

Parallelism in incremental sentence generation

Koenraad J.M.J. De Smedt
1994

Abstract

IPF (Incremental Parallel Formulator) is a computer model in which the formulation stage in sentence generation is distributed among a number of parallel processes. Each conceptual fragment which is passed on to the Formulator gives rise to a new process, which attempts to formulate only that fragment and then exits. The task of each formulation process consists basically of instantiating one or more syntactic *segments* and attaching these to the present syntactic structure by means of a general *unification* operation. A shared memory provides the necessary (and only) interaction between parallel processes and allows the integration of segments created by different processes into one syntactic structure. The race between parallel processes in time may partly explain some variations in word order and lexical choice. The system is provided with a Dutch grammar and is implemented in the object-oriented language CommonORBIT on a Symbolics machine with simulated parallelism.

1 INTRODUCTION

1.1 Parallelism in natural language generation

Several arguments can be given to support the assumption that human language processing in general, and generation in particular, exploits parallelism. A first argument relates to human information processing in general. Experiments on the division of attention have led to the *multi-channel* hypothesis, which suggests “a number of independent, special purpose computers (processors and stores) operating in parallel and, at least in some cases, capable of accepting only one message or ‘chunk’ of information for processing at a time” (Allport, Antonis & Reynolds, 1972). This theory suggests that complex tasks depend on the operation of independent, specialized processors which carry out certain subtasks.

A second argument for parallelism relates specifically to sentence generation. Phenomena such as speech errors have been interpreted by psycholinguists as clues to how human speakers plan their utterances. Some kinds of fusion and omission errors seem to originate from the fact that speakers prepare several linguistic expressions at the same time. Due to internal or external factors, the human language processing system may substitute unintended elements for the expected ones. Garrett (1975, 1980) assumes that computational simultaneity is a general condition for such interchanges.

A third, more general argument is neurolinguistic. Circuit switching in the human brain is relatively slow compared to that in present-day computers. In order to account for the speed of human language processing—and indeed human information processing in general—an upper bound of about 100 single consecutive computational steps must be placed on the processing of a sentence (Feldman & Ballard, 1982). This is vastly below the number of steps required by algorithms based on any present-day non-parallel theory of language.

There are several approaches to parallelism in language processing, and to cognitive processing in general. The first approach consists of the introduction of parallelism in a symbolic programming paradigm. The symbolic approach assumes an explicit representation of rules and concepts. Such a mentalistic view puts the grammar at the origin of linguistic behavior (cf. Chomsky, 1959). The second approach is to relinquish symbolic representations and instead represent knowledge at a subsymbolic level. Connectionist models do without an explicit, discrete representation of grammar rules and concepts. The grammar is then no more than an abstraction *a posteriori*. Regularities, and indeed mentality in general, are then statistically emergent properties of subsymbolic behavior (cf. Elman, 1989). In the present chapter, a relatively coarse-grained parallelism in a symbolic framework is explored.

1.2 Incremental sentence generation

Introspection, speech errors and psycholinguistic experiments suggest that natural language generation is planned in distinct stages and proceeds in a piecemeal fashion. It is generally accepted now that three main modules can be distinguished in a generator: a *Conceptualizer* determines the content of a message, a *Formulator* builds the necessary grammatical and phonological structures for each message, and an *Articulator* utters the phonetic representations computed from these structures (Kempen & Hoenkamp, 1987; Levelt, 1989). Figure 1 gives a schematic view of the operation of these modules.

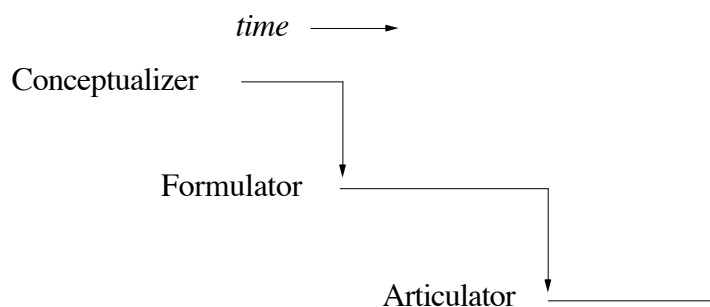


Figure 1
Stages of processing (horizontal) and flow of information (vertical)

It is assumed that the modules representing the stages of language processing are relatively autonomous components. They meet the requirement of informational encapsulation in the sense that their input is of a maximally restricted sort and their mode of operation minimally affected by other components (Fodor, 1983). The flow of information between the modules is downward; in other words, each module passes intermediate structures to the next module but receives no information back from that module. All upward information is a result of monitoring—the inspection of output of each module by a monitor process (Hoenkamp, 1980; Levelt, 1989).

Although these modules operate in sequence, there is no need to process only units corresponding to whole sentences on each level (Kempen & Hoenkamp, 1987; De Smedt & Kempen, 1987). Speakers start to articulate a sentence before the syntactic structure, or even the meaning content of that sentence has been fully determined. Therefore, it is suggested that the modules can operate independently and in parallel on different fragments of an utterance. For instance, while the Formulator is constructing the syntactic and phonological form of one conceptual fragment, the Conceptualizer can simultaneously plan the next message. This *intercomponent* parallelism is depicted in Figure 2.

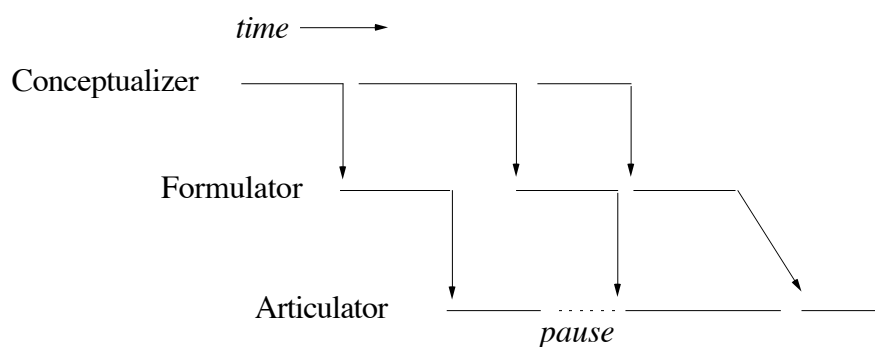


Figure 2
Intercomponent parallelism in sentence generation

Furthermore, there is no *a priori* reason why the Formulator itself should consist of just one sequentially operating process. If the formulation stage does not necessarily proceed in a linear fashion, but can be *distributed* among a number of separate processes, then distinct parts of a syntactic structure could be worked out in parallel. Some speech errors of the exchange type (Garrett, 1975) reflect this form of computational simultaneity. Therefore, an alternative architecture is proposed here,

which allows the Formulator to process different conceptual input fragments at the same time and independently of each other. This *intracomponent* parallelism is shown in Figure 3.

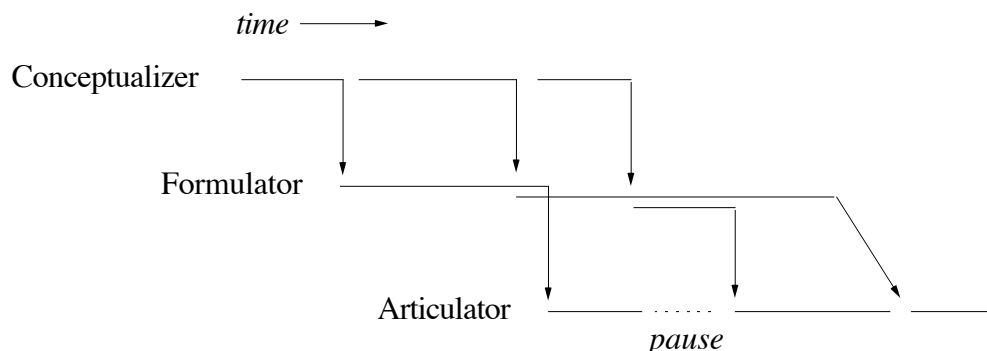


Figure 3
Intracomponent parallelism in the Formulator

The distribution of the formulation task among a number of parallel processes, each of which is responsible for the construction of part of the syntactic structure, may significantly improve the performance of the generator. For instance, newly incoming conceptual fragments which take little processing time can be uttered before older but more difficult fragments are ready¹. This possibility, which is suggested in Figure 3, presupposes that the time when a fragment can be uttered depends not only on the moment when it has entered the Formulator, but also on how much processing time the Formulator spends on it. Thus, the processes which together embody the formulation stage can be viewed as competing, i.e., in a race with each other. The outcome of the race shows up at the surface level in the form of lexical choices and word order variations. For instance, the choice between (1a) and (1b) in spontaneous speech could be a reflection of timing differences in the generation system.

- (1) a. Last week ... I saw John ... in Amsterdam.
- b. I saw John ... in Amsterdam ... last week.

IPF (Incremental Parallel Formulator) is a computer model of incremental sentence generation. In Section 2, it will be described what constitutes Segment Grammar, the grammar formalism used by IPF. In Section 4 it is shown how the formulation stage in IPF makes use of the proposed parallel architecture. Since Segment Grammar is unification-based, Section 3 explains how unifications in SG can be performed in parallel. A comparison with related research concludes the chapter.

2 SEGMENT GRAMMAR

2.1 Requirements for an incremental grammar formalism

A natural language generator which operates according to the assumptions outlined in Section 1 must be supported 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 for the grammar formalism have been put forward to allow maximally incremental sentence generation (Kempen, 1987):

1 Because it cannot be assumed that conceptual fragments which are given as input to the Formulator are chronologically ordered in a particular way, it must be possible to expand syntactic structures upward as well as downward.

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 trees.

3 While the chronological order of incoming conceptual fragments does not necessarily correspond to the linear precedence in the resulting utterance (due to language specific restrictions on word order), the grammar should exploit word order variations to allow the formulation and utterance of fragments as soon as they are available.

In order to meet these criteria, Kempen (1987) proposes a formalism based on *syntactic segments*. After a brief explanation of this formalism, which will be called Segment Grammar (SG), it will be described how this formalism can be exploited in the proposed parallel architecture. Furthermore, the proposed intracomponent parallelism within the Formulator imposes a fourth constraint which was not addressed by Kempen:

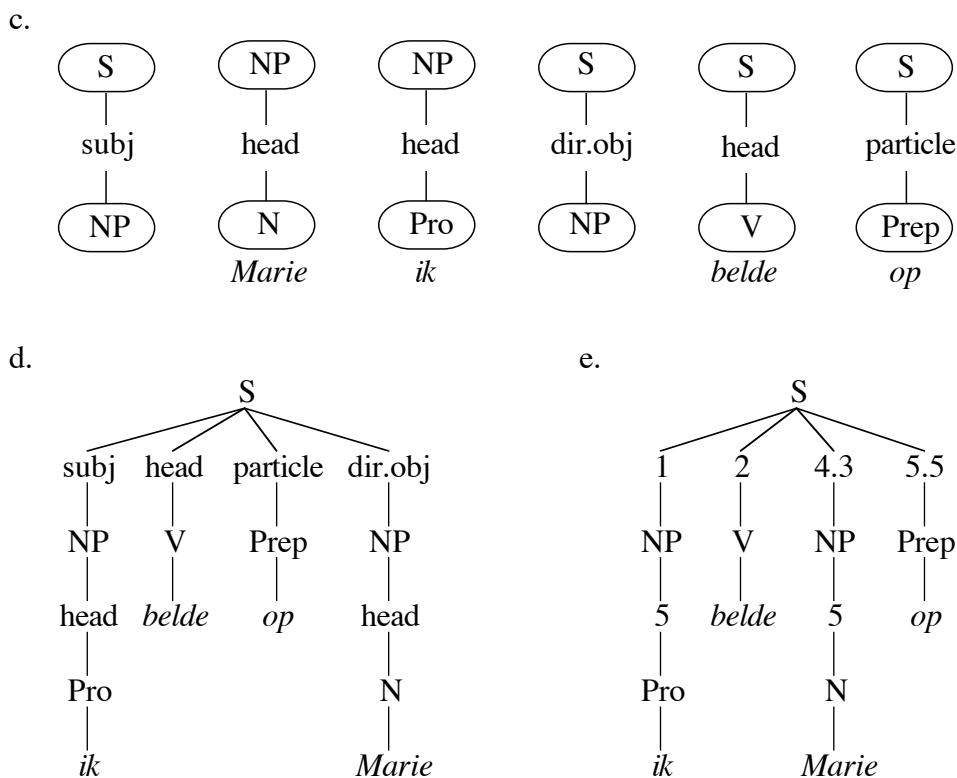
4 Because there is no reason to assume that the Formulator can process only one conceptual message at a time, the grammar must allow branches to be added independently of each other and simultaneously.

It will be shown how grammatical knowledge is distributed in SG among syntactic segments and how this allows the formation of syntactic structures to be distributed among various processors.

2.2 Syntactic segments

In SG, syntactic structures are constructed out of so-called *syntactic segments* which each represent a single immediate dominance relation between nodes. 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*. For instance, the Dutch sentences (2a,b) consist of six segments (2c). Segments join to form a *functional structure* or *f-structure* (2d) by means of a general *unification* operation which merges two nodes into one. This operation is applicable in two situations: (i) when the root of one segment unifies with the foot of another, two segments are *concatenated*; and (ii) when two roots unify, the segments are *furcated*. Furcation is not possible in other formalisms such as Tree Adjoining Grammars (TAGs; Joshi 1987) or grammars based on Phrase Structure (PS) rules; hence SG differs from these other formalisms in that sister nodes can be incrementally added.

- (2) a. Ik belde Marie op.
(I called up Mary)
- b. Marie belde ik op.
(Mary I called up)



F-structures are unordered graphs (sometimes called *mobiles*) expressing grammatical relations between constituents. F-structure (2d) describes sentence (2a) as well as (2b). The assignment of left-to-right positions to constituents is modeled as the incremental derivation of a different kind of structure—a *constituent structure* or *c-structure*. By way of example, c-structure (2e) is assigned to (2a). Left-to-right order of constituents is indicated by means of system of numbered slots. The relation between f-structure and c-structure is somewhat similar to that in LFG (Bresnan, 1982) and will not be further discussed here. For a more complete discussion of SG, the reader is referred to De Smedt and Kempen (1991) and De Smedt (1990a, 1990b)².

2.3 Agreement

SG is a lexicalized grammar (Schabes, Abeillé & Joshi, 1988) in the sense that it distributes grammatical rules among the various syntactic segments which together make up the grammar of a language. By viewing nodes as feature structures in the sense of Shieber (1986), constraints can be specified as features on the foot or root of a segment. An example is the feature *nominative* on the foot of the S-subject-NP segment in Figure 4. Features can be *shared* among the root and foot, for instance, the feature *nominative* in the NP-head-NOUN segment in Figure 4. When both segments are concatenated by unification of the NP nodes, the features are also unified; the new value of a unified feature is the intersection of the old values.

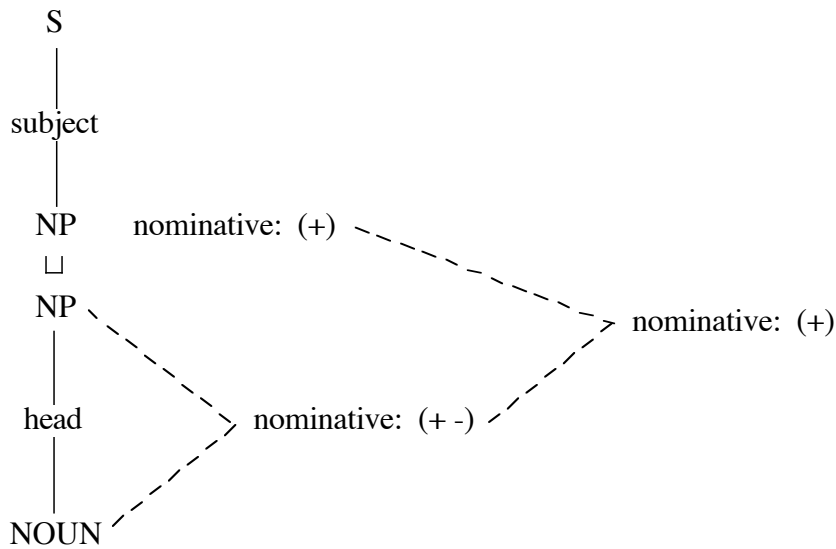


Figure 4
Unification of a shared feature

In a similar way, subject-verb agreement can be modeled by feature sharing in furcated segments. The features *number* and *person* are shared in S-subject-NP as well as in S-head-FINITE-VERB. If such segments are furcated by unifying their S nodes, the shared features in both segments are unified, as depicted in Figure 5.

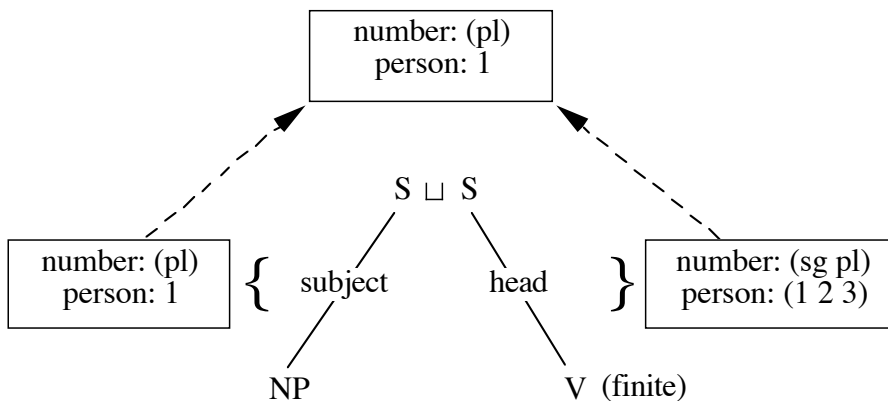


Figure 5
Agreement by means of feature sharing

Agreement in many other unification-based formalisms can be specified between constituents at arbitrary distances by using *reentrant paths*, notated as coindexing in (3), or as the equations (4) and (5).

(3)

Error! (Shieber, 1986)

(4) <subject agreement> = <predicate agreement> (Karttunen, 1984)

(5) VP: (\uparrow subject agreement) = (\downarrow agreement) (LFG)

The SG approach is different by virtue of the fact that agreement can only be specified on the level of individual segments, i.e., as shared features between root and foot. Thus, SG expresses information at the level of the segment, i.e., encapsulated in many separate objects rather than in one large structure or rule base. This makes unification more local and facilitates parallel unification (see Section 4). It seems that such a local specification is sufficient for all purely syntactic kinds of agreement.

3 AN INCREMENTAL PARALLEL FORMULATOR

3.1 Incremental formulation of conceptual fragments

Let us now turn our attention to the question, how conceptual fragments are entered into the Formulator and how they result in the parallel construction of syntactic structures. In order to allow a piecemeal input into the Formulator, it is assumed that the Conceptualizer packages conceptual information into separate preverbal messages of three types:

1 *Semantic concepts*. These are references to entities, events, etc. in the domain of discourse which are to be referred to in the utterance. The formulation of a concept will normally result in the addition of one or more *lexical segments* containing content words at their feet. An example of a lexical segment is the NP-head-N (*Marie*) segment in (2c). The concept “call up” activates a combination of two lexical segments.

2 *Case relations*. These are ‘deep’ cases expressing semantic roles between concepts. There is no special meaning attached to the case labels; they simply serve to distinguish which participants in the situation are expressed. A case relation will result in the addition of a so-called *intercalating segment*, which relates phrases by means of a grammatical relation. An example of an intercalating segment is S-subject-NP in (2).

3 *Features*. For simplicity, it is assumed in this model that certain semantic features are prepared in a rather language-specific form and can thus readily be used as syntactic features. Examples are definiteness, number, etc. Features may result in the addition of *functor segments* containing, for instance, auxiliaries or determiners.

If we assume, with Hoenkamp (1983:18), that incremental generation is guided by the principle “What *can* be uttered *must* be uttered immediately”, then the order in which the Formulator adds segments to the syntactic structure will exert an influence on the order in which they are uttered. Surely this is not the only determining factor: language-specific word order restrictions will in general prevent constituents from being uttered when they do not form a grammatically well-formed sequence. Thinking backwards, the Formulator can only start working on a conceptual fragment when that fragment is made accessible by the Conceptualizer. Extending Hoenkamp’s principle of incremental generation so that “What *can* be formulated *must* be formulated immediately”, we deduce that the order in which conceptual fragments are passed on to the Formulator is—indirectly—also a factor affecting the shape of an utterance.

It is therefore important that conceptual messages can be entered into the Formulator individually rather than grouped in a large conceptual structure. This allows their effective relative ordering and spacing in the time dimension. In particular, time delays between conceptual inputs may simulate their different conceptual accessibilities. The more time there is between successive inputs, the more sequential their formulation will be and the more their ordering in the input will be reflected in the eventual utterance.

Entering successive inputs shortly after one another will cause more overlap in their processing, as if a whole chunk of conceptual material has entered the Formulator.

3.2 Effects on word order and lexical choice

IPF is currently implemented as a pseudo-parallel multi-processing system. Each conceptual fragment which is input to the formulating stage spawns a new process, which attempts to formulate only that fragment and then exits. The task of each Formulator process consists basically of instantiating one or more segments and attaching these to the present syntactic structure by means of unification. A shared memory provides the necessary (and only) interaction between Formulator processes and allows the integration of segments created by different processes into one syntactic structure.

Experience with the implementation of IPF shows that these timing factors can be well simulated in a program. Alternative orderings of the same conceptual input stimuli which are provided to the Formulator can indeed result in word order variations on the surface level. For instance, the utterances (6) are both allowed in Dutch. They can be generated by IPF under circumstances differing solely in the fact that in generating (6a) the concept underlying *Otto* is accessible sooner than in (6b), while in (6b) the concept underlying *appel* (apple) is given sooner. If the concepts are input at roughly the same time, the chances of (6a) or (6b) being generated are about equal.

- (6) a. Otto ... heeft een appel gegeten.
(Otto has eaten an apple)
- b. Een appel ... heeft Otto gegeten.
(id., with word order variation)

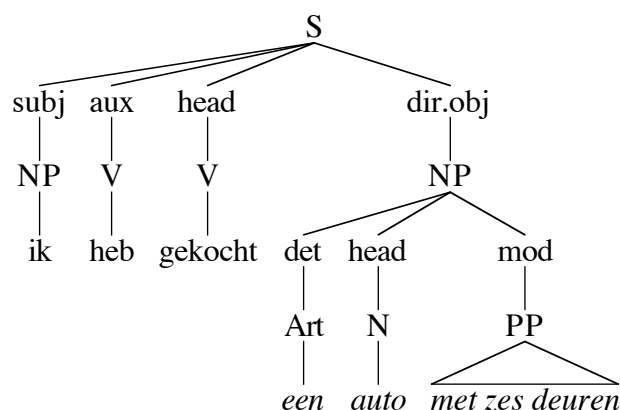
Similarly, in SVO languages like English, passivization can be triggered by conceptual accessibility. If a constituent which would normally be object in an active sentence is fronted, it becomes a likely candidate for the subject function; hence a passive lemma is appropriate.

An additional argument for the proposed architecture can be found in the fact that several languages allow the addition of constituents at the end of the sentence even if this results in discontinuous constituency. Consider, for instance, the Dutch sentences (1) which show such right dislocations:

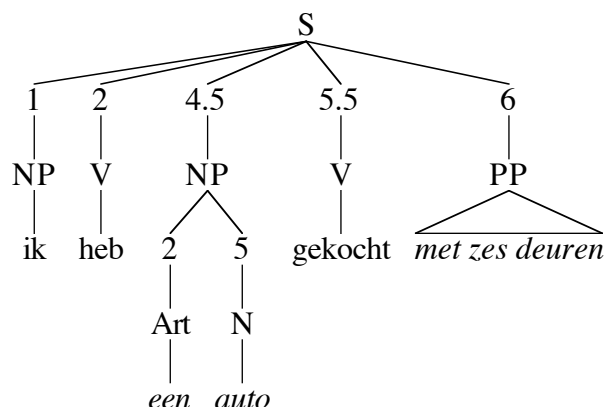
- (7) a. Ik heb een auto gekocht met zes deuren.
(I have bought a car with six doors)
- b. Een van zijn vrienden kwam, die in Brussel woont.
(One of his friends came, who lives in Brussels)

Right dislocation fits naturally into the incremental generation process. In IPF/SG, it is triggered by the fact that the extraposed constituent cannot be added incrementally to the NP in the c-structure, because the utterance has already proceeded beyond that point. Therefore it is exceptionally allowed to move to the S level, which has a slot where some kinds of 'late' constituents can be placed. The c-structure for (7a), which is shown in (8b) therefore has different dominance relations than the f-structure (8b). Furthermore, right dislocation seems to occur more easily if the dislocated constituent is 'heavy', which corroborates the hypothesis that the process load of the constituent—in competition with other ones—may cause it to be 'late'.

(8) a.



b.



Summing up, if there is indeed a strong link between word order, topicalization and conceptual accessibility (Bock, 1987; Bock & Warren, 1985; Levelt, 1989) then topicalization by fronting (and also by means of some 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 passed to 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 pragmatic factors determining word order are coded in terms of an order imposed on conceptual elements as they enter the Formulator.

3.3 Some IPF simulations

In order to make the foregoing mechanisms more concrete, I will now present some simulation examples. IPF is implemented in CommonORBIT (see Section 4) on a Symbolics Lisp Machine. A *program frame* supporting multiple windows is used to display intermediate results of the generation of a sentence (see Figures 6 and 7). F-structures are displayed in the upper left window while c-structures are displayed in the lower left window. These windows scroll up as new information is displayed at the bottom. By watching successive structures appear in the windows, one gets an impression of the progress of the generation process over time. In the lower right window, a summary of actions is presented.

The examples will involve the generation of (6a) and (6b) and are provided to make clear that the timing of input to the Formulator can influence the assignment of left-to-

right order of constituents in the c-structure. The input of the first simulation consists of the following LISP expression:

```
(DEFUN S-TEST ()
  "Generation of: 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
  ))
```

This input first creates three empty linguistic signs and then sends a number of commands to the Formulator to further formulate and relate these signs. The `DEFINE-FEATURES` commands simply assign the features to the linguistic sign. More substantial commands are three involving `FORMULATE` and two involving `DEFINE-CASE`. Each command immediately spawns an independent process which runs in parallel to everything else going on in the system. However, the commands in this simulation 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. With such a simulated 'slow' Conceptualizer, the generator will profit more from its incremental mode of generation.

Figure 6 presents three snapshots of the generation process. Due to the large time gap between the inputs, the sign for the concept *Otto* 'stays ahead' of the sign for *apple* and occupies a position earlier in the sentence. This results in a c-structure corresponding to sentence (6a). The left-to-right order of the NPs reflects the order of the conceptual fragments in the input.

a.
<Insert Figure 6a here>

b.
<Insert Figure 6b here>

c.
<Insert Figure 6c here>

Figure 6

Three snapshots of the generation of example (6a)

In a different simulation run, the SLEEP commands in the input were removed. Consequently, the computation of parallel encoding processes overlap more, and the constituents of the sentence therefore have a higher chance of occupying alternative left-to-right positions in the c-structure. In other words, left-to-right order in the sentence will have less chance to correspond to the order of the conceptual fragments in the input and will therefore be less predictable. This is illustrated in Figure 7. The trace displayed in the lower right window shows how closely spaced the conceptual messages are (their traces are each preceded by the character '>'; compare Figure 6). The sign for *Otto* happens to be 'overtaken' by that for *apple* and ends up in holder slot 3 of the S. This results in a c-structure corresponding to sentence (6b).

a.
<Insert Figure 7a here>

b.
<Insert Figure 7b here>

c.
<Insert Figure 7c here>

Figure 7

Three snapshots of the generation of example (6b)

4 PARALLEL UNIFICATION IN AN OBJECT-ORIENTED SEGMENT GRAMMAR

4.1 Unification in an object-oriented paradigm

We will now turn our attention to some representational and computational aspects of the proposed architecture. In IPF, the SG formalism is joined with object-oriented representation in CommonORBIT, an extension of Common LISP (De Smedt, 1989, 1987). Nodes in the syntactic structure are implemented as active computational objects³. Since they are feature structures, the *aspects* (or *slots*) of an object are interpreted as features. For instance, the following object definition is equivalent to feature structure (6):

```
(A FEATURE-OBJECT
  (CATEGORY 'NP)
  (PLURAL '-)
  (NOMINATIVE '+))
```

(9) [category = ,NP,plural = ,-,nominative = ,+]

The value of a feature need not be atomic. In order to facilitate feature sharing, features can themselves be represented as objects which can also be unified (see Section 2.3). Features are the simplest objects, with just two aspects, one for the name, and one for the value. Binary features are features with just two possible values.

```
(DEFOBJECT SYNTACTIC-FEATURE
  FEATURE-OBJECT
  (NAME)
  (VALUE))
```

```
(DEFOBJECT BINARY-FEATURE
  FEATURE
  (VALUE '(+ -)))
```

```
(DEFOBJECT PLURAL-FEATURE
  BINARY-FEATURE
  (NAME 'PLURAL))
```

Using such features, feature structure (9) can be redefined as:

```
(A FEATURE-OBJECT
  (CATEGORY 'NP)
  (PLURAL (A PLURAL-FEATURE (VALUE '-)))
  (NOMINATIVE (A NOMINATIVE-FEATURE (VALUE '+))))
```

Now such a definition is more verbose, but fortunately we can use inheritance to define features structures as combinations of other ones. ‘Mixins’ may be defined to hold more specific feature values. For instance, the following object *singular* contains only the value ‘-’ for the feature *plural*:

```
(DEFOBJECT SINGULAR
  (PLURAL (A PLURAL-FEATURE
            (VALUE '-))))
```

Using such mixins as abbreviations, the CommonORBIT definition of a feature structure can be much more concise. The following definition again describes feature structure (9). Figure 8 shows part of the hierarchy which is relevant to this definition.

```
(A SINGULAR NOMINATIVE NP)
```

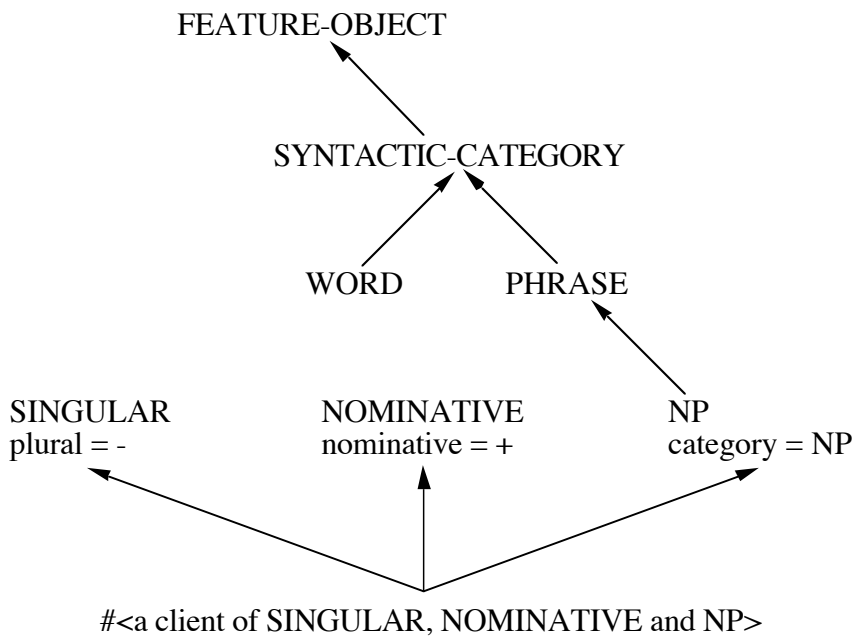


Figure 8
Example hierarchy of objects as feature structures

If feature structures are represented as objects, how is their unification defined? CommonORBIT provides a unification operation, called `UNIFY`, which unifies two objects by merging all the information contained in both objects into one. This operation succeeds if the values of all these features in both objects match in one of the following ways:

- 1 If the values are both objects, then the objects are unified. The features match if this unification succeeds; the unification becomes the new feature value. Thus, unification is recursive⁴.

- 2 Otherwise, the values are interpreted as sets (by coercing to lists) and the intersection is computed. However, the value `UNDEFINED` matches anything. The features match if the result of the intersection is non-`NIL`; the intersection becomes the new feature value.

If unification succeeds, the two objects are merged into one and the new feature values are stored into this one object, as well as all other information which was present in both objects. If unification fails, the result is `UNDEFINED`.

4.2 Locking shared resources

In SG, unification is a local operation on nodes in syntactic structures, not on syntactic structures themselves. This locality allows unification to be distributed in the sense that unifications can be carried out simultaneously on different parts of the structure. Parallel local unifications are possible because each node in an f-structure contains all necessary features to allow or disallow unification with other nodes.

At the same time, we must prevent several parallel unifications to operate on one and the same node, lest incompatible operations by different processes cause an inconsistent state. Unification in SG is thus a *critical section* (cf. Ben-Ari, 1982) in which certain shared resources are to be protected from manipulation by other processes. The shared resources in this case consist of the nodes involved in the unification and of their features. An example of a simultaneous unification of a node with two others is

depicted in Figure 9. The S nodes in this example may only be involved in one unification at the same time.

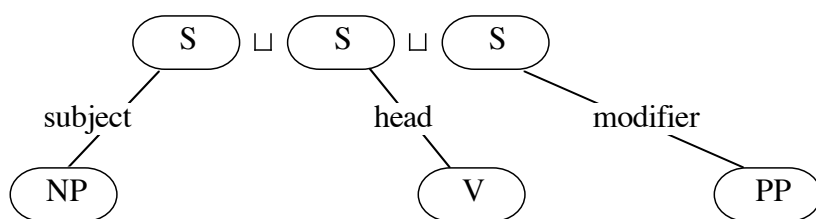


Figure 9

Two simultaneous unifications: nodes are shared resources

Even if two unifications involve a different pair of nodes, some features of one pair may still be shared with the other pair; hence these features must also be protected during unification. Suppose, for instance, that the unifications in Figure 10 happen in parallel. While the features shared by the root and the foot of the S-subject-NP segment are involved in one unification process, the other unification process might change them ‘behind its back’.

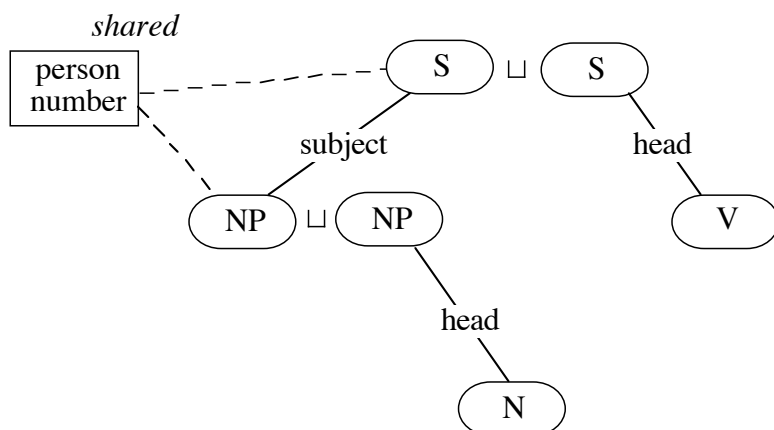


Figure 10

Two simultaneous unifications: features are shared resources

If we want to allow unification to take place in a distributed fashion, we must take care that the shared resources involved in one unification operation are temporarily made inaccessible to other processes, so that other processes cannot interfere with the delicate checking and changing of features. The simplest solution would be to make unification an ‘atomic’ operation, which means that other processes cannot run at all during its execution. However, a system based almost completely on unification would then hardly be parallel, since little is performed outside the unifications. A more feasible solution is to *lock* the objects involved in a unification so that they cannot be unified in other processes. Naturally, other processes can continue to run; they must wait only if they want to access an object locked by another process.

In an object-oriented language, a lock can easily be associated with each object. This is implemented as an aspect which contains a list with a value representing the state of the lock:

```

(DEFOBJECT FEATURE-OBJECT
  (LOCK
    :IF-NEEDED (SELF)
    (LIST NIL))
  (UNIFY
    :FUNCTION (SELF OTHER &KEY (FEATURES (UNION (FEATURES SELF)
                                                    (FEATURES OTHER))))
    (LOCKING-FEATURE-OBJECTS (SELF OTHER)
      (UNIFY-OBJECTS SELF OTHER :FEATURES FEATURES))))

```

Any process wishing to access an object first checks the lock of the object. If it contains a non-NIL value, the object is in a locked state and the process has to wait. If it is NIL, the object is free. The process then inserts its own identity in the lock to signal that the object is busy and proceeds. Unification, as defined for FEATURE-OBJECT, is ‘wrapped’ in a macro LOCKING-FEATURE-OBJECTS which locks the objects and their features in this way before it proceeds.

It is clear that checking all locks and setting them must be performed as one atomic operation, lest another process grabs a lock in between, potentially causing a deadlock situation. The macro LOCKING-FEATURE-OBJECTS was written to check and set the locks of two given objects and their features in one atomic operation by disabling process scheduling during execution of its body.

5 CONCLUSION AND DISCUSSION OF RELATED RESEARCH

Human speakers seem to produce speech in an incremental and parallel fashion. IPF is a model of incremental sentence formulation where conceptual fragments are processed simultaneously and independently of each other. SG, the grammar formalism used in IPF, provides a maximal incremental mode of generation by allowing the syntactic structure to be built by individual branches. SG furthermore distributes knowledge so that the unification process itself can be distributed. This allows the Formulator to take advantage of a relatively coarse-grained parallel architecture. Concurrent programming techniques are used to allow the simultaneous execution of several unifications in one syntactic structure.

IPF is a further development of Incremental Procedural Grammar (IPG; Kempen and Hoenkamp, 1987). It introduces a clean separation between the grammar formalism and the formulation process in IPG theory. Because the new model is unification-based, it allows the expansion of the syntactic structure in upward as well as downward direction. Furthermore, the parallelism in the Formulator which Kempen and Hoenkamp suggested is worked out in considerable detail and implemented in the multi-processing environment of the Symbolics Lisp machine.

A kind of distributed unification similar to the one presented here seems to be possible in Feature-structures based Tree Adjoining Grammar (FTAG: Vijay-Shanker & Joshi 1988), which uses feature unification as a general mechanism to specify restrictions on tree adjoining. Vijay-Shanker and Joshi claim that this formalism, which allows co-occurrence of features to be stated within elementary trees, can be extended to allow simultaneous adjoining operations in distinct nodes of an elementary tree. However, FTAG is limited by the fact that only complete trees can be adjoined; it does not allow the incremental addition of sister nodes and thus allows incrementation in a less fine-grained way.

In contrast to the system proposed here, which applies unification locally, most other unification-based formalisms involve arbitrarily large functional descriptions, containing many alternations and arbitrarily distant paths. Although parallel parsing algorithms have been proposed for a class of unification grammars, for instance by Haas (1987), such proposals involve a parallel exploration of alternations rather than the simultaneous addition of branches in one syntactic structure.

Other, subsymbolic models have attempted to exploit parallelism by means of a connectionist architecture. Experiments such as the connectionist phrase generation by Kukich (1987) have yielded some success. However, these models have so far not shown a clear sense of a syntactic structure and are therefore—from a linguistic viewpoint—unsatisfactory. Elman (1989) has reported the emergence of structured representations in connectionist networks, but it remains to be seen whether his claims apply to the generation of non-trivial syntactic structures from conceptual information.

Most natural language generation systems in the fields of AI and Computational Linguistics have not been designed for the simulation of spontaneous speech but for the construction of carefully planned sentences and texts. Hence, it will not be surprising that in most systems the conceptual and lexico-syntactic stages are ordered strictly serially for a complete sentence. Still, some attention has been given to incremental generation in at least four other systems: MUMBLE, KAMP, Φ DMDIALOG, and SOCCER.

In MUMBLE (McDonald & Pustejovsky, 1985a), a *conceptual planner* (Conceptualizer) and a *linguistic module* (Formulator) call each other recursively. The conceptual planner determines the salience of the concepts to be uttered and sends a stream of propositions to the linguistic module. The latter finds where the surface structure of the sentence can be extended with a formulation of the new proposition, finds appropriate words and phrases in the lexicon, and extends the output word stream. Predefined *attachment points* in the surface structure determine where and how it can be extended. These extensions seem to be limited to expansions from the top of the syntactic structure downward and coordinations: “another adjective added to a certain noun phrase, a temporal adjunct added to a clause...” (1985a:189).

In the KAMP system (Appelt, 1983, 1985), there is a component called TELEGRAM which couples the processes of conceptualization and formulation in an incremental architecture based on *unification*. In Functional Unification Grammar (FUG; Kay, 1979), a sentence can be produced by the *unification* of two *functional descriptions* (FDs). One of these represents a partially specified utterance and possibly includes some conceptual information. The other one is the grammar of the language. Instead of doing a single unification between a completely specified FD for the sentence as a whole (the ‘text FD’) and the grammar (the ‘grammar FD’), the TELEGRAM planner works by gradual refinement. Initially, a high-level, incomplete text FD is produced by the planner and unified with the grammar FD. Subsequent planning produces more FDs, which are unified with the grammar FD and incorporated into the text FD. However, the system plans hierarchically, and the resulting enrichments of the text FD seem to be limited to downward expansion and possibly coordination.

In a simultaneous interpretation task, an incremental mode of generation as well as parsing is evidently mandatory. Kitano (1990) presents Φ DMDIALOG, a system for interpreting telephone conversations based on a parallel marker-passing scheme. Parsing and generation are concurrent in the sense that the generation process does not wait until the entire parse is completed, so that an utterance is translated in a piecemeal

way. Lexical selection and partial generation are conducted while parsing is in progress. In addition to this intercomponent parallelism, the system also seems to exploit a kind of intracomponent parallelism, where a consistent syntactic structure is maintained by constraint satisfaction.

Another application area for incremental generation is the simultaneous description of real world image sequences. André, Herzog and Rist (1988) describe SOCCER, a system which visually interprets a sequence of soccer scenes and simultaneously generates German natural language descriptions of the ongoing game. SOCCER is capable of generating incrementally: formulation of a sentence can start when an action (i.e. a main verb) and its agent are known, and the partial sentence can be extended as soon as other participants are recognized. André (1988) identifies some important problems for incremental generation and recognizes the need for self-corrections. SOCCER does not seem to exploit intracomponent parallelism, nor does it allow upward expansions of the syntactic structure.

Several natural language generation systems give a detailed account for variations of word order and lexical choice, for instance Danlos (1987) and Hovy (1987). However, they do so from a stylistic perspective in the context of written text generation. It is questionable whether the strategies they propose are transferable to the task of producing spontaneous speech. They lack an incremental mode of generation and seem to be based on very conscious reasoning about pragmatic goals. Since spontaneous speech is usually produced under severe time pressure, it is likely that timing factors will affect the form of the utterance more than conscious choices.

Some evidence for the importance of time in the generation of spontaneous speech comes from psycholinguistic research. Recently, Schriefers and Pechmann (1988; also Pechmann 1987; 1989) developed an experimental paradigm based on a *referential communication task*. Subjects were asked to give a verbal description of a given target object to distinguish it from a number of other, simultaneously displayed objects. The descriptions collected in these experiments were in the form of NPs possibly including a shape category and the features color and size, for instance WHITE BIG TRIANGLE. Inspection of the resulting descriptions revealed irregularities which can be explained by non-optimal planning, in particular that speakers indeed start formulating before they have fully determined some set of distinguishing visual features. In addition to overspecified—as opposed to minimally specified—referential descriptions, Pechmann (1989) also reports non-standard order of the features color and size as prenominal adjectives—an indication that speakers favor an order depending on the piecemeal conceptual availability of the features. These results provide strong evidence that people produce incrementally in a fine-grained way, i.e., in small units corresponding to single phrases and even words, although effects due to larger chunks are also found.

References

- Adriaens, G. (1986). Word Expert Parsing revised and applied to Dutch. In: *ECAI-86. Proceedings of the 7th European Conference on Artificial Intelligence*, Brighton (pp. 222-235).
- Allport, A., Antonis, B. & Reynolds, P. (1972). On the division of attention: a disproof of the single channel hypothesis. *Quarterly journal of experimental psychology* 24, 225-235.
- André, E. (1988). *Generierung natürlchsprachlicher Äußerungen zur simultanen Beschreibung von zeitveränderlichen Szenen: Das System SOCCER*. (Memo Nr. 26), Saarbrücken: Project VITRA, Universität des Saarlandes.

- André, E., Herzog, G. & Rist, Th. (1988). On the simultaneous interpretation of real world image sequences and their natural language description: the system SOCCER. Y. Kodratoff, (Ed.) *ECAI 88: Proceedings of the 8th European Conference on Artificial Intelligence* (pp. 449-454). London: Pitman.
- Appelt, D. (1985). *Planning English sentences*. Cambridge: Cambridge University Press.
- Ben-Ari, M. (1982). *Principles of concurrent programming*. Englewood Cliffs: Prentice-Hall.
- Bock, J.K. & 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. G. Kempen (Ed.) *Natural language generation: New results in Artificial Intelligence, psychology and linguistics* (pp. 351-363). Dordrecht/Boston: Nijhoff (Kluwer Academic Publishers).
- Bresnan, J. (ed.) (1982). *The mental representation of grammatical relations*. Cambridge (MA): MIT Press.
- Chomsky, N. (1959). Review of Skinner (1957). *Language* 35, 26-58.
- Danlos, L. (1987). *The linguistic basis of text generation*. Cambridge: Cambridge University Press.
- De Smedt, K. (1987). Object-oriented programming in Flavors and CommonORBIT. In: Hawley, R. (ed.) *Artificial Intelligence Programming Environments* (pp. 157-176). Chichester: Ellis Horwood.
- 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. (1990a). *Incremental sentence generation: a computer model of grammatical encoding*. NICI Technical Report 90-01, Nijmegen Institute for Cognition Research and Information Technology, Nijmegen, The Netherlands.
- De Smedt, K. (1990b). IPF: An incremental parallel formulator. R. Dale, C. Mellish, & M. Zock (Eds.) *Current research in natural language generation* (pp. 167-192). London: Academic Press.
- De Smedt, K. & 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* (pp. 365-376). Dordrecht/Boston: Martinus Nijhoff Publishers (Kluwer Academic Publishers).
- De Smedt, K. & Kempen, G. (1991). Segment Grammar: a formalism for incremental sentence generation. C.L. Paris, W.R. Swartout, & W.C. Mann (Eds.) *Natural language generation in Artificial Intelligence and Computational Linguistics* (pp. 329-349). Dordrecht/Boston: Kluwer Academic Publishers.
- Dowty, D.R. (1990). Toward a minimalist theory of syntactic structure. In: *Proceedings of the Symposium on Discontinuous Constituency*, Tilburg, 25-27 January 1990 (pp. 33-72). ITK, Tilburg University, The Netherlands.
- Elman, J.L. (1989). Structured representations and connectionist models. In: *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society, 16-18 August 1989, Ann Arbor* (pp. 17-25). Hillsdale: Lawrence Erlbaum.
- Feldman, J.A. and Ballard, D.H. (1982). Connectionist models and their properties. *Cognitive Science* 6, 205-254.
- Fodor, J.A. (1983). *The modularity of mind*. Cambridge, MA: MIT Press.

- Garrett, M. (1975). The analysis of sentence production. In: Bower, G. (ed.) *The psychology of learning and motivation (Vol. 9)*. New York: Academic Press.
- Garrett, M. (1980). Levels of processing in sentence production. In: Butterworth, B. (ed.) *Language production. Vol. 1: Speech and Talk*. New York: Academic Press.
- Haas, A. (1987). Parallel parsing for Unification Grammars. In: *Proceedings of the Tenth IJCAI* (pp. 615-618). Los Altos: Morgan Kaufmann.
- Hoenkamp, E. (1980). Spontaneous speech as a feedback process. In: *AISB-80. Proceedings of the AISB Conference on Artificial Intelligence*, Amsterdam.
- Hoenkamp, E. (1983). *Een computermodel van de spreker: psychologische en linguïstische aspecten*. Ph.D. Dissertation, University of Nijmegen, The Netherlands.
- Hovy, E. (1987). Some pragmatic decision criteria in generation. In: Kempen, G. (ed.) *Natural language generation: New results in Artificial Intelligence, psychology and linguistics* (pp. 3-17). Dordrecht/Boston: Martinus Nijhoff Publishers (Kluwer Academic Publishers).
- 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* (pp. 233-252). Dordrecht / Boston: Martinus Nijhoff Publishers (Kluwer Academic Publishers).
- Karttunen, L. (1984). Features and values. In: *Proceedings of Coling-84* (pp. 28-33). Association for Computational Linguistics.
- Kay, M. (1979). Functional Grammar. In: *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistic Society* (pp. 142-158). Berkeley, CA: Berkeley Linguistic Society.
- Kempen, G. (1987). A framework for incremental syntactic tree formation. In: *Proceedings of the Tenth IJCAI* (pp. 655-660). Los Altos: Morgan Kaufmann.
- Kempen, G. and Hoenkamp, E. (1987). An incremental procedural grammar for sentence formulation. *Cognitive Science*, 11, 201-258.
- Kitano, Hiroaki. (1990). Parallel incremental sentence production for a model of simultaneous interpretation. R. Dale, C. Mellish, & M. Zock (Eds.) *Current research in natural language generation* (pp. 321-351). London: Academic Press.
- Kolk, H. (1987). A theory of grammatical impairment in aphasia. G. Kempen (Ed.) *Natural language generation: New results in Artificial Intelligence, psychology and linguistics* (pp. 377-391). Dordrecht/Boston: Nijhoff (Kluwer Academic Publishers).
- Kukich, K. (1987). Where do phrases come from: some preliminary experiments in connectionist phrase generation. In: Kempen, G. (ed.) *Natural language generation: New results in Artificial Intelligence, psychology and linguistics* (pp. 405-421). Dordrecht/Boston: Martinus Nijhoff Publishers.
- Levelt, W. (1989). *Speaking: from intention to articulation*. Cambridge (MA): MIT Press.
- McDonald, D., & Pustejovsky, J. (1985). A computational theory of prose style for natural language generation. *Proceedings of the 2nd Conference of the European Chapter of the ACL, Geneva*. Morristown, NJ: Association for Computational Linguistics.

- Pechmann, Th. (1987). *Effects of incremental speech production on the syntax and content of noun phrases*. Report 20, Arbeiten der Fachrichtung Psychologie, Universität des Saarlandes, Saarbrücken.
- Pechmann, Th. (1989). Incremental speech production and referential overspecification. *Linguistics* 27, 89-110.
- Schabes, Y., Abeillé, A. & Joshi, A.K. (1988). Parsing strategies with ‘lexicalized’ grammars: application to Tree Adjoining Grammars. In: *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest.
- Schriefers, H. & Pechmann, Th. (1988). Incremental production of referential noun-phrases by human speakers. In: Zock, M. & Sabah, G. (eds.) *Advances in natural language generation, Vol. 1* (pp. 172-179). London: Pinter Publishers.
- Shieber, S.M. (1986). *An introduction to unification-based approaches to grammar*. CSLI Lecture Notes No. 4. Stanford: CSLI.
- Vijay-Shanker, K. & Joshi, A. (1988). Feature Structures Based Tree Adjoining Grammars. In: *Coling '88: Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, 22-27 August 1988. Association for Computational Linguistics.

¹ This processing assumption could partly explain the frequent extraposition of long and complicated phrases and clauses to the end of the sentence, for instance “heavy NP shift” (Dowty, 1990).

² The following is a concise formal definition of SG. A *Segment Grammar* G for a language L_G is a septuple $G=(N,T,S,P,F,W,O)$ with N a set of non-terminal symbols, T a set of terminal symbols, S a set of segments, P a set of phonetic symbols, F a set of feature symbols, W a set of feature value symbols, and O a set of grammatical function symbols.

For a Segment Grammar G , *f-structures* are connected directed acyclic graphs (DAGs) defined by the quintuple (V, E, F_W, F_L, F_O) where V is a set of nodes, E a set of arcs: $E \subseteq V \times V$, F_W a partial function: $F \times V \rightarrow \wp(W)$, F_L a labeling function: $V \rightarrow N \cup T$, and F_O a labelling function $E \rightarrow O$ (The symbol \wp denotes the powerset). Sets of feature values, as allowed by the function F_W , represent disjunctive values. The set of *f-structures* in G is defined as $FG = S \cup \{y \mid \exists y', y'' \in FG: y \in U(y', y'')\}$ where U is the universal unification function: $FG \times FG \rightarrow \wp(FG)$ which derives new *f-structures* by unifying a node of one *f-structure* with a node of another.

An *f-structure* is well-formed if it observes the principles of completeness and coherence stated in (a) and (b). These principles refer to valence specifications in the lexicon.

(a) An *f-structure* is *coherent* if all the grammatical functions of each node are governable by that node, i.e., if $\{\forall (v,x) \in E \mid F_O(v,x) \in O_v\}$, where O_v is the set of possible grammatical functions for v .

(b) An *f-structure* is *complete* if each non-terminal node contains all its obligatory governable grammatical functions, i.e., if $\{\forall v \in V \mid \exists (v,x) \in E, F_O(v,x) \in O_v\}$, where O_v is the set of obligatory grammatical functions for v .

Each *segment* $s \in S$ is an *f-structure* with $V = \{r,f\}$ and $E = \{(r,f)\}$ where r is called the root node and f the foot node. The subset of segments $\{s \in S \mid F_L(f) \in T\}$ is called the set of *lexical segments*.

For a Segment Grammar G , *c-structures* consist of a quadruple $(V, F_M, <, F_L)$ where V is a set of nodes $\subseteq V$, F_M a mother function: $V \rightarrow V \cup \{\perp\}$, $<$ a well-ordered partial

precedence relation: $< \subset V \times V$, and FL a labeling function: $V \rightarrow N \cup T$. While for each well-formed f-structure, there is a c-structure such that: $V_c \subseteq V_f$, there is no isomorphic mapping from c-structures to f-structures. The c-structures in G are derivable from the f-structures by means of the destination and linearization processes which are described elsewhere (De Smedt, 1990a).

For a Segment Grammar G , *phonetic strings* are structures $\in P^*$. The phonetic strings in LG are computed by the Phonological Encoder from the sequences of terminal nodes of all possible c-structures in G .

³ For an object-oriented approach to parsing, see for instance Adriaens (1986) and Yonezawa and Ohsawa (this volume).

⁴ The full recursive power of unification in CommonORBIT is not strictly necessary in SG , where the feature values of syntactic features are atomic.