quoted concepts are meant by the speaker to be less prominent (salient, foregrounded) than the ones that are spelled out in full.

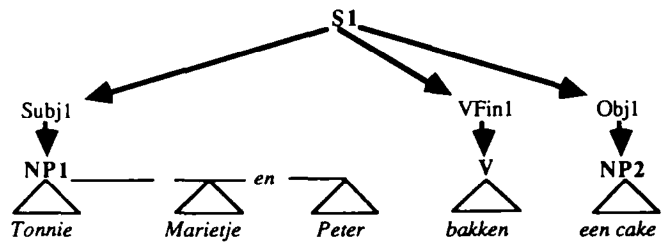
What happens to a syntactic procedure when its cp argument is a conjunction of two or more conceptual structures? The basic idea behind the

IPG approach to coordination is that of *iteration*. At the end of Section 3.5 we summarized the activities of syntactic procedures in schedule (23). The sequence of steps listed therein is repeated here as (49). However, we have added provisions for dealing with conjoined conceptual structures as cp value. In particular, note that step D attempts to lexicalize the various conjuncts of a cp one by one, and that step K instructs the procedure to resume step D as long as any conjuncts are waiting to be lexicalized. Thus an iterative loop is created spanning steps D through K. For each conjunct, the loop is traversed exactly once.

(49) A. Declare and initialize variables.
B. Create a holder.
C. Evaluate cp and synspec arguments.
D. Lexicalize (the next conjunct of) cp.
E. Apply Functorization Rules.
F. Apply Appointment Rules.
G. Run subprocedures in parallel.
H. Apply Word Order Rules to received subtrees.
I. Apply Destination Rules.
J. Return holder with contents to destination.
K. Exit if cp has been lexicalized exhaustively; otherwise go to D.

In (50a-b) we depict the construction of sentences (48b) and (48d). The horizontal dashed lines connect syntactic (sub)trees construed during successive traversals of the loop. Notice that the lemma which corresponds to the logical conjunction (*en*, and the comma between *Tonnie* en *Marietje*) are not supposed to make part of the syntactic tree proper.[7]

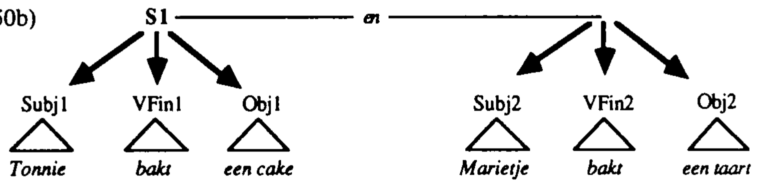
(50a)



[7] Notice furthermore that tree diagrams like (50a-b) were drawn on the assumption that only *categorial* procedures engage in the iterative coordination loop. This assumption is somewhat ad hoc because there is no technical reason such loops could not be located within FPROCs. There is an empirical advantage, though. Limiting iteration to CPROCs precludes coordination of phrases which belong to different families. For example, it will no longer be possible to generate a clause whose object consists of an NP coordinated with an S:

* Tony saw [Obj [NP Mary] and [S that she baked a cake]].

(50b)



Multiple traversals of the iterative loop create an access problem to destination holders. For example, after Subj1 has deposited its output "Tonnie" into position P1 of S-holder, this position is no longer available to Subj2. There are various possibilities of avoiding such conflicts. We have chosen to redefine holders as two-dimensional arrays with as many columns as there were positions in the original one-dimensional holders, and a sufficiently large number of rows. During the *n*th iteration of the loop, syntactic procedures deposit their values in the *n*th row of the destination holder. For instance, (50c) shows the contents of S-holder after completion of the FPROCs in (50b).

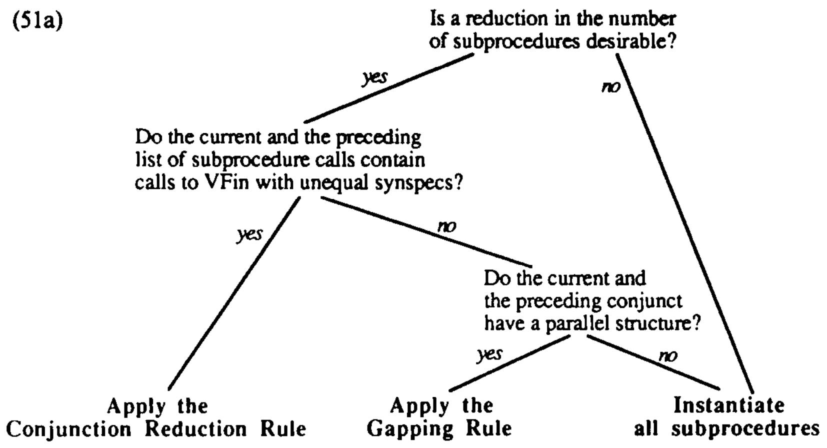
(50c)

	P1	P2	P3	P4	P5	P6	
1	Subj1 *Tonnie*	VFin1 *bakt*		Obj1 *een cake*			*en*
2	Subj2 *Marietje*	VFin2 *bakt*		Obj2 *een taart*			
3							

We propose to treat Gapping and Conjunction Reduction as a complication of step G in schedule (49). The standard action taken at that point is for a procedure to activate *all* subprocedures in the list that was compiled during previous steps. However, instantiations of a procedure are sometimes free to run only a subset of the list. This applies exclusively to second or subsequent traversals of the iterative loop, and only if the current conjunct and the current list of subprocedure calls that has been compiled show certain similarities to the preceding conjunct/list. Then, only such subprocedures are discarded which would have delivered the same output value as during the preceding iteration. Not actually instantiating and running them helps reduce the computational load of the syntactic processor. Nevertheless, we assume that the decision to consider discarding some of the subprocedure calls is optional. That is, instantiating and running the complete list of subprocedure calls will not violate grammaticality (although it does often lead to awkward sentences).

The algorithm for selecting subprocedures for actual instantiation is summarized in flow diagram (51a) and the associated rules (51b–c).

(51a)


Is a reduction in the number of subprocedures desirable?
yes
no
Do the current and the preceding list of subprocedure calls contain calls to VFin with unequal synspecs?
yes
no
Do the current and the preceding conjunct have a parallel structure?
yes
no
Apply the Conjunction Reduction Rule
Apply the Gapping Rule
Instantiate all subprocedures

(51b) *Gapping Rule.*

(1) For subprocedure calls whose cp argument is non-NIL: instantiate them if their cp argument is not quoted (but "prominent").

(2) For subprocedure calls whose cp argument is NIL: instantiate them if their synspec mentions a call to Lex(lm), where lm is a lemma which resulted from lexicalization of a nonquoted ("prominent") cp.

(3) The output values of the subprocedure which are actually instantiated and run must be ordered as the output values of their counterparts in the preceding iteration.

(51c) *Conjunction Reduction Rule.*

(1) Line the current list of subprocedure calls up with the contents of S-holder which were deposited during the preceding iteration. (The new call to Subj is paired with the value of the preceding Subj, the new VFin with the old VFin value, etc. In case of multiple SMods, also take into account the Path specified in the cp argument.)

(2) Instantiate all subprocedure calls starting from the first one having a cp or synspec argument which is "prominent" in the sense of steps (1) and (2) of (51b).

In flow diagram (51a) there is mention of "parallel structure" of two conjuncts. We will say that two conjuncts have a parallel structure if they can be lexicalized by the same lemma and, in case supplementary SMods are needed to cover all meaning aspects, if these SMods receive the same path functions. As to the VFin calls mentioned in the second node of the diagram, their synspecs always reference the Lex function with an auxiliary or a main verb as its argument. We will consider two such synspecs to be equal if the same V or Aux is involved in the same tense. We are now ready to discuss some examples.

Sentence (48f), a case of gapping, is generated from conceptualization (48e) through the following sequence of decisions:

A. Decision tree.
reduction of list of subprocedure calls is attempted;
two identical VFins: VFin(nil, <Lex(bakken)>);
conceptualization consists of two parallel conjuncts;
B. Gapping Rule.
Subj and Obj having "prominent" cp arguments will be instantiated;
VFin has a nonprominent synspec argument, so it must be discarded;
Subj and Obj values are ordered as in previous iteration.

The two cases of Conjunction Reduction in (48h–i) are constructed as follows:

A. Decision tree.
reduction of list of subprocedure calls is attempted;
unequal VFins (different lemmas);
B. Conjunction Reduction Rule.
Subj having a nonprominent cp argument is discarded;
VFin having a "prominent" synspec is instantiated;
in main clause (48h): Obj being located at the right-hand side of VFin is instantiated and delivers the accusative personal pronoun *hem*;
in subordinate clause (48i): Obj being located at the left-hand side of VFin and having a "nonprominent" cp argument must be discarded.

The ungrammaticality of (52b) is accounted for in terms of failing parallelism.

(52a) [bake (actor: Tony) (product: cake)] AND
[" (actor: ") (product: ") (time: tomorrow)]
(52b) *Tonnie bakt een cake en morgen.
*Tony bakes a cake and tomorrow.
(52c) Tonnie bakt een cake en hij bakt hem morgen.
Tony bakes a cake and he bakes it tommorow.

Since the conceptual conjuncts are not exactly parallel, the rightmost bottom node of the Decision tree is selected, that is, to run the complete list of subprocedures:

reduction attempted;
VFins equal;
no parallelism.

The resulting sentence is (52c). Leaving out any of the constituents *hij, bakt* and *hem* indeed makes the sentence ungrammatical or changes the interpretation.

The algorithm presented here cannot only handle most cases of coordination (for details see Pijls & Kempen, 1986) but it also accounts for the shape of self-corrections made during spontaneous speech. This will be the topic of Section 5.

4.4 Conditions on Transformations

One of the main objectives of transformational-linguistic research has always been "to narrow down the 'variation space' of human languages," as Koster (1978, p. 1) puts it. One attempts to define a set of possible grammars which is as small as possible and, within this set, to impose further restrictions, for example, to limit down the domain where free rule application is permitted, or to filter out certain structures produced by rules. The restrictions are typically formulated in terms of *configurations* of symbols occurring on the nodes of phrase-structure trees. An oft-used tool in defining such configurations is the relation of "command" between nodes. One version thereof, c-command, is defined as follows (Koster, 1978, p. 65): "A node A c-commands a node C, if the first branching node dominating A, dominates C, and A does not dominate C." Well-known examples of restrictions defined in configurational terms are the A-over-A principle, Subjacency, and various Island Constraints.

Koster (1978) has made a comparative study of about ten such restrictions proposed in the literature and convincingly argued that they are reducible to two very general principles. One of them he terms the Locality Principle. The notion of c-command is the main ingredient in the definition. Informally (and simplified), if a node A c-commands another node B, and B simultaneously c-commands a third node C, whereas A, B, and C are all of the same category (e.g., NP), then there is no syntactic rule which involves A and C. For example, there is no movement transformation which has A as target and C as Source; or, no rule which assigns an anaphoric relationship between A (antecedent) and C (consequent). In other words, no rule is allowed to "skip" middle term B; only A and B, or B and C, may be linked in a rule. Typical examples are (53) and (54).

```
(53)  * John  says   Mary  tried   e  to  like  himself.
        [S NP1  V   [S  NP2   V [S NP3          NP4]]]

(54)  *  What do you know      who said   that Peter saw e ?
        [S Wh1                [S Wh2      [S               Wh3]]]
```

The symbol "e" represents an empty node which is to be linked with one of the other nodes. (53) is ungrammatical because a link is attempted between NP3 and NP1, thus violating the Locality Principle because the middle term of a c-command triplet is skipped. Only *Mary* is allowed to be the subject of *like*, and *himself* should change to *herself. (54)* violates the principle because Wh2 prohibits a link between the empty Wh3 and Wh1.

In IPG both examples are easy to handle. Their ungrammaticality follows from the fact that they attempt to access a variable which is out of scope, that is, unreachable by means of the search rule: "Go up the tree and halt at the first occurrence of the varable's name." As for (54), the only whdest variable which is visible from the point of view of Obj2—the Obj instantiation running in the deepest S—is the one declared by the middle S. Obj2 simply doesn't see the topmost whdest and cannot send *what* off to S1, the S-instantiation this whdest is pointing to.

Case (53) presupposes a way of handling reflexive pronouns. Let us first sketch out how this could be done. The problem here is for one procedure (e.g., Obj) to get to know the conceptual content that is being expressed by another one (e.g, Subj) which is running in parallel. As a rule, procedures other than Subj (e.g., Obj, IObj, SMod) use a reflexive pronoun if their own content is coreferential with the "subject content." Because reflexivization is generally admissible within clauses, we propose to let every S declare a variable *subjcontent* whose value is the conceptual content associated with Subj in the V-lemma S has looked up. Figure 15 shows the resulting configuration of variables and their values. Again, the deepest Obj was no way of knowing that there is a subjcontent pointing to "John."

What has struck us in these and other constructions discussed by Koster, is that the restrictions on rule applicability that follow from the Locality Principle often come remarkably close to those imposed by the limited range of vision that syntactic procedures have.

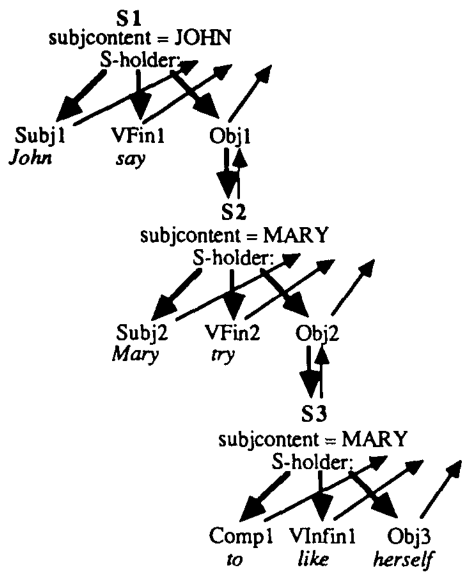


Figure 15. Construction of the grammatical variant of (53). The lowest instantiation of Obj selects *herself* rather than *himself* because of referential identity of its cp with (the lowest occurrence of) subjcontent. We have assumed that *say* and *try* make use of synspec function (copywhdest).

The second principle proposed by Koster is the Bounding Condition. In essence, it states that all major phrase types—S, NP, PP, and AP—are islands from which no elements can escape. The Bounding Condition is supposed to belong to the *core* grammar of a language: It can only be violated by noncore or peripheral rules (''markedness''). There are independent criteria, of course, for assigning a rule to the core or to the periphery of the grammar. For example, wh-movement across clauses is a peripheral phenomenon since it is only permitted in a relatively small group of languages and, even there, often restricted to special lexical entries. The Bounding Condition is needed to prevent cases such as (55) and (56) where prepositional phrases have been moved out of an NP and an AP, respectively (Koster, p. 72 and 82).

(55) *About what did Einstein attack a theory?

(56) *By Nixon, all books written are sold out. (Instead of ''...written by Nixon...'')

Again we observe that IPG embodies the principle without having to state it explicitly. The four major phrase types correspond to the four families of syntactic procedures in (6b). FPROCs always seek as their destination an instantiation of the phrasal procedure of the family they belong to. Subj seeks s-var; any NMod constituent goes to np-var, and so forth. It is clearly this convention which causes the ''island'' character of major phrases. We have seen that special gadgets are needed to break through phrase boundaries (e.g, whdest in Section 4.2). Examples (55) and (56) are ungrammatical because they presuppose NMod and AMod to have sent off their values to s-var rather than to np-var and ap-var, respectively.

The conclusion that IPG globally behaves in accordance with the two principles proposed by Koster, raises the question why these principles seem to follow from the inner workings of this grammar and need not be added as supplementary restrictions. The design feature from which the unexpected but desirable behavior originates is the ''stack'' which IPG operates. In our case the stack has a tree-like structure isomorphic with the procedure call hierarchy. The stack is a repository for all information a procedure wishes to share with other procedures. Among other things, it contains declared variables with their values, and instructions signaling which procedures are working for which other procedures. Every new piece of information is put on top of all old information, and searching the stack proceeds from top to bottom. (This corresponds with bottom-up search through the trees.) This way of setting up the stack and using it makes possible a pattern of communication between procedures which

(a) shows a strictly hierarchical organization,
(b) has facilities for recursion, and
(c) is subject to limitations of scope.

These properties are also exhibited by ALGOL-like programming languages using a stack in their implementation. It is highly remarkable, though, that these properties also characterize the design of grammars for natural languages. The features of hierarchy and recursion are usually embodied in phrase-structure rules which in some form make part of any serious grammar type. (Within IPG they can be discerned in the Appointment and Functorization Rules.) And, as we have argued earlier in this Section, the scope limitations correspond closely to the configurational conditions on transformational and other linguistic rules.

The idea that constraints on grammar rules arise from the use of a stack is not new. Marcus (1980), for example, has proposed such an account for the Complex NP constraint—one of the rules covered by Koster's Locality and Bounding principles. What we think *is* new in the present account is an answer to the deeper question why constraints of a *configurational* nature exist at all. IPG interprets syntax trees as a computational environment inhabited by active units which have specialized and limited syntactic knowledge, operate in parallel, and are relatively autonomous. There is no central construction agency which at all times has a complete overview of what it has built so far and is in full command of what it will do next. Each specialist (syntactic procedure) is aware of only a part of the total computational environment—the segment which it can access through given communication channels. The configurational constraints mirror these channels (in this case: the stack). Beyond that, no contact between procedures in the computational environment (no interaction between nodes of the syntax tree) is possible. Within a sentence construction mechanism controlled by a single processing unit (as is typically assumed in "direct realization models" of Transformational Grammar), constraints of a nonconfigurational nature are quite conceivable, such as constraints referring to the identity or the number of nodes. Imaginary examples are "A transformational rule never relates two or more NP nodes" or "No transformation may lead to a node with more than 4 sister nodes." The configurational constraints on the other hand are a natural consequence of the basic assumption of the limited scope of syntactic procedures or modules. Constraints of a different nature are impossible or, at best, can only be achieved at high computational cost. This is why we venture the claim that Incremental Procedural Grammar is potentially superior to Transformational Grammar in terms of explanatory adequacy. Potentially—because many syntactic constructions are still awaiting treatment in IPG terms.[8]

5. INCREMENTAL SENTENCE PRODUCTION

In the remaining Sections we will concentrate on psychological issues. First of all, how can the formulator build sentences which dovetail into the evolving conceptual structures delivered by the conceptualizer?

[8] For a more formal approach to the issues raised in this Section, see Hoenkamp (1983).

Let us assume the conceptualizer delivers the conceptual structure for a sentence as a cumulative sequence of *expansions* e(1), e(2),. . . e(n). Each expansion e(i) is a proper subset of its successor e(i + 1) in the sense that e(i + 1) contains concepts and/or conceptual relations which were not yet present in e(i).

The computational principle we employ for dealing with incremental sentence production is, again, *iteration.* Into every syntactic procedure we build an iterative loop spanning steps G through K, very much like in our treatment of coordination. (See Section 4.3. In Section 5.2 we shall explain that the "incrementation loop" is nested within the "coordination loop.") During each new iteration of the loop, the next expansion in the sequence is processed. Special measures should prevent such iterations from merely leading to a succession of disconnected utterances, the last of which is the final sentence, as in (57a). Instead, one integrated utterance should result which is syntactically coherent as a whole (57b).

(57a) Gister . . .
Tonnie bakte gister . . .
Gister bakte Tonnie een cake.
(Yesterday baked Tony a cake)

(57b) Gister . . . bakte Tonnie . . . een cake.
Yesterday Tony baked a cake.

(57c) Tonnie bakte . . . gister . . . een cake.
Tony baked a cake yesterday.

(58) De cake .. is door Tonnie gebakken . . . gister.
(The cake has-been by Tony baked yesterday)

Examples (57b–c) and (58), moreover, demonstrate that the syntactic shape of the integrated utterance is dependent on the order in which the various parts of the conceptual structure are expanded, that is, from their conceptualization order. We have assumed a "first in, first out" schedule which—within limits of grammaticality—attempts to assign to new parts of the utterance a position as much to the left as is possible. The Word Order Rules of Figure 3 reflect this schedule. For example, if SMod outwins all other FPROCs running under S, it will occupy the leftmost position P1. This has happened in (57b). (*Gister* is an adverb which via application of Appointment Rules is allotted the role of SMod.) The point of (58) is that the choice between active and passive voice may be determined by order of conceptualization.

We will clarify the incrementation loop in terms of sentence (58). Let us imagine the conceptualizer delivers the meaning underlying this sentence as a sequence of four expansions:

e(1) the cake
e(2) the cake, Tony
e(3) Tony having baked the cake
e(4) Tony having baked the cake yesterday.

The four columns of Figure 16 show how the hierarchy of procedure calls grows in response to the meaning expansions. Procedure S1 goes through four iterations; the corresponding lists of subprocedures calls are given in Table 1.

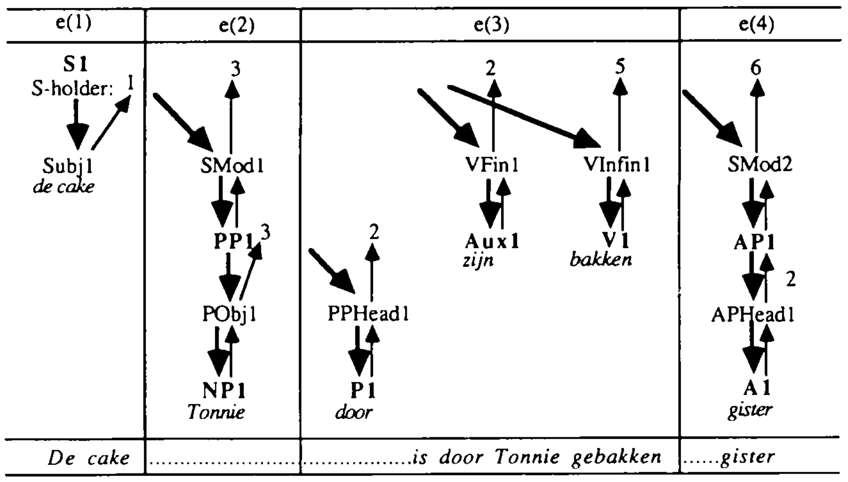


Figure 16. Construction of sentence (58) out of four conceptual fragments. Downward (procedure call arrows descending from S have been omitted. The auxiliary *is* (a form of *zijn*) replaces *worden* because of perfect tense (cf. lemma (60)).

TABLE 1
Lists of subprocedure calls composed during the incremental production of sentence (58). Cp1, cp2 and cp3 refer to the meanings underlying *cake*, *Tony* and *yesterday*, respectively. Arrow] indicates which procedures are actually run. See also Figure 16.

e(1)	old:	---
	new:	—Subj(cp1, < >)
e(2)	old:	Subj(cp1, < >)
	new:	—SMod(cp2, < >)
e(3)	after first lexicalization attempt:	
	old:	*Subj(Path(actor. . .), < >)
	new:	VFin(nil, <V(nil, <Lex(bakken)>)>)
		Obj(Path(product. . .), < >)
	after second lexicalization attempt:	
	old:	Subj(path(product. . .), < >)
		—*SMod(Path(actor. . .), <P(nil, <Lex(door)>)>)
	new:	—VFin(nil, <Aux(nil, <Lex(worden)>)>)
		—VInfin(nil, <Lex(gebakken)>)>)
e(4)	old:	see the list after second lexicalization of e(3)
	new:	—SMod(cp3, < >)

Iteration 1. After having lexicalized and applied Appointment Rules to noun lemma *cake,* S1 assigns it the role of syntactic subject. Subj deposits its value into slot P1 of S-holder and exits. The contents of P1 are passed down to the Morpho-Phonological Stage and pronounced as *de cake.*

Iteration 2. There is no reason for the lexicalization process within S1 to reconsider its decision to select the noun *cake.* It deals with the meaning increment simply by handing it over to SMod. Within SMod, Appointment Rules force the noun lemma *Tonnie* into the role of prepositional object, with the preposition left undecided yet. The new contents of S-holder's P3 slot cannot be processed by the Morpho-Phonological Stage for reasons to be explained below.

Iteration 3. Lexicalization within S1 during its third iteration yields the active verb lemma *bakken.* However, the path function associated with the Subj call in this lemma evaluates to Tony, that is, the content of the actor region of e(3), and is not coreferential with cp1 (see Table 1). This implies the lemma cannot lead to a proper continuation of the fragmentary sentence uttered so far. What is needed is a facility for examining whether a new iteration will lead to a continuation which is in keeping with the syntactic commitments made during earlier iterations. To this purpose, we introduce a *Compatibility Check* which is carried out as follows. For each subprocedure call in the list composed during the preceding iteration it is determined whether there is a compatible counterpart in the current list of subprocedure calls. The notion of compatibility is defined in (59).

(59) Procedure call PROC2(cp2, <synspec2>) is compatible with procedure call PROC1(cp1, <synspec1>) if
  (a) PROC2 and PROC1 are identical procedure names, and
  (b) cp2 and synspec2 are identical to or expansions of cp1 and synspec1, respectively.
  (The term expansion was defined above; here we add that anything non-NIL is considered an expansion of NIL.)

The Compatibility Check discloses that the new Subj call is incompatible with the call to Subj in iteration 2. (This is indicated in Table 1 by an asterisk.) Another problem concerns the SMod call which has no counterpart in the current list.

A second consultation of the lexicon yields the passive *bakken* lemma which is less incompatible: see (60).

```
(60) VInfin(nil, <V(nil, <Lex(gebakken)>)>)
     Aux(nil, <Lex(worden)>)
     Subj(Path(...), <>)
     SMod(Path(...), <P(nil, <Lex(door)>)>)
```

The path functions associated with Subj and SMod single out the product and the actor of the baking event, respectively. These happen to coincide with the contents expressed by Subj and SMod during the first two iterations. The synspec arguments presenting no compatibility problems, lemma (60) is accepted.

Iteration 4. S1 adds a second call to SMod with the new temporal information as to-be-expressed meaning content. This SMod retrieves the adverb *gister* (Eng. *yesterday*) and attempts to deposit its output value into P3 of S-holder—in accordance with the word order rules of Figure 3. This presents a little problem. After e(3), the Morpho-Phonological Stage has got as far as position P5 of S-holder: the VInfin participle *gebakken* has already been pronounced. Rather than dropping the adverb at P3, SMod now selects P6—a possibility having low priority and not mentioned in Figure 3. Position P6 is still open, that is, no output values deposited there have yet been processed by the Morpho-Phonological Stage. We assume that syntactic procedures try to avoid incursions into positions within a holder which have already undergone morpho-phonological processing.[9]

We owe an explanation yet for the fact that the Morpho-Phonological Stage could not accomplish anything after the second iteration. A problem incremental sentence production runs into, is that the slots of holders are not getting filled in an orderly left-to-right fashion. Moreover, slots often remain empty during the construction of a sentence. The Morpho-Phonological Stage could, in theory, follow an extremely conservative strategy: waiting until the utterance is complete and then process it in one go. This solution is highly unsatisfactory, though, because it would obviate the need of generating sentences incrementally in the first place. The conservative strategy is in perfect harmony with traditional grammatical systems which only generate full sentences. However, the advantages of incremental production in terms of a more regular and fluent speech output would vanish completely. A maximally progressive strategy is out of the picture altogether. The Morpho-Phonological Stage would often jump too quickly to the right and skip slots intended for obligatory constituents. Figure 16 is a case in point. Since S1 is building a main clause, there will ultimately have to be a finite verb occupying P2. This means the Morpho-Phonological Stage should not be permitted to jump over P2 before having processed a VFin constituent there. Similarly, position P3 of an NP-holder can never be passed by without having processed an NPHead value.

What we need is a device for marking the slots which are going to be occupied by obligatory constituents. A fairly straightforward way of doing this is to launch obligatory procedures at the earliest possible moment, as soon as they are dictated by the syntactic constellation. In terms of the pres-

[9] Such a strategy eliminates at least some retracings. It is bought at a price, though: sentence (58) sounds rather colloquial and is only marginally acceptable.

ent example, we could insert a call to VFin already during the first iteration. Since no proper cp and synspec arguments are available at this point, we propose the convention that a procedure which is running with NIL arguments also delivers NIL as its output value and deposits it at the standard destination. This symbol, then, is interpreted by the Morpho-Phonological Stage as a halt signal. Later on, such a dummy obligatory procedure will be replaced by a new instantiation with adequate cp and synspec arguments. It computes a non-empty holder as its output value overwriting the NIL symbol. We wish to remark, however, that this solution is tentative and ad hoc. Speakers probably engage in much richer varieties of forward syntactic inferencing than is made possible by the strategy proposed here.

6. REPAIRS AND ELLIPSIS

A speaker who decides to repair part of the utterance he/she is pronouncing, often backtracks to the beginning of the last constituent, thus restoring the integrity of the interrupted syntactic unit (a classic reference is Maclay & Osgood, 1959). Recently, Levelt (1983) has observed that speakers obey a much stricter rule when deciding how far they should backtrack. Example (61) shows that going back to the beginning of a constituent (here, the prepositional object NP) is not a sufficient condition for a repair to be successful.

(61) *With his sister he talked frequently, uh, his mother he talked frequently.

Levelt proposes the following well-formedness rule for repairs (quoted here in a slightly simplified form):

(62) A repair <A,C> is well-formed if there is a string B such that the string <AB and C> is well-formed, where B is a completion of the constituent directly dominating the last element of A.

A and C designate the original utterance and the repair, respectively; editing expressions ("uh") are ignored. The rule predicts that (63a) is an ill-formed repair because (63b) is not a well-formed coordination.

(63a) *Did you see a green, uh, you see a blue circle?

(63b) $\underbrace{\text{Did you see a green}}_{A}\ \underbrace{\text{circle}}_{B}\ \text{and}\ \underbrace{\text{you see a blue circle?}}_{C}$

(64a) $\underbrace{\text{With his sister he talked frequently}}_{A}\quad \underset{B}{\text{and}}$

$\underbrace{\text{with his mother he talked frequently.}}_{C}$

(64b) With his sister he talked frequently, uh,
with his mother he talked frequently.

Since (64a) is a grammatical coordination, it follows that (64b) is a "grammatical" version of (61). Sentence (64a) also serves to illustrate a special case of Levelt's rule where part B is empty.

IPG accounts for the well-formedness rule in a very straightforward manner. It assigns the duty of carrying out self-corrections to the mechanism which is also responsible for computing the shape of coordinate structures. As a matter of fact, some of the assumptions we made in Section 4.3, particularly the idea of an iterative loop and the distinction between new/prominent versus quoted/nonprominent parts of a conceptual structure, were inspired by—and become intuitively more acceptable in—the situation of speakers correcting their own speech. The examples in (65) show that indeed repairs can be treated on a par with coordinate structures.

(65a) Tonnie bakt een lekkere cake ... eh ...een lekkere taart.
Tonny bakes a delicious cake ... uh ... a delicious tart.
(65b) [bake(actor: Tony)(product: cake(mod: delicious))] CORR
[" (actor: ")(product: tart (mod: "))]
(65c) Tonnie bakt een lekkere cake en een lekkere taart.
Tonny bakes a delicious cake and a delicious tart.
(65d) Tonnie bakt een lekkere ... eh ... een lekkere taart.
Tonny bakes a delicious ... uh ... a delicious tart.

The symbol CORR(ection) in conceptual structure (65b) signals a "change of mind": the conceptualizer replaces the left-hand side by the right-hand side. CORR lexicalizes into a correction phrase such as "uh" or into a pause. When detected by syntactic procedures, CORR causes them to start a new iteration in a way comparable to the logical conjunctions AND, OR, and so forth (see Section 4.3). In terms of the example, CORR induces within S a second iteration which leads to a new instantiation of Obj—the call to Obj being the only one having a (partly) prominent argument. Sentence (65c) will be composed in case AND is substituted for the correction symbol CORR. Utterance (65d) differs from (65a) in that the object constituent was not fully realized. We might assume that morpho-phonological or articulatory processing was interrupted just before the word *cake* could surface. Apparently there exist (conceptualizing, monitoring?) processes having the authority to interrupt ongoing speech production activity at any point in time (cf. Van Wijk & Kempen, 1987, for some supporting experimental evidence).

It is a well-known observation that the members of a self-correction (i.e., reparandum and repair) and of a coordination (i.e., the various conjuncts) obey certain structural constraints. See (52) for an example violating such constraints. In his 1983 paper, Levelt points out that there exists a third group of language production phenomena displaying a very similar type of structural transfer between members, namely, wh-questions and their (elliptical) answers. For instance, (66b) is alright as an answer to (66a) as long as the preposition *with* is not deleted.

(66a) With whom did he talk frequently?
(66b) With his mother he talked frequently.
(66c) With his mother.

Suppose we would have S express the meanings underlying (66a) and (66b) in two successive iterations. The utterance after the second iteration would be the elliptical answer (66c) because SMod is the only procedure that is assigned a prominent cp argument during the second iteration: the gap marked by dummy symbol X (cf. (37) in Section 4.2) has been filled. None of the other subprocedures within S need be instantiated anymore.

An important implication of our proposal to handle coordination, self-repairs as well as elliptical question-answering in terms of an iterative loop within syntactic procedures, is that the structural transfer must be very similar in the three cases. Moreover, no special mechanism is needed for realizing the structural transfer (except for the parallelism check in (51a), of course; see also footnote 7). In contrast to Levelt's proposal, it will not be necessary to invoke some external parsing process which analyses the structural properties of one member (conjunct, reparandum, wh-question) and transfers these to the other one (a second conjunct, the repair, the answer).

The last issue we wish to raise in this Section concerns the relationship between incremental production and self-correction. More specifically, how does the incrementation loop introduced in the preceding Section interact with the coordination/correction loop? Formula (67) gives the general form of the cp argument of syntactic procedures.

(67) [e(1,1), e(1,2), . . . , e(1,m)] C [e(2,1), e(2,2) . . . , e(2,n)] C . . .

The symbol C indicates a logical conjunction or CORR and separates the members of a coordination or a self-correction. A conceptual structure contains at least one member. Each member (between square brackets) consists of a cumulative sequence of one or more *expansions* as defined in the previous Section. Syntactic procedures go through the coordination/correction loop once for every new member; within a member, the incrementation loop is traversed once for every new expansion. Therefore, the incrementation loop is nested within the coordination/correction loop.

7. FURTHER PSYCHOLOGICAL ISSUES

The last Section of this paper is devoted to some miscellaneous issues that models of the speaker have tried to tackle.

Speech Errors. In Section 2 we summarized Garrett's (1975) observations on two classes of speech errors: word exchanges and combined-form exchanges ("stranding errors"). Exchanged words are very often members of the same part of speech, fulfill similar syntactic functions in the sentence, and may be far apart in the utterance. None of these regularities hold for stranding errors. We propose to view a word exchange as a *lemma* exchange between two syntactic procedures which are running in parallel and consult the lexicon at approximately the same time. Such procedures typically serve similar syntactic functions (e.g., Subj and Obj within S) and use lexical material of the same grammatical category, but their values may end up at remote positions in the sentence. Stranding errors, on the other hand, involve an exchange between *lexemes* during the Morpho-Phonological Stage. The rules of inflection, however, are executed in the normal way as if no interchange had taken place. Only lexemes at nearby positions in the utterance can get involved in such interchanges because they are looked up in close temporal succession (Van Wijk & Kempen, 1987). Similarity between lexemes in terms of word class membership or syntactic function is uncorrelated with their distance in the utterance.

Notice that a lemma exchange will indirectly cause an exchange of dependent function words as well. For instance, suppose a speaker has interchanged two noun lemmas as in (68).

(68) Tonnie deed DE BAKVORM in HET DEEG.
Tony put the baking tin into the dough.

Since Functorization Rules, which are applied after lexicalization, are unaware of the exchange, they will insert the articles at the wrong places. (*Deeg* is a *het*-woord (neuter), *bakvorm* a *de*-woord.) Exchanged lexemes, on the other hand, cannot carry along dependent function elements simply because their dependence is not specified at the morpho-phonological level. There is more to Garrett's findings and analyses than we can mention here, but we believe that the essentials are within reach of IPG.

Clitic Omission in Agrammatism. Kean (1977, 1979) has observed that the most pervasive phenomenon in the speech of agrammatic patients is their tendency to leave out words and morphemes best characterized as belonging to the class of *clitics,* that is, inflections, articles, pronouns, auxiliaries, subordinating conjunctions, and small prepositions (especially monosyllabic). Of special importance is the fact that the class of clitics cuts across the class of prepositions. At the end of Section 3.4 we have already indicated that non-clitical prepositions are activated via lexicalization in direct response to conceptual input. Clitical prepositions, on the other hand, come into play as a result of the application of Functorization Rules, that is, primarily in response to the configuration of syntactic procedures and other aspects of the current computational environment.

Kolk, Van Grunsven, & Keyser (1985) argue convincingly that agrammatic speech is caused by a simplified conceptual input which is detailed enough to enable the patient to find the communicatively important content words (through lexicalization) but lacks information triggering the insertion of clitics. Thus, the patient obviates the necessity of maintaining the complex computational environment presupposed by correct application of Functorization Rules while minimizing communicative losses. IPG appears compatible both with Kean's observations and with Kolk's theory.

Speech Formulae. Fluency profits from the ease with which speakers avail themselves of all sorts of idiomatic expressions which may range over a fair number of words. It should be unproblematic for the formulator to look up and retrieve such speech formulae from the lexicon and to fit them into the grammatical structure it is working on. Lexical entries which correspond to idiomatic expressions spanning more than a single word, have no special status in IPG. For instance, (69) is the lemma for *een poets bakken* (*to hoax*).

```
(69) V(nil, <Lex(bakken)>)
     Subj(Path(...), < >)
     Obj(nil, <Art(nil, <Lex(een)>)
             N(nil, <Lex(poets)>)>)
     IObj(Path(...), < >)
```

The path functions are supposed to lead to the "joker" (subject) and the "victim" (indirect object). Notice that the shape of the object phrase is determined not by its cp argument, which is NIL, but by synspec (in conjunction with Appointment Rules which make a noun phrase out of it).

Formulating as Automatic Activity. The numerous syntactic computations which are carried out during language production hardly require conscious attention on the part of the speaker (Kempen, 1981; Bock, 1982). Whole sentences may "spring to mind" even if they have never been heard or used before. (Exceptions, of course, are speakers with insufficient mastery of the language.) This tallies with the idea, embodied in IPG, of sentence formulation by a *team* of syntactic experts rather than by a single processor. It also helps to understand that sometimes several formulations of the same conceptual structure seem to be developing simultaneously, as witnessed by "syntactic fusion errors." We observed a speaker of Dutch who produced a blend of the two synonymous sentences (70a) and (70b).

(70a) Alles moet morgen klaar zijn.
(Everything must tomorrow ready be)
Everything must be ready tomorrow.
(70b) Morgen moet alles klaar zijn
Tomorrow everything must be ready.

What the speaker said was "Argen...," a mixture of *alles* and *morgen,* after which he stopped immediately. Apparently, two constructions were being prepared in parallel, with different orders, both grammatical, of subject (*alles*) and adverbial modifier (*morgen*). The speaker's introspection right after he produced the speech error were in agreement with this analysis.

One Grammar for Perception and Production? The idea that one and the same grammar is utilized for both sentence production and sentence perception has always appealed to the minds of linguists and psycholinguists. Theoretical proposals for a grammar that can do both jobs are conspicuously absent from the psycholinguistic literature, though, and discussions of the attainability of such a grammar tend to end with discouraging conclusions (Fodor, Bever, & Garrett, 1974). Unificational Grammar (Kay, 1984) is the first linguistic formalism which is truly bidirectional. However, most of the psycholinguistically desirable features which we described in Section 1 (e.g., incremental generation) seem to be lacking.

Without claiming to have a workable plan, we wish to draw attention to the fact that, from the point of view of IPG, syntactic parsing (as part of the language perception process) is remarkably similar to syntactic formulating (as part of the language production process). (1) Parsing and formulating are both lexically driven, that is, operate on the basis of syntactic information stored with individual words of the lexicon. (2) Both processes use that information for the purpose of constructing a syntactic tree with these words as terminal elements. (3) They both have facilities for growing syntactic trees from left to right. The parser needs them for attaching new words to the current syntactic tree, the formulator for computing a continuation (incremental production). The origin of the words is different, of course: They stem from speech recognition in case of parsing, but from lexicalization in case of formulating. We hope that exploring these unexpected parallels will stimulate the study of both human language perception and language production, and bring us to the attractive situation of having one device which is a syntactic parser and a syntactic formulator at the same time.

REFERENCES

Bock, J.K. (1982). Toward a cognitive psychology of syntax: Information processing contributions to sentence formulation. *Psychological Review, 89,* 1–47.

Bresnan, J. (Ed.). (1982). *The mental representation of grammatical relations.* Cambridge, MA: MIT Press.

Bresnan, J., & Kaplan, R.M. (1982). Introduction: Grammars and mental representations of language. In J. Bresnan (Ed.), *The mental representation of grammatical relations.* Cambridge, MA: MIT Press.

Bresnan, J., Kaplan, R.M., Peters, S., & Zaenen, A. (1982). Cross-serial dependencies in Dutch. *Linguistic Inquiry, 13,* 613–635.

Cole, P., & Sadock, J. (Eds.). (1977). *Grammatical relations. Syntax and semantics.* Vol. 8. New York: Academic.

Dik, S.C. (1978). *Functional grammar.* Amsterdam: North-Holland.

Fodor, J., Bever, T., & Garrett, M. (1974). *The psychology of language.* New York: McGraw-Hill.

Fromkin, V. (1971). The non-anomalous nature of anomalous utterances. *Language, 47,* 27-52.

Garrett, M. (1975). The analysis of sentence production. In G. Bower (Ed.), *The psychology of learning and motivation, Vol. 9.* New York: Academic.

Garrett, M. (1980). Levels of processing in sentence production. In B. Butterworth (Ed.), *Language production* (Vol. 1 Speech and Talk). New York: Academic.

Gazdar, G. (1981). Unbounded dependencies and coordinate structure. *Linguistic Inquiry, 12,* 155-184.

Geurts, B. (1985). *Semantics for IPG.* Paper, University of Nijmegen.

Goldman, N. (1975). Conceptual generation. In R. Schank (Ed.), *Conceptual information processing.* Amsterdam: North-Holland.

Goldman-Eisler, F. (1968). *Psycholinguistics: Experiments in spontaneous speech.* New York: Academic.

Harley, T.A. (1984). A critique of top-down independent levels models of speech production: Evidence from non-plan-internal speech errors. *Cognitive Science, 8,* 191-219.

Hoenkamp, E. (1983). *Een computermodel van de spreker: psychologische en linguistische aspecten.* Doctoral dissertation, University of Nijmegen.

Kaplan, R., & Bresnan, J. (1982). Lexical-functional grammar: A formal system for grammatical representation. In J. Bresnan (Ed.), *The mental representation of grammatical relations.* Cambridge, MA: MIT Press.

Kay, M. (1984). Functional Unification Grammar: A formalism for machine translation. *Proceedings of COLING84.* Stanford.

Kean, M.-L. (1977). The linguistic interpretation of aphasic syndromes: Agrammatism in Broca's aphasia, an example. *Cognition, 5,* 9-46.

Kean, M.-L. (1979). Agrammatism: a phonological deficit? *Cognition, 7,* 69-83.

Kempen, G. (1977). Conceptualizing and formulating in sentence production. In S. Rosenberg (Ed.), *Sentence production: Developments in research and theory.* Hillsdale, NJ: Erlbaum.

Kempen, G. (1978). Sentence construction by a psychologically plausible formulator. In R.N. Campbell & P.T. Smith (Eds.), *Recent advances in the psychology of language. Formal and experimental approaches.* New York: Plenum.

Kempen, G. (1981). De architektuur van het spreken. *Tijdschrift voor Taal- en Tekstwetenschap, 1,* 110-123.

Kempen, G., & Hoenkamp, E. (1982). Incremental sentence generation: Implications for the structure of a syntactic processor. *Proceedings of the Ninth International Conference on Computational Linguistics.* Prague.

Kempen, G., & Huijbers, P. (1983). The lexicalization process in sentence production and naming: Indirect election of words. *Cognition, 14,* 185-209.

Kolk, H., van Grunsven, M., & Keyser, A. (1985). On parallelism between production and comprehension in agrammatism. In M.-L. Kean (Ed.), *Agrammatism.* New York: Academic.

Koster, J. (1978). *Locality principles in syntax.* Dordrecht: Foris.

Levelt, W. (1982). Linearization in describing spatial networks. In S. Peters & E. Saarinen (Eds.), *Processes, beliefs, and questions.* Dordrecht: Reidel.

Levelt, W. (1983). Monitoring and self-repair in speech. *Cognition, 14,* 41-104.

Maclay, H., & Osgood, C. (1959). Hesitation phenomena in spontaneous English speech. *Word, 15,* 19-44.

Mann, W. (1982). Text generation. *American Journal of Computational Linguistics, 8,* 62-69.

Marcus, M. (1980). *A theory of syntactic recognition for natural language.* Cambridge, MA: MIT Press.

McDonald, D. (1980). *Natural language production as a process of decision-making under constraint.* Doctoral dissertation, MIT, Cambridge, MA.

Miller, G., & Chomsky, N. (1963). Finitary models of language users. In R. Luce, R. Bush, & E. Galanter (Eds.), *Handbook of mathematical psychology*. New York: Wiley.

Pijls, F., & Kempen, G. (1986). *Een psycholinguistisch model voor syntactische samentrekking De Nieuwe Taalgids, 79,* 217-234. (English translation available as a report of the Department of Experimental Psychology, University of Nijmegen, The Netherlands.)

Schank, R.C. (1975). *Conceptual information processing*. Amsterdam: North-Holland.

Simmons, R., & Slocum, J. (1972). Generating English from semantic networks. *Communications of the ACM, 15,* 891-905.

Van Wijk, C., & Kempen, G. (1985). From sentence structure to intonation contour: An algorithm for computing intonation contours on the basis of sentence accents and syntactic structure. In B.S. Mueller (Ed.), *Sprachsynthese*. Hildesheim: Olms.

Van Wijk, C., & Kempen, G. (1987). A dual system for producing self-repairs in spontaneous speech: Evidence from experimentally elicited repairs. *Cognitive Psychology, 19.*

Yngve, V. (1960). A model and a hypothesis for language structure. *Proceedings of the American Philosophical Society, 104,* 444-466.