IPF: An Incremental Parallel Formulator

Koenraad J.M.J. De Smedt
Nijmegen Institute for Cognition research and Information technology (NICI)
University of Nijmegen
Postbus 9104
NL - 6500 HE Nijmegen
The Netherlands
E-mail: desmedt@hnykun53.bitnet

A computer simulation model of the human speaker is presented which generates sentences in a piecemeal way. The module responsible for Grammatical Encoding (the tactical component) is discussed in detail. Generation is conceptually and lexically guided and may proceed from the bottom of the syntactic structure upwards as well as from the top downwards. The construction of syntactic structures is based on unification of so-called *syntactic segments*.

Keywords: incremental sentence generation, parallelism, unification grammar, Dutch.

Published in: Dale, R., Mellish, C. and Zock, M. (eds.) Current research in natural language generation (pp. 167-192). London: Academic Press, 1990.

1 Introduction

Introspection, speech errors and psycholinguistic experiments suggest that natural sentence generation is planned in distinct stages and proceeds in a piecemeal fashion. It is generally accepted now that two main modules can be distinguished in a generation system: a *Conceptualizer* determines the content of the message, and a *Formulator* builds the necessary grammatical structures for each message [Levelt 1989; Garrett 1980]. The output of this linguistic module is then articulated by the *Articulator*, a motor component. Table 1 gives an overview of these modules and presents the Formulator in more detail.

Table 1
Components of the natural language generation process

A. CONCEPTUALIZER (What to say?)
B. FORMULATOR (How to say it?)
B.1. Grammatical Encoder (preverbal messages to surface structures) • lexicalization (lemma selection) • formation of syntactic structures (functional and surface structures)
B.2. Phonological Encoder (surface structures to phonetic plans) • selection of lexical forms (internal composition of words) • prosodic planning (intonation contour, metrical structure)
C. ARTICULATOR (Say it!)

Although the modules operate in sequence, there is no need to process only units corresponding to whole sentences on each level. Therefore, it is suggested that these modules can operate independently and in parallel on different fragments of an utterance [Kempen and Hoenkamp 1987; De Smedt and Kempen 1987]. E.g., while the Formulator is constructing the syntactic form of one conceptual fragment, the Conceptualizer can simultaneously plan the next message.

Moreover, there may be parallelism *within* the Formulator itself; in particular, the syntactic tree formation process does not necessarily proceed in a linear fashion. Sister branches of a syntactic structure may be worked out in parallel by separate processes. Some speech errors of the exchange type [Garrett 1975] reflect this form of computational simultaneity. Figure 1 illustrates the global operation of incremental parallel generation.

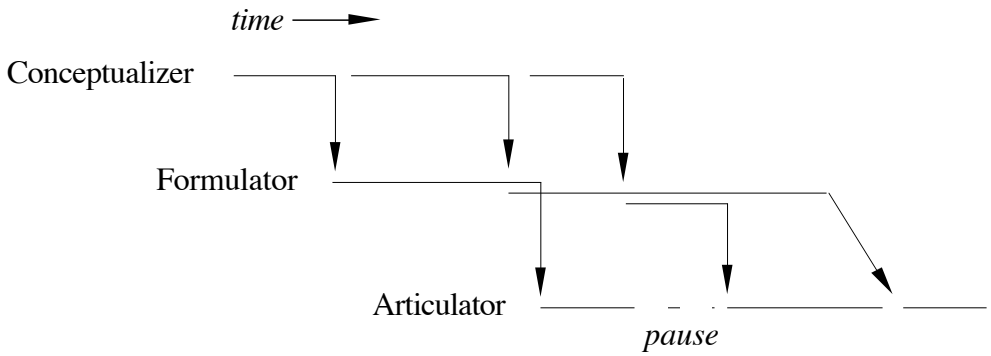


Figure 1
Parallelism in the sentence generation indicated by means of overlapping lines

A natural language generation system which operates according to these assumptions must be supported not only by an incremental and parallel generation strategy, but also by a grammar formalism which is designed to meet the constraints imposed by an incremental mode of generation. In previous work, the following three requirements have been put forward to allow maximally incremental sentence generation [Kempen 1987]:

1 Because it cannot be assumed that conceptual fragments which are input to the Formulator are chronologically ordered in a particular way, it must be possible to expand syntactic structures *upwards* as well as *downwards* (cf. De Smedt and Kempen [1987]).
2 Because the size of each conceptual fragment is not guaranteed to cover a full clause or even a full phrase, it must be possible to attach *individual* branches to existing syntactic structures.
3 Because the chronological order in which conceptual fragments are attached to the syntactic structure does not necessarily correspond to the linear precedence in the resulting utterance, the language generation process should exploit variations in word order as they are made necessary by the partial utterance, but observing language particular restrictions on word order.

In order to meet these criteria, Kempen [1987] proposes a formalism based on *syntactic segments*. This grammar, which was later called *Segment Grammar* (SG), is further developed by De Smedt and Kempen [forthcoming; De Smedt 1990]. After a brief overview of SG, it will be shown how a Formulator can exploit this grammar formalism to generate sentences in an incremental and parallel mode. A computer simulation program called IPF (Incremental Parallel Formulator) has been implemented on the basis of SG for a small but interesting subset of the Dutch grammar and lexicon.

2 Segment Grammar

In order to meet the demands of incremental sentence generation mentioned above, Kempen [1987] proposes a formalism for constructing syntactic structures out of so-called syntactic *segments* which each represent one immediate dominance (ID) relation. A segment consists of two nodes representing grammatical categories, and an arc representing a grammatical function. They are graphically represented in vertical orientation, where the top node is called the *root* and the bottom node the *foot*. The syntactic structure shown in Figure 2 consists of four segments.

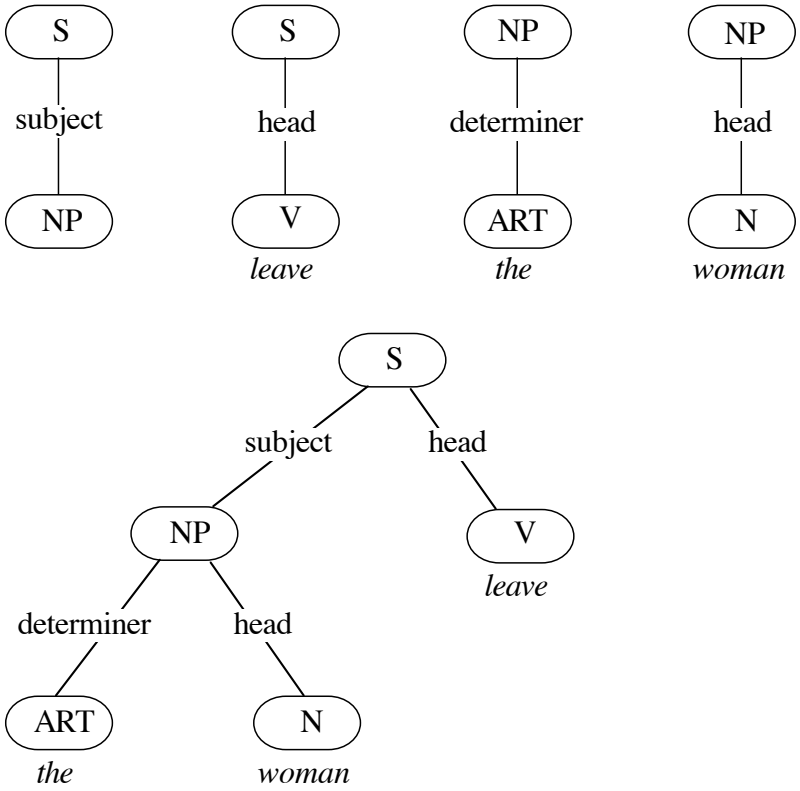


Figure 2
Syntactic segments and syntactic structure for the sentence "The woman left."

Syntactic structures in SG are composed by means of an elementary operation which I will call *local unification* of nodes. This operation is applicable in two situations:

1 When the root of one segment unifies with the foot of another, two segments are *concatenated*; causing the syntactic structure to grow in depth.

2 When two roots unify, the segments are *furcated*, causing the structure to grow in breadth only. Furcation is not possible in other formalisms such as Tree Adjoining Grammars (TAGs; [Joshi 1987]) or Phrase Structure Grammars; hence SG differs from these other formalisms in that sister nodes can be incrementally added.

Agreement is enforced by specifying which features are shared between the root and the foot of a segment. This sharing relation is the same as *reentrancy* in other unification-based grammars [Shieber 1986]. The unification of nodes which share features causes 'feature transport' in the syntactic structure. Actually, shared features are not transported but unified. Subsequent unifications may eventually cause many nodes in the structure to share the same features.

Segments and their components are modeled on a computer as *objects* in the object-oriented language CommonORBIT, an extension of Common LISP [De Smedt 1989, 1987, 1990]. The *aspects* (or *slots*) of an object representing a node are interpreted as features. Aspects may also contain procedural knowledge. Thus feature structures are represented as active computational objects rather than passive stores of information. This allows the generation algorithm itself to be be distributed over a number of objects.

The incremental addition of constituents to a syntactic structure suggests that knowledge about the linear precedence (LP) of constituents is kept separate from

knowledge about immediate dominance (ID). An SG assigns two distinct representations to a sentence, an *f-structure* (or *functional structure*) representing functional relationships between constituents, and a *c-structure* (or *constituent structure*) which represents left-to-right order of constituents.

3 An overview of IPF

We will now be mainly concerned with *Grammatical Encoding,* which is that part of the Formulator which is responsible for the construction of syntactic structures expressing a speaker's intention. Little attention will be given to the origin of the semantic structures which are input[1] to this module. To demarcate the scope of the computer model, it will be assumed that three types of semantic information enter the Formulator:

1 *Semantic concepts*. These are references to entities, events, etc. in the domain of discourse which are to be referred to in the utterance.

2 *Semantic roles*. These are deep *case* relations between concepts. There is no special meaning attached to the case labels; they simply serve to distinguish which concept has which case.

3 *Features*. For simplicity, it is assumed that these are prepared by the Conceptualizer in a rather language-specific form and can thus readily be used as syntactic features. Examples are definiteness, number, etc.

On the output side, the details concerning the realization of surface structures as phonologically specified strings will not be dealt with here. The output of the Grammatical Encoder consists of c-structures which are taken as input by the Phonological Encoder, presumably in a left-to-right fashion. C-structures contain syntactically specified words in order and are incrementally derived from f-structures. During generation they are provided with a constellation of features which allow the Phonological Encoder to produce phonologically specified strings in order. A schematic overview of the formulation process is given in Figure 3.

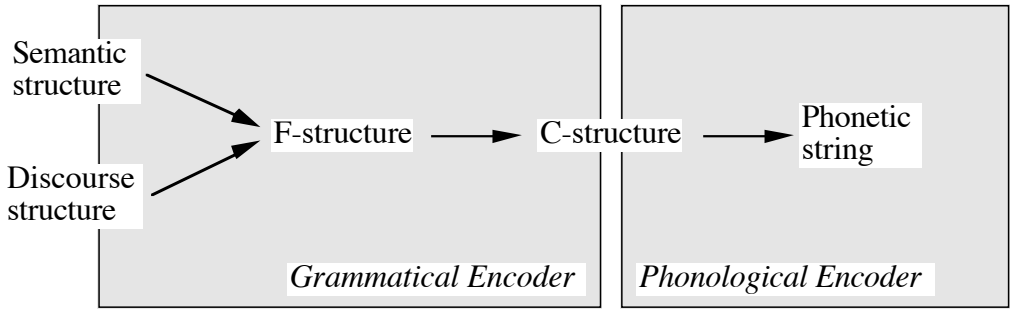


Figure 3
The generation of subsequent linguistic descriptions during the formulation stage

[1] Discourse structures are also input to the Formulator. However, since IPF at present handles only individual sentences, discourse information is not taken into account here, except in the disguise of features such as *definiteness*.

4 The generation of f-structures

The Grammatical Encoder first creates f-structures, which reflect grammatical relations between constituents. Meaning elements are encoded as syntactically categorized lexical elements, syntactic relations and syntactic features. These are incrementally added to form a syntactically coherent functional structure.

4.1 Lexical guidance

If syntactic structures have a conceptual origin, then the subcategorization properties of lexical items rendering those concepts in the input must clearly be taken into account [Kempen and Hoenkamp 1987]. E.g., the similarly structured conceptual representations underlying (1a,b) result in different syntactic structures due to the differing subcategorization properties of *know* and *want*. Constructing a complete syntactic frame before selecting lexical items could result in ungrammaticality, as in (1c); therefore, generation must be conceptually and lexically driven.

(1) a. John knew he hit Peter.

b. John wanted to hit Peter.

c. *John wanted he hit Peter.

On the other hand, lexical items to be incrementally incorporated in a partially constructed sentence are subject to categorial restrictions imposed by the syntactic structure which has been built up to that point. E.g., suppose that in (1b) the verb *hit* is chosen first; a subsequent lexicalization of its direct object is restricted to an NP. Incremental generation is also partially syntax-driven in this sense.

Moreover, IPF does not assume any particular order in which conceptual material is input into the Formulator. Consequently, it is very well possible that the direct object NP is created first; a subsequent 'upward expansion' of the syntactic structure may then cause the NP to obtain a role in an S which has been created at a later stage. Thus, subcategorization restrictions do not always propagate from the top of the tree downward; in the case of upward expansion, restrictions propagate upward as well.

4.2 The lexicon

Lexical entries in SG are not simple strings but are instead structured objects. It will not be surprising that the basic building block of the lexicon is modeled as a segment. A *lexical segment* is a segment where the foot is a word. Content words are modeled as lexical segments where the arc is *head,* whereas function words have different grammatical relations. Like in X-bar theory, the category of the phrasal node is determined by that of its head, e.g. NP-head-NOUN, S-head-VERB, PP-head-PREPOSITION, etc. Examples of the English lexicon are NP-head-CITY (a nominal lexical segment) and S-head-TRY (a verbal lexical segment). The representation of lexical entries as lexical segments allows a clear separation of two kinds of information traditionally assigned to words, as schematically represented in Figure 4:

1 *Syntactic/semantic* information: syntactic category and valence (in the form of a subcategorization frame), as well as the meaning content of the lexical entry, are assigned

to the roots of lexical segments (the phrases). This part of a lexical entry is sometimes called the *lemma* [Kempen and Huijbers 1983].

2 *Morpho-phonological* information: morphological class associated with a categorial label (part of speech), as well as the phonological sound form, are associated with the feet of lexical segments (the words).

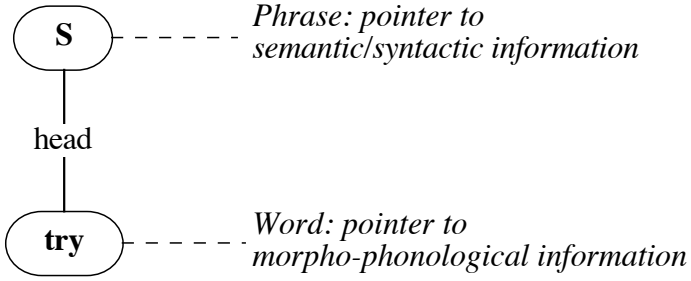


Figure 4
The distribution of information in a lexical segment

Since meaning is associated with the root of the segment, the segment is always accessed through its root in generation (whereas in parsing it could be accessed via the lexeme on the foot). An SG further contains a set of lexical segments for functorization and a set of non-lexical segments for intercalation (see below). Since these segments have in principle the same form as the segments in the lexicon, there is no true distinction between the 'grammar' and the 'lexicon'. All language-specific knowledge is contained in a broad lexicon of segments. Hence an SG is a *lexicalized grammar* [Schabes, Abeillé and Joshi 1988].

Because lexical segments always link a word to a phrase, the lexicon is thus essentially a *phrasal* lexicon. In such a lexicon it is straightforward to include multi-word phrases in the form of ready-made furcations of segments. In this way, idioms and frequently used syntactic fragments can readily be stored. Idioms and other multi-word phrases are thus more or less fully specified syntactic structures[2] comparable to the trees proposed in TAG by Abeillé & Schabes [1989].

It will now be explained in detail what happens when a conceptual message enters the Formulator. Recall that there are three kinds of input messages to the Formulator: semantic concepts, semantic roles, and features. To explore incremental production in a fine-grained way, each input message is processed individually. Each conceptual fragment will spawn its own parallel formulation process which may overlap with other processes.

4.3 Lexicalization

The first kind of message is a request to formulate a semantic *concept*. For each such input, a linguistic *sign* is created with the semantic concept as its referent. An example input (in LISP notation) is the following:

```
(FORMULATE (A SIGN
                (CONCEPT (AN APPLE)))
```

[2] For ease of storage, the current implementation stores multi-segment expressions as a collection of individual segments which are not composed until they are activated.

This *sign* is actually nothing else than a generic, empty syntactic category, to be refined to a more specific category by the Formulator. In contrast to Steels and De Smedt [1983] who use the inheritance hierarchy to refine a linguistic sign, this kind of refinement is performed by an independent lexicalization process, which selects lexical entries and subsequently unifies the sign with one of these entries. We will not be concerned here with the semantic aspects of lexicalization, but only with the syntactic aspects.

The process of lexical selection can be controlled by different control architectures, e.g., parallel or sequential. It is likely that the human speaker accesses lexical entries in parallel (Levelt, 1989), so IPF selects an entry from competing parallel processes. However, since the sign to be refined can only unify with one entry at a time, lexicalization proceeds in a largely sequential mode anyway. If the sign to be lexicalized is unrelated to any other sign, then obviously the first refinement by unification will succeed. This is exemplified in Figure 4.1.

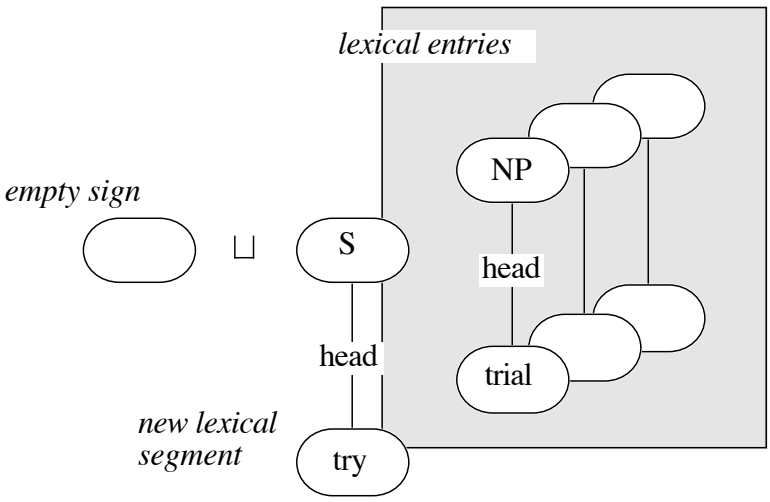


Figure 5
Refinement by unification

However, if the sign expressing the concept is already syntactically related to another one in the current sentence, its lexicalization may be subject to subcategorization restrictions. Thus, unification may fail; if so, another lexical entry is tried, and so on. If no lexical entries succeed in the unification process, a syntactic dead end is signalled and perhaps a repair or restart occurs. Figure 6 illustrates this selection process.

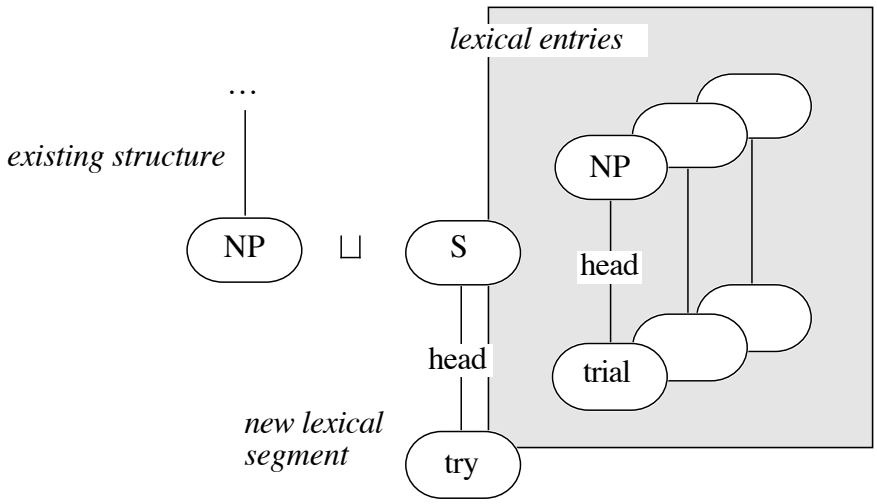


Figure 6
Subcategorization affects lexicalization: when the verbal lexical entry *try* has been rejected, the nominal lexical entry *trial* will unify with the existing NP

SG is a lexicalist grammar in the sense that the lexicon is an autonomous component where lexical entries are generated by lexical rules. These processes include, e.g., passivization (the relation between an active lexical entry and a passive one), nominalization (the relation between a verbal and a nominal lexical entry) and other derivational processes. All lexically governed, bounded, structure-preserving processes are indeed treated lexically [Bresnan 1982]. Thus SG disavows structure-changing transformations in favor of having just one level of f-structure. When needed, the lexical selection process will generate new lexical entries by means of lexical rules.

4.4 Case, grammatical function, and intercalation

The Formulator may receive a request to formulate a case relation between two signs. Case relations signal which roles are filled by semantic concepts in other concepts. An example input is the following:

```
(LET ((SIGN1 (A SIGN
                (CONCEPT 'OTTO)))
       (SIGN2 (A SIGN
                (CONCEPT (AN EAT))))
    (DEFINE-CASE SIGN1 'AGENT :IN SIGN2))
```

The bridge between descriptions of situations and syntactic structure is provided by means of a *case frame* which assigns syntactic roles to particular participants in the situation represented by the sentence. Since lexical entries in SG are phrasal categories, it is logical to attach case frames to the roots of lexical segments rather than to the feet. A case frame is represented as an association list, where with each case a syntactic function is associated. Sometimes more information may be associated with the function, e.g. categorial restrictions, a 'surface' case marker in the form of a preposition, and an indication of whether this function is optional (marked by a 0) or obligatory. Furthermore, a case frame includes not only 'pure cases' (Tesnière's [1959] *actants*), but also

modifiers or adjuncts, i.e., Tesnière's *circonstants*). An example of a Dutch case frame is the following:

```
((AGENT (SUBJECT))
 (THEME (DIRECT-OBJECT NP))    ;categorial restriction
 (LOCATION (MODIFIER IN-LEMMA) 0)   ;prep = surface case
marker
 (DIRECTION (MODIFIER NAAR-LEMMA) 0))   ;0 = optional
```

When the Formulator receives a request to formulate a semantic role between two linguistic signs, this role is looked up in the case frame of the superior sign. This, of course, presupposes that this constituent has already been lexicalized, for otherwise no case frame is available. If necessary, the Formulator process waits until lexicalization has occurred. The second step consists of looking up the grammatical relation associated with the semantic role. This will yield corresponding segments, e.g., an S-subject-NP segment. Such segments are called *intercalating* segments, because they relate one phrase to another. Thus, the lexicon directly constrains the choice of intercalating segments to be incorporated in an f-structure, e.g., the choice between S-subject-NP or S-subject-S. When more than one intercalating segment is possible, they are tried one after the other until one fits between the two phrases. If the foot has not been lexicalized yet—which is very well possible—then clearly the first intercalation will succeed; this must be considered a random guess. The occurrence of a lexicalization later on in the context of an existing intercalating segment may invalidate this choice, and backtracking should occur. The intercalation mechanism is schematically shown in Figure 7.

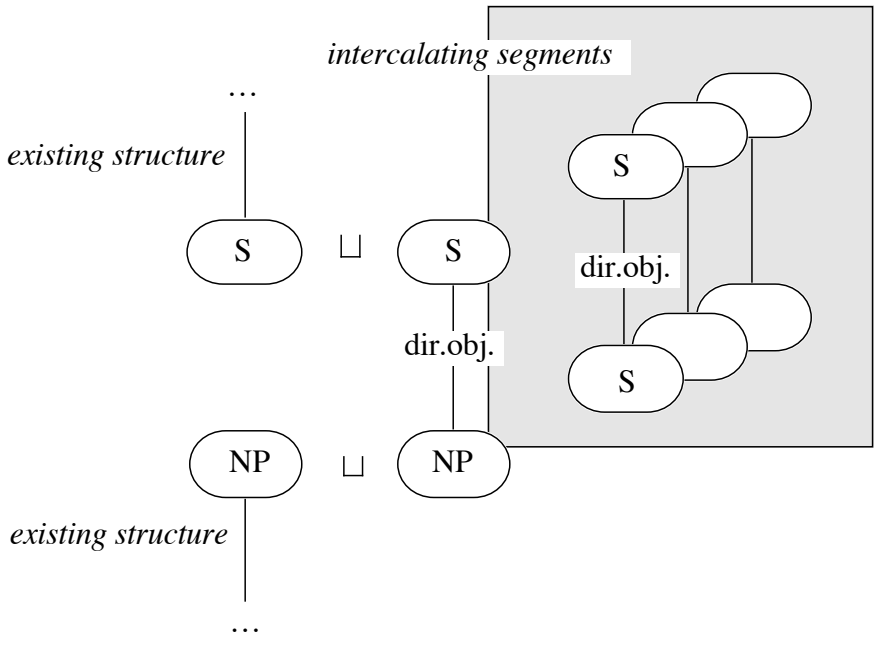


Figure 7
The intercalation process

4.5 Features and functorization

Features (e.g., *plural* or *definite*), are formulated by simply assigning them to a node. Certainly such semantico-syntactic features are derivable from other semantic or discourse

information, but this will not concern us here. Thus the Formulator may receive e.g. the following request:

```
(LET ((SIGN1 (A SIGN
                  (CONCEPT (AN APPLE))))
   (DEFINE-FEATURES SIGN1 '(DEFINITE + PLURAL -)))
```

It will be assumed that this attachment of features to linguistic signs occurs *before* they are lexicalized. This is because they may actually restrict lexicalization, for they play a role in the unification with lexical segments. Attachment *after* lexicalization is not guaranteed to be successful and might require backtracking. The main role of features is that they guide various *functorization* processes, e.g. the addition of function words or inflections. The manner in which features trigger functorization is language dependent. E.g., when a sentence in Dutch gets the value '+' for the feature PERFECT, an auxiliary is added. Other languages such as Latin, e.g., share the feature PERFECT between S and the finite verb, which undergoes an inflectional change.

Features which trigger inflectional change may simply be shared. On the other hand, features which surface as function words will give rise to the addition of one or more segments, which are called *functor* segments. These segments are associated with categories, e.g., determiners are associated with the NP and auxiliaries are associated with the S. Unification is used as a general mechanism to choose among various possible functor segments. In Dutch, e.g., one of several articles can be added, depending on the definiteness, gender and number of the NP[3]. Unification will try to unify the roots of all these functor segments with the NP until one succeeds. In principle, this could be a parallel operation, except that the NP cannot be involved in more than one unification at the same time. The addition of articles by means of functorization in the NP is schematically represented in Figure 8.

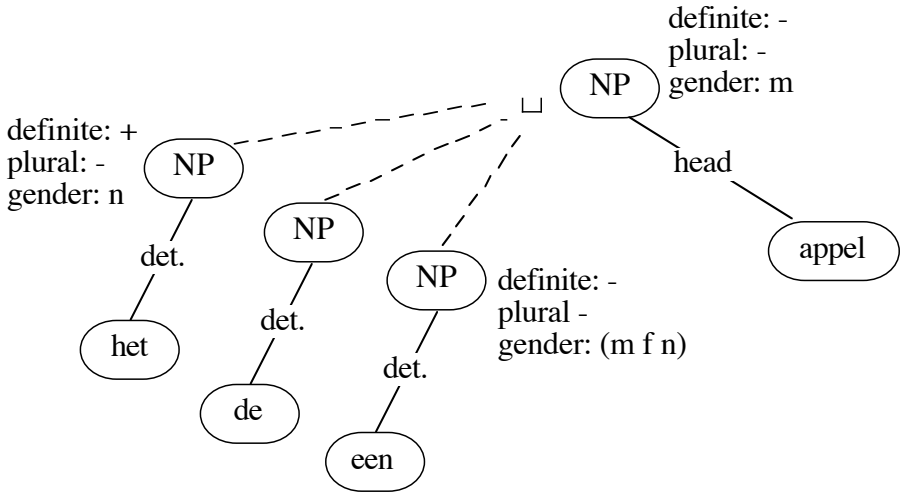


Figure 8
Functorization of the NP in Dutch

[3] It must be added that functorization of the NP is slightly simplified here, for it deals only with articles, and does not account for quantifiers and other elements which may occur in conjunction with—or in replacement of—articles.

On the S level, functorization may result in the addition of auxiliaries. This is performed in a fashion similar to functorization in the NP. In addition, it is decided which verb (main or auxiliary) shares with S the feature *finite*. A finite verb agrees with its matrix sentence, in the sense that features like *person* and *plural* are shared between them[4].

5 The generation of c-structures

F-structures, as constructed by the process explained above, express ID relations but do not express any left-to-right order of constituents. By way of example, f-structure (2e) is assigned to the Dutch sentences (2a-d). F-structures are complete when the head and other obligatory segments (according to the valence information in the phrasal nodes) have been incorporated and when the functorization process has taken place. The assignment of left-to-right positions to constituents is modeled as the piecemeal derivation of a different kind of structure—a c-structure.

Somewhat like ID/LP format for PS rules, SG handles ID and LP separately. However, there are two crucial differences. First, whereas a PS-based system specifies a relative ordering of sister nodes, SG assigns a position to a constituent independently of its sisters; therefore, a system of *absolute* positions is used. Second, the assignment of LP in SG may be attended by a revision of ID relations. Consequently, the ID relations in the f-structure and those in the c-structure for a sentence may not be isomorphic. E.g., c-structure (2f) is assigned to (2a). For clarity, the internal details of the relative clause are left out.

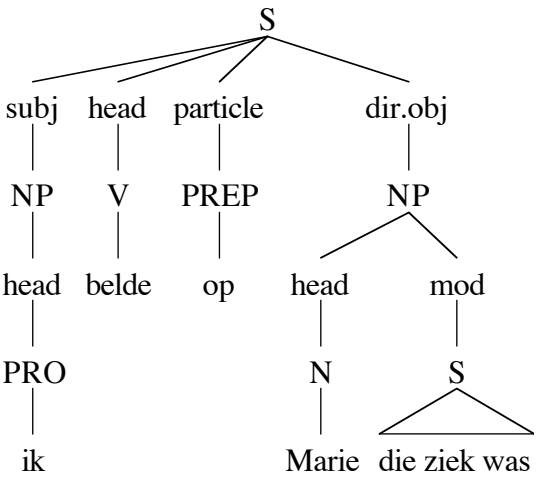
(2) a. Ik belde Marie op, die ziek was. (I called up Mary, who was ill)

b. Ik belde Marie, die ziek was, op. (id.)

c. Marie, die ziek was, belde ik op. (id.)

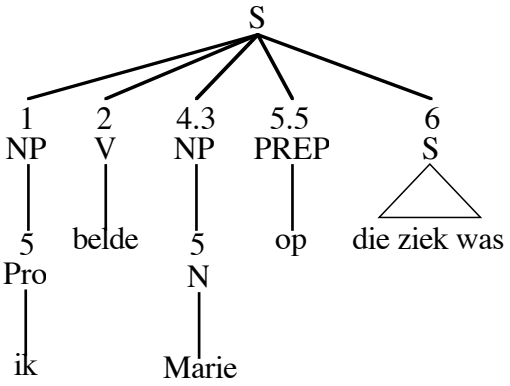
d. Marie belde ik op, die ziek was. (id.)

e.



[4] Following a classical TGG approach, this could also be modeled as the addition of a segment with as its foot a constituent +INFL.

f.



5.1 Destinations

The procedure which assigns left-to-right positions works in a bottom-up fashion: the foot node of a segment is attached in the c-structure directly under its *destination*, which is normally the root of the segment. The destination of a constituent is determined by its mother in the f-structure, i.e., the node which is root of the segment where the constituent is the foot. Normally, the address which the matrix constituent assigns as destination of its dependents is the matrix constituent itself, i.e., ID relations in the c-structure are by default the same as those in the corresponding f-structure. Figure 9 is a schematic representation of the destination procedure.

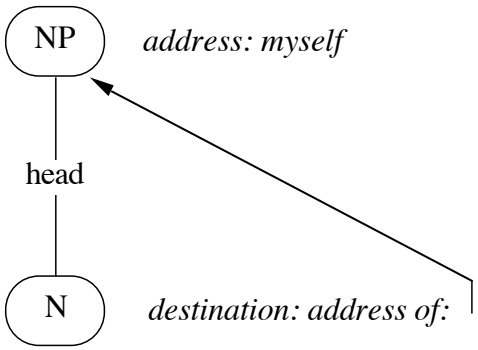


Figure 9
Finding the destination of a node via the address of its mother

Such indirect determination of the destination may seem complicated, but it guarantees that the *root* node of a segment in f-structure exerts control over the ID relation of the *foot* node. This will prove useful in the treatment of constructions where nodes go to higher-level destinations, e.g. the extraposed relative clause in (2c). C-structures which are non-isomorphic to the corresponding f-structures may account for discontinuous constituency. This is discussed more fully in De Smedt and Kempen [1990].

5.2 Holders

Since f-structures are constructed in a piecemeal fashion, it is natural to assign word order incrementally as well. As soon as a node has been lexicalized and attached to its mother in the f-structure, IPF attempts to assign it a left-to-right position in the corresponding c-

structure. Because not all constituents are available at the same time, it is difficult to encode word order relative to other constituents. Therefore, SG prefers an *absolute* order of constituents. For this purpose, a *holder* is associated with each phrase. A holder is a vector of numbered slots that can be 'filled' by constituents. Figure 10 shows some holders associated with c-structure (2f).

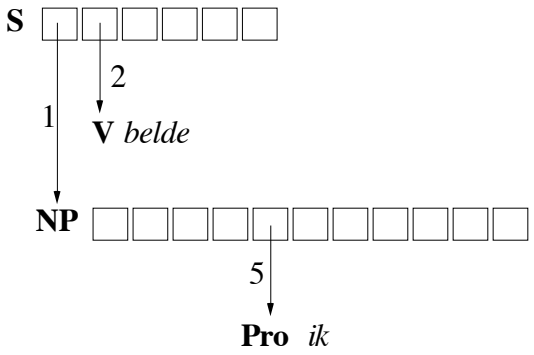


Figure 10
Diagram showing some holders for (2f); the first and second positions of the S have just been occupied

The foot node of each segment in the grammar has a feature POSITIONS which lists all possible positions that the node can occupy in its destination. These possibilities can be seen as language-specific constraints on word order variation. Constituents will try to occupy the first available slot. E.g., the subject NP in Dutch may occur either in sentence-initial or third position (cf. examples (2a) vs. (2c)). In the grammar for Dutch this is specified so that the foot of an S-subject-NP segment may go to holder slots 1 or 3. When a subject NP is assigned a position in the holder of its destination, it will first attempt to occupy position 1. Suppose that position 1 has already been occupied by another constituent, due to earlier conceptual input, it will attempt to go to position 3. A schematic overview of such a situation is given in Figure 11.

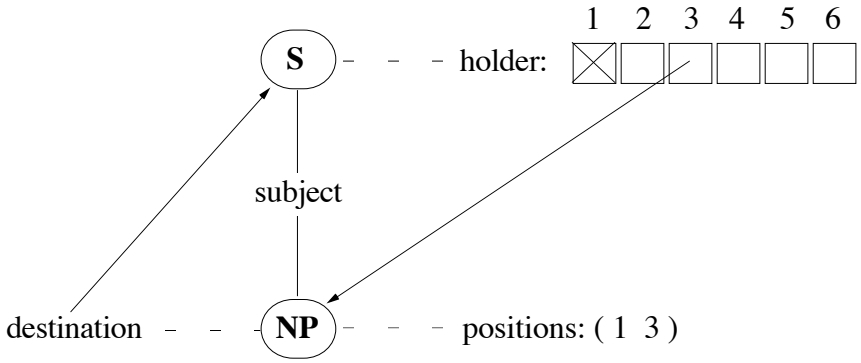


Figure 11
Destination and linearization processes: assign NP to the third slot in the holder of its destination because the first slot is already taken

Such a situation may give rise to a word order variation like (2c,d), where *ik* takes third position rather than first. If the utterance has proceeded beyond the point where a

constituent can be added, a syntactic dead end occurs and a self-correction or restart may be necessary. Alternatively, the constituent may end up somewhere else. E.g., the final slot in the S is a 'dump' for late material (*right dislocation*). The relative clause in (2a,d) is an instance of such a construction.

In order to utter partial sentences, it must be possible to determine the word order of those constituents to be uttered independently of any constituents which are to be incorporated in the utterance at a later stage. For example, it must be possible to determine the word order of fragment (3a) correctly, avoiding e.g. the ungrammatical order in (3b):

(3) a. Yesterday, I …

b. *I, yesterday …

Although human speakers do not always make perfect sentences, and sometimes produce clear ordering errors, it seems generally possible, at least in languages like English and Dutch, to determine the order of single fragments one after the other during incremental production. In fact, it seems that *incremental production is only possible if for the assignment of left-to-right position to a constituent, the simultaneous presence of other constituents of the phrase is not required.* If this empirical claim is true, the necessary knowledge to determine word order can be encoded locally on the level of single segments, as is done in SG.

The number of holder slots in a Dutch clause is substantial. In order to keep an overview, positions *within* positions are sometimes used. The Dutch sentence can be divided into six main parts, each having its own internal ordering. Decimal notation is used to represent such slots, e.g., *3.2* is the second slot in the third main slot. Some holder slots can be occupied by a single constituent only; others may be occupied by an unspecified number of constituents, e.g., an indefinitely long list of APs in front of a noun.

5.3 Topicalization, accessibility and word order variation

In IPF, there is no explicit feature signaling what is to be topic of an utterance. Rather, what is topic is implicitly coded in the order in which the Conceptualizer passes concepts to the Formulator. We assume that pragmatic notions such as topic/focus strongly correlate with conceptual accessibility, i.e., the topic in the discourse may be conceptually more accessible and hence tends to be realized sooner (cf. Bock and Warren [1985]).

Similarly, constituents which are long or complex and hence consume more processing time in the Grammatical Encoder will tend to move toward the end of the utterance, because earlier positions are likely to have been taken up by other constituents. Thus, the tendency of such constituents to occur at the end of the sentence is not a hard rule of linguistic structure, but an emerging property of the generation process as it is proposed here.

Experience with the implementation of IPF/SG shows that these timing factors can be simulated in a program. Alternative orderings of the same conceptual input stimuli into the Formulator can indeed result in word order variations on the surface level. E.g., the utterances (4) were generated by IPF under circumstances which differed solely by the fact that in (4a) the concept underlying *Otto* was given sooner than in (4b), while in (4b) the concept underlying *appel* (apple) was given sooner.

(4) a. Otto … heeft een appel gegeten.
(Otto has eaten an apple)

b. Een appel … heeft Otto gegeten.
(id., variation in word order)

Similarly, in SVO languages like English, passivization can be triggered by conceptual accessibility. If a constituent which would be object in an active sentence is fronted, it becomes a likely candidate for the subject function; hence a passive lemma is appropriate.

Summing up, if there is indeed a strong link between fronting, topicalization and conceptual accessibility [Bock 1987; Bock and Warren 1985; Levelt 1989] then topicalization by fronting (and by means of other mechanisms like passivization) is an important emerging property of the generation strategy. This is not to say that accessibility is the *only* factor determining the order in which conceptual fragments are input into the Formulator. It seems that certain rhetorical effects unrelated to accessibility may affect word order as well. Kolk's [1987] adaptation theory suggests that the conceptual input to the Formulator can be subject to manipulation which is learned by the speaker. Thus, it is very well possible that certain factors determining word order are coded in terms of an order imposed on conceptual elements as they enter the Formulator.

Although experiments with IPF show that conceptual accessibility can indeed account for variations in word order, the timing of the parallel processes in Grammatical Encoding is not very sophisticated, since the current implementation gives all parallel processes the same priorities; they are therefore capable to do roughly the same proportion of processing in the same period of time. In a multiprocessing environment it is often possible to allot more computational resources to one process than to another. This would be another way to affect word order by means of timing in the generation process.

5.4 A small example

In order to make the foregoing mechanisms more concrete, we now present a small example of Grammatical Encoding. The example will involve the choice between (4a) and (4b). The input consists of the following LISP expression:

```
(DEFUN S-TEST ()
   "Test for: Otto heeft een appel gegeten."
   (LET ((SIGN1 (A SIGN (SEMANTIC-REFERENT (THE-OBJECT
'OTTO))))
         (SIGN2 (A SIGN (SEMANTIC-REFERENT (AN EAT))))
         (SIGN3 (A SIGN (SEMANTIC-REFERENT (AN APPLE)))))
     ;; first define features; these could be computed
     (DEFINE-FEATURES SIGN1 '(DEFINITE + PLURAL -))
     (DEFINE-FEATURES SIGN2
       '(INTERROGATIVE - PERFECT + FUTURE - PAST - FINITE +))
     (DEFINE-FEATURES SIGN3 '(DEFINITE - PLURAL -))
     (FORMULATE SIGN1)
     ;; Otto...
     (SLEEP 5)
     (FORMULATE SIGN2)
     ;; heeft gegeten ...
     (SLEEP 5)
     (DEFINE-CASE SIGN1 'AGENT :IN SIGN2)
     ;; Otto heeft gegeten ... = upward expansion
     (SLEEP 5)
     (FORMULATE SIGN3)
     ;; een appel
     (SLEEP 5)
     (DEFINE-CASE SIGN3 'THEME :IN SIGN2)
     ;; Otto heeft een appel gegeten ... = downward expansion
     ))
```

Note that, apart from the DEFINE-FEATURES commands, there are five commands to the Formulator, three involving FORMULATE and two involving DEFINE-CASE. They are spaced in the time dimension by means of SLEEP commands which each cause the system to be dormant for approximately 5 seconds of real time. Shorter periods of time between the commands to the Formulator cause more competition between parallel processes, as will be indicated below.

Each command to the Formulator results in an independent Grammatical Encoding process which runs in parallel to everything else going on in the system. Figure 12 presents two snapshots of the generation process; f-structures are shown in the upper window while c-structures are shown in the lower window. Figure 12b shows how the sign for the concept *Otto* has stayed ahead of the sign for *apple* and occupies a position earlier in the sentence. This results in a c-structure corresponding to sentence (4a).

a.

b.

Figure 12
Two snapshots of a generation of example (4a)

If the timing in the test input is changed so that there are shorter periods of time between the various inputs, then their computation overlaps more. The chance that the constituents of the sentence occupy alternative left-to-right positions in the c-structure therefore increases. Figure 13 shows how the sign for *Otto* is 'overtaken' by that for *apple* and ends up in holder slot 3 of the S. This results in a c-structure corresponding to sentence (4b).

Figure 13
A snapshot of a generation of example (4b)

6 Concluding remarks

IPF is a computer model for incremental grammatical encoding which exploits the representation of grammatical and lexical knowledge in terms of syntactic segments. It allows the piecemeal input of conceptual material and can construct functional structures not only from the top down, but also from the bottom up. Surface structures (c-structures) are derived in an equally piecemeal fashion. Conceptual fragments are small, although 'chunking' of conceptual material can be achieved by a number of simultaneous inputs.

A lexically guided system is proposed, in the sense that meaning is directly responsible for the choice of appropriate lexical items, which in turn are associated with possible categories. Possible words and categories are filtered by the current syntactic context. Although conceptual changes other than addition of new concepts are currently not

implemented in IPF, they are in principle possible (cf. De Smedt and Kempen [1987]). Conceptual replacement or deletion of concepts, semantic roles or features could be modeled either by undoing previous unifications with functor segments or by the construction of a new sentence structure. The latter seems to be preferred because undoing previous unifications involves a great deal of bookkeeping of past states of the unification space. When a conceptual change invalidates a piece of currently built (and maybe partially uttered) structure, the structure as a whole is discarded, although dependent constituents may be reused.

Within the Formulator, the c-structure produced by the Grammatical Encoder is made accessible to the next substage, the Phonological Encoder. In principle, left-to-right processing of the c-structure by the Phonological Encoder can be achieved by traversing this structure in a depth-first fashion. However, it is possible that the sentence is generated in a 'bottom-up' fashion by means of upward expansion. Should the Phonological Encoder start with bits and pieces or should it wait until a reasonable sentence structure has emerged? We believe that there is no absolute answer to this question, but that both are possible. People sometimes produce very spontaneous speech and sometimes very careful and deliberate speech. It seems that a main determinant of these modes of speech is how the Phonological Encoder interfaces to the output of the Grammatical Encoder. In the case of careful speech, it could be assumed that the Phonological Encoder applies certain heuristics to plan its processing. Certainly there should be a mechanism to prevent 'skipping over' the position of required constituents, e.g., the finite verb of a clause. One solution to this problem consists of marking the positional slots in c-structures which are to be occupied by such required constituents. When the Phonological Encoder finds such a marker, it waits until the position is filled by a constituent which overwrites the marker.

A remaining question is, when a new sentence is to be started. In an incremental mode of sentence generation, the answer is not so straightforward, for a sentence which is in principle complete may be extended by adding another modifier or even a case relation. E.g.:

(5) a. He came to my house … at about ten o'clock … because he needed money.

b. I already told you … that she got married.

A new sentence seems to be triggered by at least two kinds of conceptual addition:

1 When a new concept is not related to anything in the sentence.

2 When a new concept is related to the current utterance, but does not fit into the syntactic structure, because the utterance has proceeded beyond the point where the new constituent can be added (as discussed above). In that case, either the sentence is already complete, and a new one is started, or the current structure is abandoned and a restart is attempted.

References

Abeillé, A. & Schabes, Y. 1989 Parsing idioms in lexicalized TAGs. In: *Proceedings of the 4th Conference of the European Chapter of the ACL,* Manchester (pp. 1-9). ACL.

Bock, J.K. and Warren, R. 1985 Conceptual accessibility and syntactic structure in sentence formulation. *Cognition* 21: 47-67.

Bock, J.K. 1987 Exploring levels of processing in sentence production. In Kempen, G. (ed.) *Natural language generation: New results in Artificial Intelligence, psychology and linguistics*. Martinus Nijhoff Publishers, Dordrecht/Boston: 351-363.

Bresnan, J. (ed.) 1982 *The mental representation of grammatical relations*. MIT Press, Cambridge, Mass.

De Smedt, K. and Kempen, G. [forthcoming] Segment Grammar: a formalism for incremental sentence generation. To appear in *Proceedings of the 4th International Workshop on Natural Language Generation, Santa Catalina Island, 17-21 July 1988*.

De Smedt, K. and Kempen, G. 1990 Discontinuous constituency in Segment Grammar. In *Proceedings of the Symposium on discontinuous constituency, Tilburg, 25-27 January 1990* (pp. 97-111).

De Smedt, K. and Kempen, G. 1987 Incremental sentence production, self-correction and coordination. In Kempen, G. (ed.) *Natural language generation: new results in Artificial Intelligence, psychology and linguistics*. Martinus Nijhoff Publishers, Dordrecht/Boston: 365-376.

De Smedt, K. 1987 Object-oriented programming in Flavors and CommonORBIT. In Hawley, R. (ed.) *Artificial Intelligence programming environments*. Ellis Horwood, Chichester: 157-176.

De Smedt, K. 1989 *Object-oriented knowledge representation in CommonORBIT*. Internal report 89-NICI-01, Nijmegen Institute for Cognition research and Information technology, University of Nijmegen.

De Smedt, K. 1990. *Incremental sentence generation: a computer model of grammatical encoding*. Doctoral dissertation, NICI Technical Report 90-01, University of Nijmegen.

Garrett, M. 1975 The analysis of sentence production. In Bower, G. (ed.) *The psychology of learning and motivation (Vol. 9)* (pp. 133-177). Academic Press, New York.

Garrett, M. 1980 Levels of processing in sentence production. In Butterworth, B. (ed.) *Language production. Vol. 1: Speech and Talk* (pp. 177-219). Academic Press, New York.

Joshi, A. 1987 The relevance of tree adjoining grammar to generation. In Kempen, G. (ed.) *Natural language generation: New results in Artificial Intelligence, psychology and linguistics*. Martinus Nijhoff Publishers, Dordrecht/Boston: 233-252.

Kempen, G. and Hoenkamp, E. 1987 An incremental procedural grammar for sentence formulation. *Cognitive Science* 11, 201-258.

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

Kempen, G. 1987 A framework for incremental syntactic tree formation. In: *Proceedings of the 10th IJCAI, Milan*. Morgan Kaufmann, Los Altos: 655-660.

Kolk, H. 1987 A theory of grammatical impairment in aphasia. In Kempen, G. (ed.) *Natural language generation: New results in Artificial Intelligence, psychology and linguistics*. Martinus Nijhoff Publishers, Dordrecht/Boston: 377-391.

Levelt, W. 1989 *Speaking: from intention to articulation*. MIT Press, Cambridge, Mass.

Schabes, Y., Abeillé, A. and Joshi, A.K. 1988 Parsing strategies with 'lexicalized' grammars: application to Tree Adjoining Grammars. In *Proceedings of COLING '88, Budapest*.

Shieber, S.M. 1986 *An introduction to unification-based approaches to grammar*. CSLI Lecture Notes No. 4. Center for the Study of Language and Information, Stanford.

Steels, L. and De Smedt, K. 1983 Some examples of frame-based syntactic processing. In Daems, Fr. and Goossens, L. (eds.) *Een spyeghel voor G. Jo Steenbergen*. Acco, Leuven: 293-305.

Tesnière, L. 1959 *Eléments de syntaxe structurale*. Klincksieck, Paris.