Chapter 13

Segment Grammar: a Formalism for Incremental Sentence Generation

Koenraad De Smedt
and Gerard Kempen

Abstract: Incremental sentence generation imposes special constraints on the representation of the grammar and the design of the formulator (the module which is responsible for constructing the syntactic and morphological structure). In the model of natural speech production presented here, a formalism called Segment Grammar is used for the representation of linguistic knowledge. We give a definition of this formalism and present a formulator design which relies on it. Next, we present an object- oriented implementation of Segment Grammar. Finally, we compare Segment Grammar with other formalisms.

13.1 INTRODUCTION

Natural speech is often produced in a piecemeal fashion: speakers start to articulate a sentence before the syntactic structure, or even the meaning content of that sentence has been fully determined. Under the assumption that the human language processing apparatus is capable of carrying out different tasks in parallel, the speaker may already utter the first fragments of a sentence while simultaneously processing more content to be incorporated in the sentence. This mode of generation, which we call *incremental* generation, seems to serve a system whose major purpose is to articulate speech without long pauses, even if it is imperfect or incomplete.

Once a speaker has started to utter a sentence, the *formulator* (i.e. the module which is responsible for the syntactic and morphological structure) will try to complete the sentence in a maximally grammatical way and will try to avoid making revisions. However, a speaker who starts a sentence without knowing the entire content in detail forces the formulator to operate with incomplete knowledge. In an incremental mode of production, the formulator will sometimes make a choice which turns out to be incompatible with new conceptual input at a later moment. De Smedt and Kempen (1987) discuss how various conceptual changes may affect the structure of the utterance which is under construction.

We are currently designing a computational model of a formulator which operates under the special constraints imposed by incremental generation. In this paper we discuss some aspects of that formulator and of the grammatical knowledge it uses. In particular, we argue that, regardless of their formal generative properties, not all grammar formalisms are equally suited to support incremental generation. Consider the following requirements put forward by Kempen (1987):

- Three kinds of syntactic incrementation are distinguished: *upward expansion, downward expansion,* and *insertion.* A grammar should allow all three varieties (although insertion could be treated as a special case of combined upward and downward expansion).

- Lexical increments can be small, even a single word. Therefore the syntactic tree should be able to grow by individual branches. This implies that all daughters of a node in the tree should not necessarily be generated at once: the formalism should be able to add sister nodes incrementally.

- There is no reason to assume that the chronological order in which branches are attached to the syntactic tree corresponds to their linear precedence in the resulting utterance. Hence the grammar should separate knowledge about immediate dominance from knowledge about linear precedence.

In order to satisfy these requirements, Kempen proposes *Incremental Grammar (IG),* a new formalism for the representation of grammatical knowledge. It is especially suited to - but not restricted to - incremental generation. In order to clearly distinguish between the grammar formalism and the processing model, we will rename the grammar formalism *Segment Grammar (SG)* and we will refer to the processing model as the *incremental formulator.* After a definition of SG, we discuss how SG representations are used in our incremental formulator. Then we will present an implementation of SG using object-oriented programming techniques, compare it with other formalisms, and point out its main advantages.

13.2 SEGMENT GRAMMAR

Somewhat like a lexical-functional grammar (LFG; Kaplan and Bresnan, 1982), an SG assigns two distinct descriptions to every sentence of the language which it generates. The constituent structure (or *c-structure)* of a sentence is a conventional phrase structure (PS), which is an ordered tree-shaped graph. It indicates the 'surface' grouping and ordering of words and phrases in a sentence. The functional structure (or *f-structure)* provides a more detailed representation of 'functional' relationships between words and phrases, as traditionally expressed by notions like subject, direct object, etc. The representation in f- structures also accounts for phenomena like agreement, and it does so by using features like number, gender, etc. Since SG is used for incremental processing, it assigns representations to partial sentences as well as to full ones.

When an SG is used for generation, semantic and discourse information is mapped into f-structures, which in turn are mapped into c-structures. C-structures are then subjected to morpho-phonological processing, producing phonetic strings which are eventually uttered as speech sounds. This overall process is depicted in Figure 13.1. We will now be concerned with the elements which constitute the grammar.

13.2.1 Formal Definition Of Segment Grammar

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.[1]

For a segment grammar G, f-structures are connected directed acyclic graphs defined by the quintuple (V, E, F_W, F_L, F_O) where V is a set of nodes, E a

[1] **It is questionable whether grammatical functions are strictly necessary in SG. This will be discussed later.**

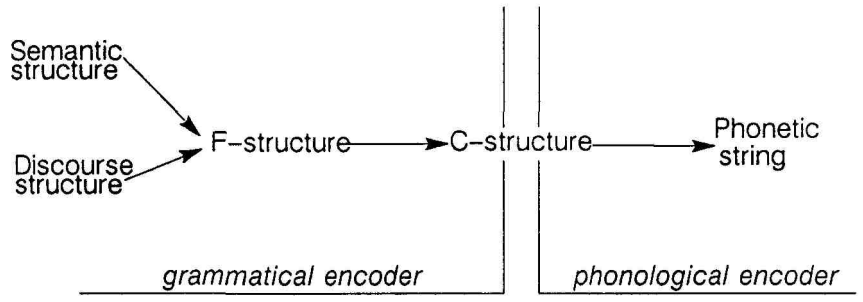

Figure 13.1: Subsequent linguistic descriptions during generation

set of arcs: $E \subseteq$ V x V, F_W a partial function: F x V $\rightarrow \wp$ W, F_L a labelling function: V $\rightarrow$ N $\cup$ T, and F_O a labelling function E $\rightarrow O^2$.[2] The set of f-structures in G is defined as $\Im_G$= S $\cup$ { y | $\exists$ y', y'' $\in \Im_G$: y $\in$ U(y',y'')} where U is the universal unification function: $\Im_G$ x $\Im_G \rightarrow \wp$ ($\Im_G$). The unification function is essentially the same as that proposed by Kay (1979).

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 | F_L(f) $\in$ T } is called the set of *lexical segments.*

For a segment grammar G, c-structures (PS trees) consist of a quadruple (V, F_M, <, F_L) where V is a set of nodes, F_M a mother function: V $\rightarrow$ V $\cup$ { $\perp$ }, < a well-ordered partial precedence relation: < $\subset$ V x V, and F_L a labelling function: V $\rightarrow$ N $\cup$ T. In contrast with LFG and other formalisms where c-structures are derived only by means of a context-free grammar, the c-structures in G are derived from the f-structures in G by means of the destination and linearization processes which are described later.

For a segment grammar G, phonetic strings are structures that are elements of the set P. The phonetic strings in L_G are the sequences of terminal nodes of all possible c-structures in G.

13.2.2 Informal Synopsis of Segment Grammar

Segments are the elementary building blocks of the grammar. They are graphs with two nodes: a root node and a foot node. Isolated segments are conventionally represented in vertical orientation with the root node, labeled with its

[2] $\wp$ denotes the powerset.

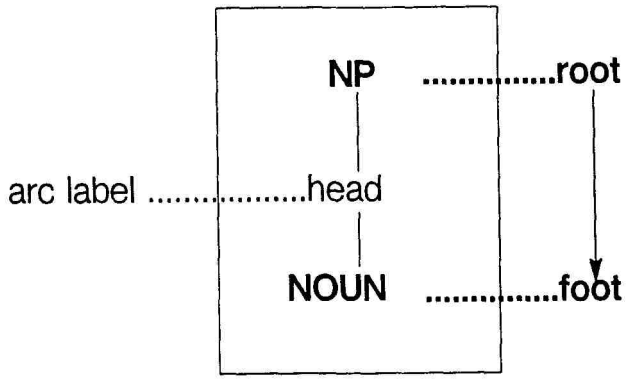

Figure 13.2: A syntactic segment

category, at the top, the foot node, labeled with its category, at the bottom, and an arc, represented as a vertically directed edge labeled with a grammatical function, between the nodes. An example is shown in Figure 13.2. In running text, segments are also written left-to-right (root-to-foot), e.g., S-SUBJECT-NP or NP-HEAD-NOUN.

Syntactic segments are the smallest possible f-structures and may therefore be considered as atomic units. Just like atoms in chemistry combine to form molecules, segments combine to form larger f- structures. These structures are unordered (they are sometimes called mobiles), since word order is assigned at a later stage. F-structures are graphs consisting of nodes labeled with syntactic categories (or with lexical items). C-structures are ordered graphs derived from f-structures by a process described later, and phonetic strings are the sequences of terminal nodes in c-structures.

One basic operation, *unification,* governs the composition of smaller f-structures into larger ones. By unifying two nodes belonging to different f-structures, the nodes may merge into one; thus a graph of interconnected segments is formed. The two basic variants of unification are *concatenation* (vertical composition by unifying a root and a foot) and *furcation* (horizontal composition by unifying two roots). E.g., two segments which are instances of S-SUBJECT-NP and of NP-HEAD-NOUN can be concatenated by unifying their NP nodes; two segments which are instances of NP-DETERMINER-ARTICLE and NP-HEAD-NOUN can be furcated, also by unification of their NP nodes. This is schematically represented in Figure 13.3.

To each node, a set of *features* may be attributed. For example, S nodes have a feature FINITE with '+' or '–' as possible values. If no values are explicitly specified,

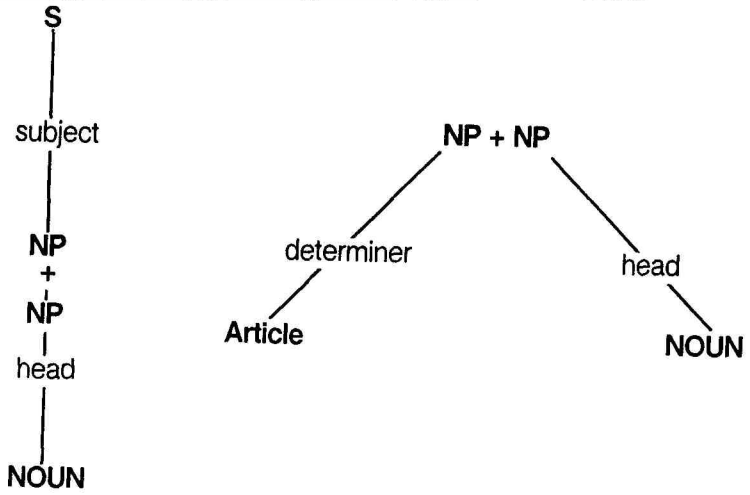

Figure 13.3: Concatenation (left) and furcation (right)

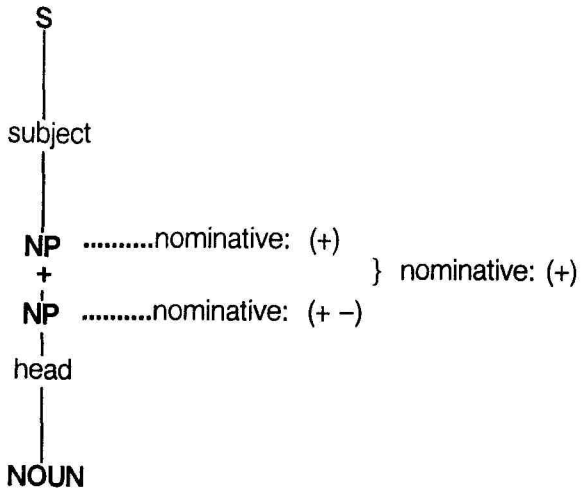

Figure 13.4: Feature unification

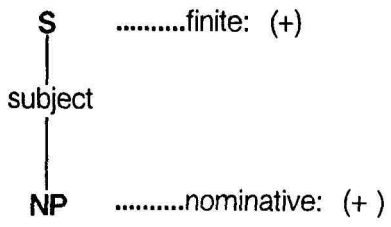

Figure 13.5: Co-occurrence of features in a segment

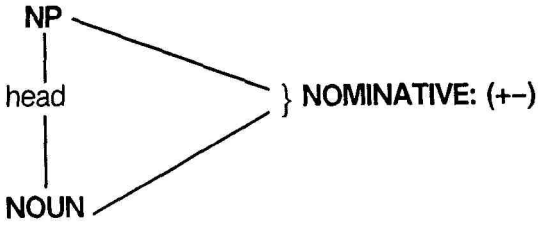

Figure 13.6: Feature sharing

then the feature has all possible values by default, in this case the set (+ –).[3] When two nodes are unified (in either concatenation or furcation), their features are also unified. This process, illustrated in Figure 13.4, is essentially the same as feature unification in other unification-based formalisms (Karttunen, 1984). It consists of computing the union of all features in both nodes, and for each feature the intersection of the values in both nodes.

The co-occurrence of feature restrictions on the root of a segment with feature restrictions on the foot may be used to model syntactic constraints. E.g., the constraint that "if the subject of a finite sentence is an NP, it must be nominative" is modeled by specifying a feature FINITE with value '+' on the root of a S-SUBJECT-NP segment, and a feature NOMINATIVE with value '+' on the foot, as depicted in Figure 13.5. In addition, the root and the foot of a segment may *share* certain features. For example, NOMINATIVE is shared in the NP-HEAD-NOUN segment as depicted in Figure 13.6.

The combination of feature sharing and unification amounts to 'feature transport'. By virtue of the sharing relationship in NP-HEAD-NOUN, the con-

[3] Sets are represented here in list notation.

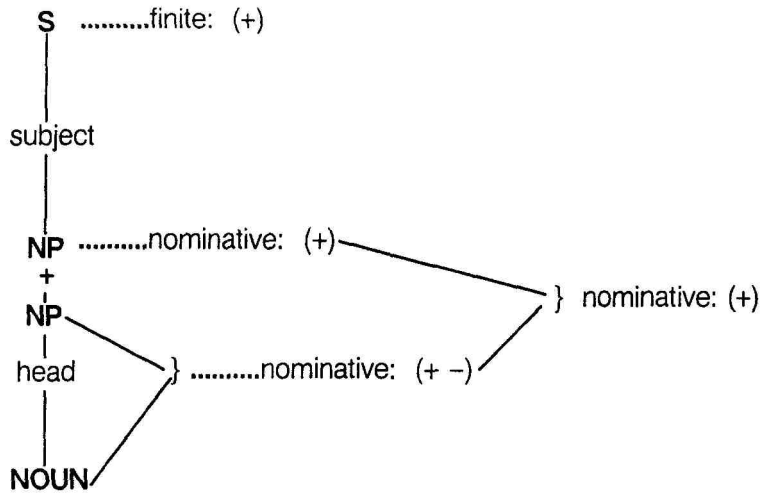

Figure 13.7: Unification of a shared feature

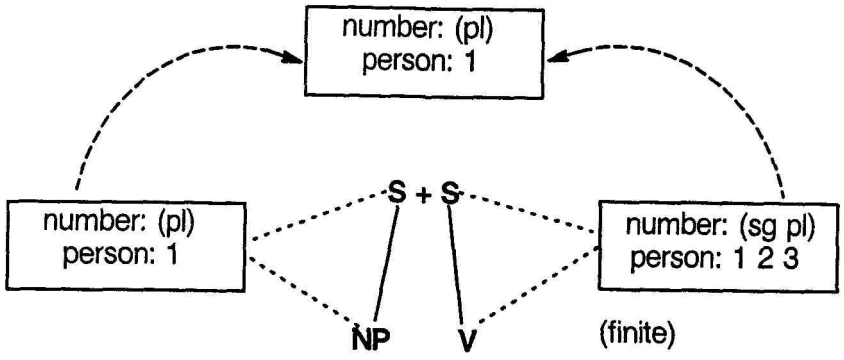

Figure 13.8: Agreement by means of feature sharing

catenation depicted in Figure 13.7 results in a feature change to the foot of the lower segment as well as to its root. Features are in fact not transported, but they are unified. Agreement can easily be modeled by feature sharing in concatenated and furcated segments. For example, 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 13.8.

The combination of root and foot in a segment is a declarative representation of a single immediate dominance (ID) relationship. Restrictions on sisterhood are encoded in a procedural way. The addition of function words, e.g., determiners and auxiliaries, is governed by a *functorization* process during formulation, which attaches grammatical combinations of function words to a phrase in a procedure-based way. The addition of non-function-words, i.e., constituents which are in a case relation to the phrase, is driven by the conceptual module and is restricted by valency information in lexical segments. E.g., the possible addition of a direct object in a clause is specified in lexical segments of the type S-HEAD-V.

It is unclear whether there is a need for an explicit specification of additional, more global restrictions on sisterhood, e.g., the restriction that only one direct object may occur in a clause. We assume that conceptual input to the formulator cannot normally give rise to such circumstances, because there will be no contradictory information in case relationships. Hence, these restrictions are seen as emerging properties of the formulation process rather than defining properties of the grammar. If there is evidence that this kind of restrictions must be explicitly defined in the grammar, then it remains possible to specify *ad hoc* restrictions on unification based on the grammatical function labels in segments. If not, then the notion of grammatical function is disposable in SG.

Linear precedence (LP) is encoded by assigning to each foot node a POSITIONS feature which contains a list of possible positions that the node may occupy in its destination, i.e., (by default) its mother node in the c-structure. The assignment of left-to-right order in c-structures will be further explained in the next section.

13.3 A LEXICALLY DRIVEN FORMULATOR DESIGN

We now turn to the question of how SG is used in an incremental formulator. The lexical origin of syntactic configurations is present in some form or other in many modern grammar theories. In Government-Binding (GB) theory, for example, the *Projection Principle* states that

> Representations at each syntactic level are projected from the lexicon, in that they observe the subcategorization properties of lexical items. (Chomsky, 1981)

Conceptual and lexical guidance is also argued for by Kempen and Hoenkamp (1987). Approaches which differ from this principle in that they assume the insertion of lexical material (i.e., content words) into a previously made syntactic structure, as done in early transformational grammar (Chomsky, 1965) but also in some recent psycholinguistic models (e.g., Dell, 1986) are in our opinion unrealistic for purposes of natural language generation.

On the one hand, syntactic tree formation is lexically guided in our theory, because the choice of lexical material clearly puts constraints on syntactic choices. For example, *to see* can take an infinitival object clause whereas to *know* cannot. It follows in our theory that the lexicon is responsible for the choice of any segments to be incorporated into the tree, e.g., the choice between S-OBJECT-INFINITIVAL-S and S-OBJECT-FINITE-S.

On the other hand, the opposite guidance also holds, especially in incremental sentence generation: the choice of lexical material is subject to categorial restrictions imposed by the partial tree which has been constructed so far. For example, *to dedicate* may take a NP as a direct object, but not a S. This categorial restriction will cause a nominal lexical entry to be preferred to a verbal one when the direct object is to be lexicalized.

It will not be surprising that the basic lexico-syntactic building block is modeled as a segment. A *lexical segment* is a segment where the foot is a word. Examples are NP-HEAD-CAKE (a nominal lemma) and S-HEAD-EAT (a verbal lemma). Because these segments always link a word to a phrase, the lexicon is essentially a *phrasal* lexicon. Information which is traditionally assigned to words, such as features and valency, is in our approach assigned to the roots of the lemma segments (the phrases) rather than their feet (the words), as schematically represented in Figure 13.9. Multi-word phrases are part of the lexicon in the form of ready-made furcations of segments, e.g., for *kick the bucket* in Figure 13.10. The lexicon is a set of *lemmas,* which are lexical entries consisting of one or more segments.

The lexical (pre-syntactic) stage in the formulator activates one or more lemmas on the basis of a conceptual fragment. It also assigns features with a non-lexical origin, such as PLURAL or DEFINITENESS, to those lemmas. A case relation gives rise to one or more non-lexical segments, such as S-DIRECT- OBJECT-NP.

The syntactic stage of the formulator tries to attach the lemmas and non-lexical segments to the tree by means of unification, which enforces the categorial and feature restrictions in the existing (partial) tree. F-structures are complete when the head and other obligatory segments (according to the valency information in the phrasal nodes) have been incorporated and when the *functorizatton* process, which causes the addition of function words such as determiners and auxiliaries, has taken place.

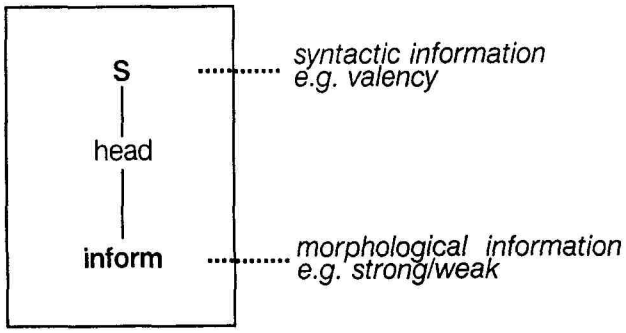

Figure 13.9: Distribution of information in lexical segments

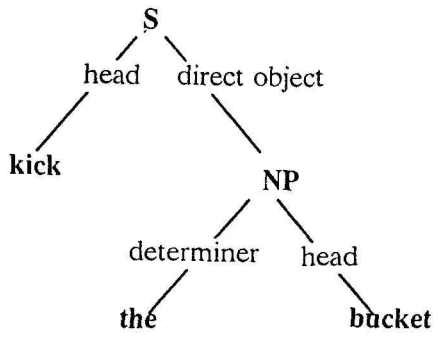

Figure 13.10: A multi-segment lemma

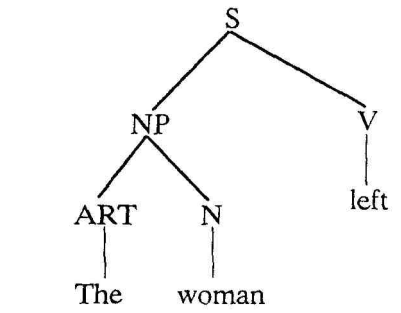

Figure 13.11: C-structure for "The woman left."

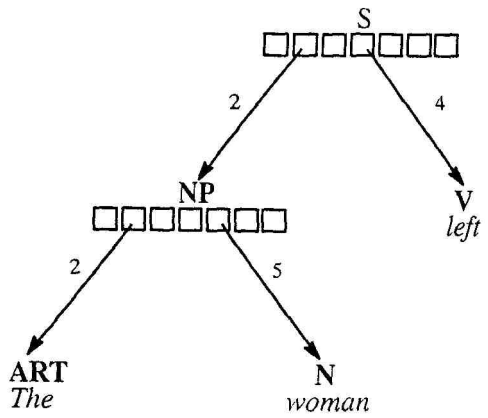

Figure 13.12: Diagram showing the holders for "The woman left."

F-structures, as constructed in this fashion, are unordered. The assignment of left-to-right positions to constituents is not simply an ordering imposed on the existing arcs of an f-structure, because dominance relations in the *c-structure* are not necessarily identical to those in the f-structure (e.g., in the case of discontinuous constituents). Therefore, a completely new, ordered structure - the c-structure, or surface tree - is constructed. The procedure which assigns left-to-right positions is incremental and works in a bottom-up fashion: the foot node of a segment is attached directly under its *destination,* which is normally the root of the segment. However, nodes may go to higher level destinations in situations like *clause union* and *WH-fronting,* which give rise to c-structures which are non-isomorphic to the corresponding f-structures (in particular, they may be flattened). We refer to Kempen and Hoenkamp (1987) for details. Figure 13.11 is an example of a c-structure.

Since f-structures as well as c-structures are constructed in a piecemeal fashion, it is natural to assign word order on a first-come, first-serve basis. For this scheme we use absolute rather than relative positions. With each phrase, a *holder* is associated, which is a vector of slots that can be filled by its constituents. Figure 13.12 shows an example of a constellation of holders and constituents for the c-structure in Figure 13.11.

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. E.g., in the grammar for Dutch it is specified that the foot of the S-SUBJECT-NP segment may go to absolute positions 1 or 3. When the foot of such a segment is to be assigned a position in the holder of its destination, it will first attempt to occupy position 1. If position 1 has already been occupied, it will attempt to go to position 3. A schematic overview of such a situation is given in Figure 13.13. 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 will be necessary.

13.4 AN OBJECT-ORIENTED IMPLEMENTATION OF SEGMENT GRAMMAR

An object-oriented version of SG has been implemented in CommonORBiT (De Smedt, 1987, 1989). This language (based on Common LISP) stems from the object-oriented and frame-based paradigms. *Objects* are basic computational units which represent physical or abstract entities in the problem domain. The properties of an object as well as the actions it may perform are defined in *aspects (slots)* associated with the object.

SG is implemented by uniformly representing all grammar concepts, such as nodes (phrases, words), features and syntactic segments as *objects.* Segments

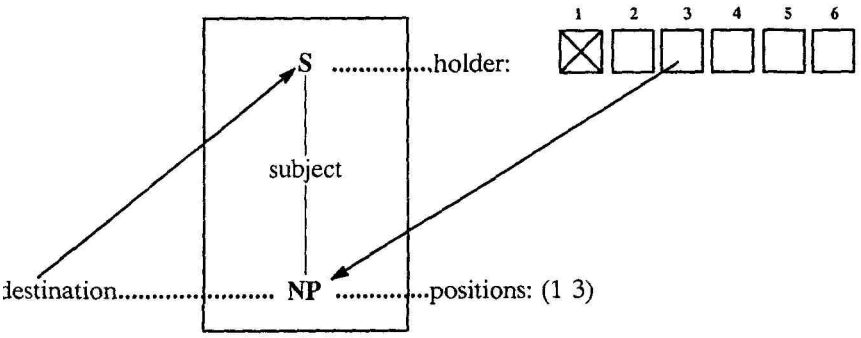

Figure 13.13: Destination and linearization processes: assign NP to the third slot in the holder of its destination

have aspects for the root, foot, arc label, and features which are shared between root and foot. The Commonsc orbit definition of some segments, including a lexical segment, is given below. Also given is a representation of the Dutch word *watermeloen* (watermelon).

```
(defobject np-head-noun
   syntactic-segment              ; delegate to prototypical segment
   (root (a np))
   (arc 'head)
   (foot (a noun))
   (agreement-features '(person plural gender nominative))
   ((foot positions) '(6))        ; word order possibilities
   )

(defobject watermeloen-segment ; a lexical segment
   np-head-noun
   (foot (a watermeloen-noun)))

(defobject watermeloen-noun    ; a word
   compound-noun
   (component-s (list (a water-noun)
                      (a meloen-noun))))
```

Object-oriented and frame-based formalisms typically allow the use of a specialization hierarchy in which specific objects *(clients)* may *delegate* requests for

information to other, more general, objects *(proxies)* in order to avoid redundancy. The use of such a hierarchy for linguistic concepts in the grammar as well as the lexicon has been advocated by De Smedt (1984). The prototypical object SYNTACTIC-SEGMENT, which acts as a proxy for specific segments such as NP-HEAD-NOUN, contains general knowledge about segments:

```
(defobject syntactic-segment
    feature-object              ; an object with features
    (root (a syntactic-category))
    (arc)
    (foot (a syntactic-category))
    (agreement-features nil)    ; default = no sharing
    (agree
      :function (self)
      (share-features (root self) (foot self)
                      :features (agreement-features self)))
    (initialize :if-needed #'agree)
    (concatenate-segments
      :function (high low)
      ;; merge the root of the low one with the foot of
      ;; the high one
      (unify (foot (initialize high)) (root (initialize low))))
    (furcate-segments
      :function (left right)
      ;; merge the roots of both segments
      (unify (root (initialize left)) (root (initialize right)))
))
```

More specific knowledge is distributed among the specific segments. For example, in NP-HEAD-NOUN, categories restricting the root and foot slots are given.

Syntactic categories are also defined as CommonoRBiT objects. Phrasal categories, parts of speech and lexical entries are represented in the same delegation hierarchy. Consequently, the grammar and the lexicon are on a continuum: words are merely the most specific objects in the hierarchy of categories. By way of example, some delegation relations between concepts in SG are represented in Figure 13.14.

13.5 DISCUSSION AND RELATION WITH OTHER WORK

The use of segments for the expression of grammatical knowledge is advantageous in an incremental sentence formulator. We will sum up some of these advantages here and at the same time draw comparisons with other work.

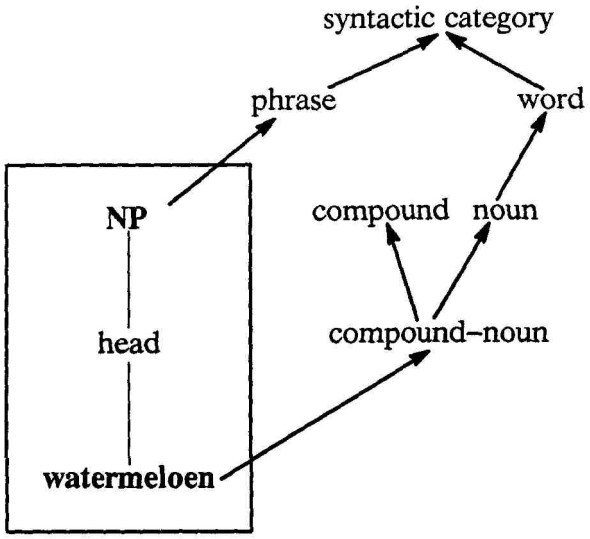

Figure 13.14: Some delegation relations from the nodes in a segment

13.5.1 Phrase Structure Rules and Categorial Grammar

Phrase Structure (PS) rules are string rewriting rules which express immediate dominance (ID, motherhood) relationships *together* with sisterhood (co-occurrence) and linear precedence (LP) relationships. Hence, it is often necessary to express the same dominance relationship more than once, namely for every possible sisterhood and linear precedence relationship. The example below is from Sells (1985).

(1)	a.	VP → V NP	kiss the bride
	b.	VP → V NP PP	send the message to Kim
	c.	VP → V NP S'	tell the class that break is over
	d.	VP → V NP VP	expect results to be forthcoming

We must observe that pure PS rules are seldom used. In Government and Binding (GB) theory, which uses PS rules mainly to specify hierarchical structure, order is fixed by other components of the grammar, such as case assignment. Generalized Phrase Structure Grammar (GPSG) (Gazdar, Klein, Pullum and Sag, 1985) uses rules in ID/LP format, where a comma in the right hand side of the rewrite rules indicates that the categories are unordered. These rules are then complemented with separate rules for precedence relations, e.g.:

(2)

a.	VP → V, NP	kiss the bride
b.	VP → V, NP, PP	send the message to Kim
c.	VP → V, NP, S'	tell the class that break is over
d.	VP → V, NP, VP	expect results to be forthcoming
e.	V < NP < XP	

Notice that, although the linear precedence is now encoded separately, ID relations are still expressed redundantly. SG offers a more economic way of encoding ID by specifying only one relationship between a pair of nodes at a time.

But the real problem in incremental production is that the choice between rules (2a-d) cannot be made deterministically. If daughter constituents are produced one at a time by means of a PS grammar, the system will be forced to choose between rules and backtrack when necessary. By virtue of its orientation toward the representation of separate ID relationships, SG allows the incremental addition of sister nodes and hence avoids backtracking (cf. Kempen and Hoenkamp, 1987).

This problem with PS rules could, in theory, be obviated by redefining the grammar as in (3). But then the PS structures generated by the grammar would not be isomorphic to those generated by grammar (2): the grammars are only weakly equivalent.

(3)

a. VP → V, NP, VPrest

b. VPrest → Ø

c. VPrest → PP

d. VPrest → S'

e. VPrest → VP

Although classical *categorial grammar* (CG), unlike PS rules, is lexically guided and therefore suited for generation, a similar objection could be raised against it. In classical CG, word order and co-occurrence constraints are encoded as syntactic types on lexical items. Whatever choices there are with respect to either LP or sisterhood will result in alternative syntactic types for the same word. For languages with relatively free word order or many sisterhood alternatives, this may result in a drastic increase of possibilities encoded in the lexicon. By comparison, the opposite is true for SG, which encodes restrictions on sisterhood rather than alternative possibilities. In SG, a relatively word order free language will therefore have a relatively small grammar and lexicon.

13.5.2 Unification

Unification as a general mechanism in language generation as well as in parsing has been proposed by Kay (1979). In Functional Unification Grammar (FUG), language processing is seen as the unification of an initial functional description with a grammar which is a very large functional description containing many alternatives. As shown by Appelt (1985), it is possible to use FUG in incremental production by letting an initial functional description unify with the grammar, then unify the result with more input in the form of another functional description, etc., until a sentence is completed.

However, a drawback of FUG is that the grammar is a complicated, monolithic structure. Our approach is different by virtue of the fact that unification is a local operation on two nodes. Consequently, the grammar can be represented as a set of segments, which express information at a local level, i.e. encapsulated in many separate objects rather than in one large structure or rule base.

The role of features in unification-based grammars (FUG, LFG, PATR-II and GPSG) is described by Karttunen (1984). Features are often treated as *attribute-value* pairs which are grouped in a feature *matrix* which then functions as the *agreement* of one or more nodes. Our approach is similar but we treat features as first-class objects in the sense that they may themselves be unified. This makes unification an even more universal mechanism and obviates the need for coreference to features by means of variables in the grammar.

13.5.3 TAG

Tree Adjoining Grammar (TAG; Joshi, 1987) is a tree generating system consisting of a finite set of elementary trees and a composition operation *(adjoining)* which builds derived trees out of elementary trees. Like SG, and opposed to PS-rules, TAG is tree-based rather than string-based. FTAG, the recent "feature structures based" extension of TAG (Vijay-Shanker and Joshi, 1988) uses unification of features as a clearer way of specifying restrictions on tree adjoining, and is therefore even more similar to SG. The role of some elementary trees in TAG is comparable to that of SG segments, while adjoining takes the role of unification. For example, the auxiliary tree for an adjective (Figure 13.15 (a)) can be said to correspond to the NP-modifier-AP segment (Figure 13.15 (b)), although it must be noted that SG always creates an adjectival phrase rather than just an adjective. The adjoining operation of the auxiliary tree for an adjective yields two NP nodes in the resulting structure (Figure 13.16 (a)), which is redundant, whereas the corresponding composition of SG segments will result in only one NP node (Figure 13.16 (b)).

Word order and immediate dominance are factored in the TAG formalism, which provides considerable flexibility in the generation process of a sentence.

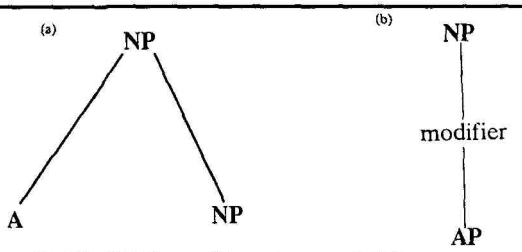

Figure 13.15: TAG auxiliary tree and SG segment

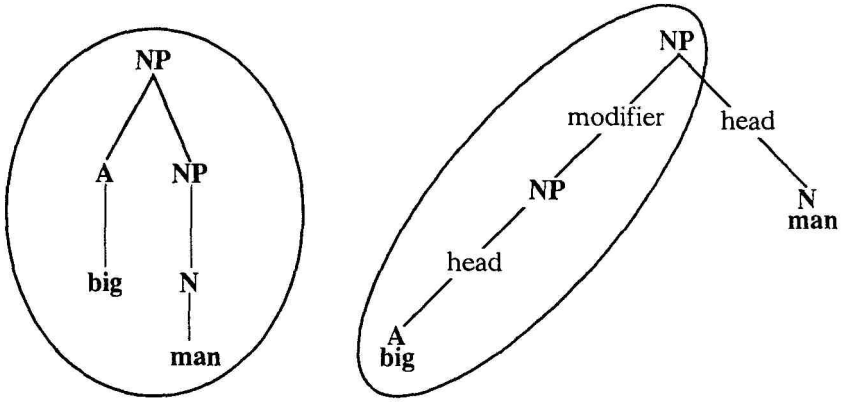

Figure 13.16: Results of TAG adjoining and SG unification

TAG allows incremental generation, but only as defined by the adjoining operation, which means that it does not allow the addition of sister nodes without structural growth in vertical direction. Unlike SG structures, TAG trees always contain all sisters. E.g., there are completely different elementary trees for transitive and intransitive verbs. It does not seem possible to build a transitive tree incrementally by starting with the intransitive tree and expanding it into a transitive one by furcating with a sister node, as SG might allow (if valency permits it).

Also, TAG seems to require, for many lexical items, a number of variants of elementary trees. For example, the transitive verb eat requires separate elementary trees for the constructions *Subj-V-Obj, WH(Obj)-Subj-V, to-V(infiniiive)-Obj*, etc. Since, presumably, the speaker has to make a choice between such alternatives at an early stage, this imposes restrictions upon incremental production which are avoided by SG. There, the choice between such constructions can be postponed to a later stage, so that minimal commitments with respect to further incrementation are imposed.

13.6 CONCLUDING REMARKS

SG describes sentences of a language in terms of syntactic segments - atomic units larger than syntactic nodes - and their possible combinations, governed by unification. Because SG specifies ID relations at the level of individual segments, f-structures can be generated by adding daughter, mother, *and* sister nodes incrementally. Because SG specifies LP relations at the level of individual segments, word order can be determined for partial sentences. These properties make SG a particularly flexible formalism for incremental generation. Although other formalisms can in principle be adapted for incremental generation, a lot of bookkeeping would be required to achieve the same effect (e.g., backtracking in PS grammars).

It seems that syntactic segments, as they are proposed here, are elementary structures which are small enough to allow any kind of incrementation, yet large enough to hold information (e.g., features for agreement, valency, LP rules) without having to resort to any additional global knowledge outside the segment definitions. We believe that this approach gives the grammar more modularity because the set of segments for a language is easily extendable and modifiable.

BIBLIOGRAPHY

Appelt, D. 1985. *Planning English Sentences.* Cambridge: Cambridge U.P.

Chomsky, N. 1981. *Lectures on Government and Binding.* Dordrecht: Foris.

Chomsky, N. 1965. *Aspects of the theory of syntax.* Cambridge, MA: MIT Press.

Dell, G.S. 1986. A Spreading-Activation Theory of Retrieval in Sentence Production. *Psychological Review,* 93, 238-321.

De Smedt, K. 1984. Using object-oriented knowledge-representation techniques in morphology and syntax programming. In: O'Shea, T. (ed.) *ECAI-84- Proceedings of the Sixth European Conference on Artificial Intelligence* (pp. 181-184). Amsterdam: Elsevier.

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* (pp. 365-376). Dordrecht/Boston: Martinus Nijhoff Publishers (Kluwer Academic Publishers).

De Smedt, K. 1987. Object-oriented programming in Flavors and CommonOR-BIT. 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.

Gazdar, G., Klein, E., Pullum, G. and Sag, I. 1985. *Generalized Phrase Structure Grammar.* Oxford: Basil Blackwell.

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. *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.

Sells, P. 1985. *Lectures on contemporary syntactic theories.* CSLI Lecture notes Nr. 3. Stanford: CSLI.

Vijay-Shanker, K. and Joshi, A. 1988. Feature Structures Based Tree Adjoining Grammars. In: *Coling '88: Proceedings of the 12th International Conference on Computational Linguistics, 22-27 August 1988.* Association for Computational Linguistics.