5 Segment Grammar

Koenraad De Smedt
NICI
University of Nijmegen
Postbus 9104
6500 HE Nijmegen, The Netherlands

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 (SG; Kempen, 1987) is used for the representation of linguistic knowledge. One basic operation, unification, is responsible for the construction of syntactic structures. This chapter presents a formal definition of this grammar formalism as well as an informal synopsis. Furthermore, it is shown how feature structures can naturally be seen as objects in an object-oriented representation language; an implementation of SG in CommonORBIT is presented.

5.1 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) show how various conceptual changes may affect the structure of the utterance

In C. L. Paris, W. R. Swartout & W. C. Mann (Eds.), Natural language generation in artificial intelligence and computational linguistics (pp. 329-349). Kluwer Academic Publishers.

which is under construction. The present chapter is concerned with the definition of a grammar formalism which is designed to meet the constraints imposed by an incremental mode of generation. It is argued that, regardless of their formal generative properties, not all grammar formalisms are equally suited to support incremental generation. In previous work, the following three requirements have been put forward to allow maximally incremental sentence generation (Kempen, 1987):

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

2 Because the size of each conceptual fragment is not guaranteed to cover a full clause or even a full phrase, it must be possible to attach individual branches to existing syntactic structures. This includes the incremental addition of sister nodes.

3 Because the chronological order in which conceptual fragments are attached to the syntactic structure does not necessarily correspond to the linear precedence in the resulting utterance, the language generation process should exploit variations in word order as they are made necessary by the partial utterance, but observing language particular restrictions on word order.

In order to satisfy these requirements, Kempen (1987) proposes a new formalism for constructing syntactic structures out of so-called syntactic *segments*. 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*. Segments join to form a syntactic structure by means of a general *unification* operation. The syntactic structure shown in Figure 5.1 consists of four segments.

[1] Kempen presupposes the necessity of a third operation, *insertion*. However, this operation would involve undoing previous expansions, which is computationally expensive and therefore psychologically questionable. Insertion can therefore better be treated as a reformulation (due to conceptual replacement) where the formulation of a fragment is possibly reused.

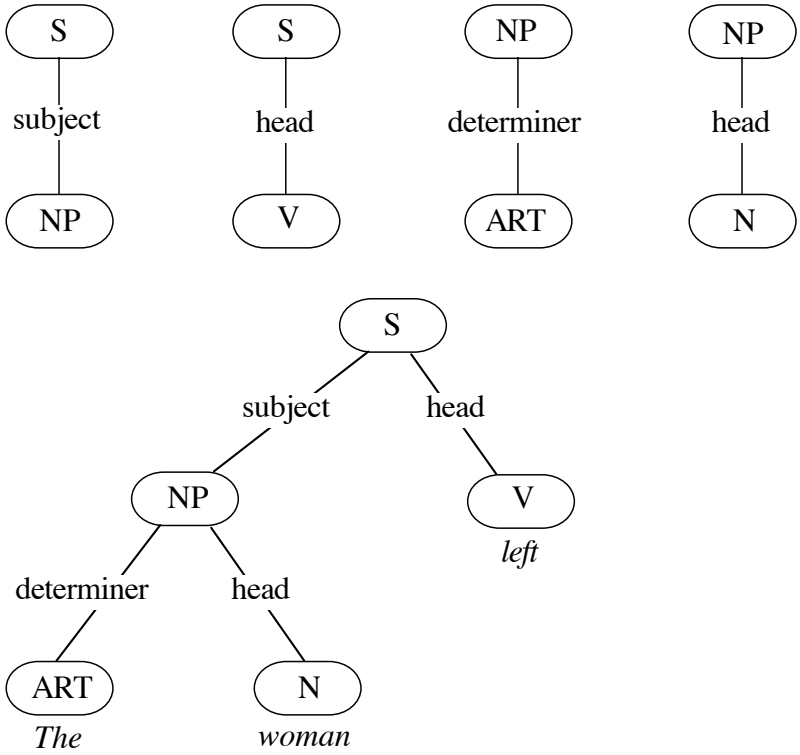

Figure 5.1
Syntactic segments and syntactic structure for the sentence "The woman left."

This formalism, which Kempen originally called *Incremental Grammar*, 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)*. In the remainder of this chapter, the reader will first be introduced to unification. Then it will be defined what constitutes an SG, and it will be shown how unification is used in this formalism. Then we will present an implementation of SG using object-oriented programming techniques, compare it with other formalisms, and point out its main advantages.

5.2 Unification

In IPG, as worked out by Kempen & Hoenkamp (1987), f-structures are implicitly represented in the hierarchy of procedures which generate a sentence. The enforcement of agreement was also achieved in a procedural manner. In SG, linguistic knowledge is represented in a more declarative manner. *Unification* is proposed here as a universal structure-building mechanism to replace the procedural mechanisms in IPG. It is embedded in an object-oriented grammar.

5.2.1 Features and agreement

Languages like English or Dutch exhibit certain regularities which are traditionally described as *agreement* (subject-verb agreement) or *concord* (gender concord). Context free phrase structure grammars fail to capture the regularities underlying these phenomena. E.g., consider the following fragment for English:

(1) a. S → {NPsg Vsg (NP) (NP),
NPpl Vpl (NP) (NP)}

b. NP → {NPsg, NPpl}

c. NPsg → (DETsg) Adj* Nsg

d. NPpl → (DETpl) Adj* Npl

e. Nsg → chair, computer, …

f. Npl → chairs, computers, …

The rules for singular phrases in this grammar have nothing in common with those for plural phrases. PS grammars fail to capture the fact that singular and plural phrases show essentially the same gross internal structure. There are several solutions for this deficiency. Many modern theories factor out *features*, such as number, gender and person, and formulate agreement principles as a separate part of the grammar. Thus, the arrangement of the agreeing elements is described in one part of the theory, and the details of how the elements agree are described in another part.

5.2.2 Unification in FUG and other unification-based formalisms.

Unification as a general mechanism for language generation as well as parsing was proposed by Kay (1979). In his Functional Grammar, later called Unification Grammar, now Functional Unification Grammar (FUG; Kay, 1985), generalized feature/value graphs called *feature structures* (or functional structures, or functional descriptions) are the sole informational entities. Language processing is seen as the unification of an initial feature structure with a grammar which is a large feature structure containing many alternative feature structures representing grammar rules. In unification-based grammars, a feature structure is a partial function from features to their values. E.g., there might be a function mapping the feature *plural* onto the value '-' and mapping *person* onto *1*. The notation used by Kay is:

(2)

As general graph structures, feature structures are recursive in the sense that feature values may themselves be structured, e.g.:

(3) cat =, NP, agreement =,

A feature structure in which features share one value is said to be *reentrant*. In that case, the common value is written just once and features are coindexed to indicate the sharing relation. E.g., in the following structure, the agreement of the subject is the same as the agreement of the predicate:

(4)

Error!

Sharing can also be notated as an equation, e.g.:

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

(6) VP: (↑ subject agreement) = (↓ agreement) (LFG)

Unification is based on the notion of *subsumption*—an ordering on feature structures expressing their compatibility and relative specificity. Shieber provides the following definition:

> "Viewed intuitively, then, a feature structure D subsumes a feature structure D' (notated $D \sqsubseteq D'$) if D contains a subset of the information in D'. More precisely, a complex feature structure D subsumes a complex feature structure D' if and only if $D(l) \sqsubseteq D'(l)$ for all $l \in dom\ (D)$ and $D'(p) = D'(q)$ for all paths p and q such that $D(p) = D(q)$. An atomic feature structure neither subsumes nor is subsumed by a different atomic feature structure." (Shieber, 1986:15)

E.g., feature structure (7) subsumes (2): it carries less information and contains no differing or conflicting information. Feature structures may not be in a subsumption relation with each other because they have differing but compatible feature values, e.g. (7) and (8), or because they contain conflicting information, e.g. (7) and (9).

(7) [plural =,-]

(8) [person =,1]

(9) [plural =,+]

In the case of differing but compatible information, there exists a more specific structure that is subsumed by both feature structures. E.g., (2) is the most specific structure which is subsumed by both (7) and (8). In other words, the unification D combines the information from two feature structures D' and D''. This is notated as $D = D' \sqcup D''$. For a more comprehensive overview of the formal foundations of unification and its use in present-day grammar formalisms, the reader is referred to Pollard and Sag (1987) and Shieber (1986); the latter also discusses Definite Clause Grammar as a kind of unification-based grammar.

5.2.3 CommonORBIT objects as feature structures

Unification-based grammar can be joined to object-oriented representation by representing feature structures as objects in CommonORBIT (De Smedt, 1989, 1987). The aspects (slots) of an object are then interpreted as features. E.g., the following object definition is equivalent to feature structure (10):

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

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

Since features are represented as aspects, they can be inherited in a specialization hierarchy. Thus, a hierarchy of feature structures can be constructed. Figure 5.2 shows an example hierarchy with an object equivalent to feature structure (10) at the bottom. The CommonORBIT definition can therefore be much simpler:

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

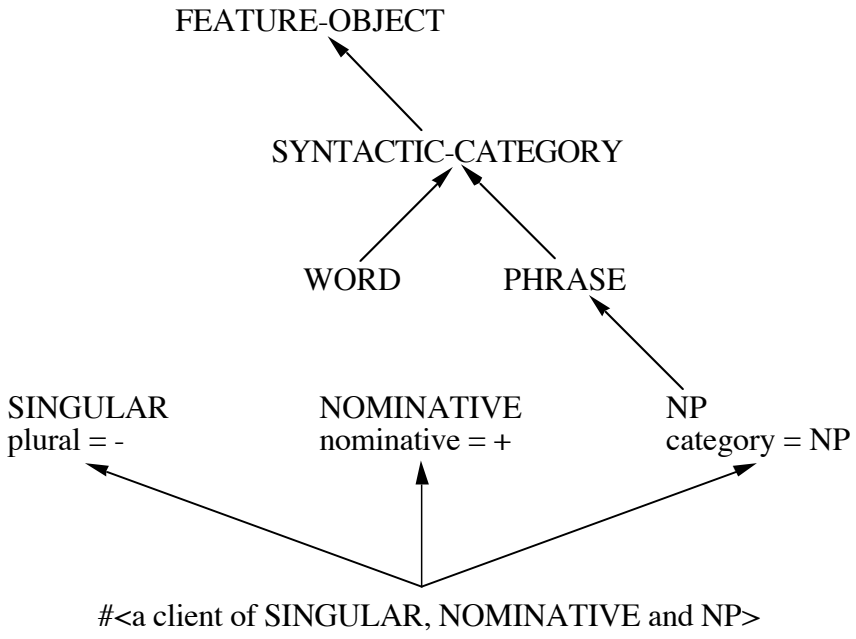

Figure 5.2
Example hierarchy of objects as feature structures

Whereas feature structures are usually seen as functions which map features onto values, an object-oriented representation does the reverse in the sense that features are *generic* functions which map feature structures onto values. If they were normal functions, this would be problematic, for those functions would have an infinite domain. However, since the definition of a generic function is distributed among

feature structures, function and argument can in theory be reversed, and thus the problem is not real. In fact, some object-oriented languages (e.g. FLAVORS: Moon, 1986) do represent objects as functions.

CommonORBIT provides a unification operation, called `UNIFY`, which unifies two objects by merging all the information contained in both objects into one. Not all aspects of an object are always essential for unification. The aspects involved in the unification of an object is obtained by the generic function `FEATURES`. The `UNIFY` 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. Match if this unification succeeds; the unification becomes the new feature value. Thus, unification is recursive.
2 Otherwise, the values are interpreted as sets (by coercing to lists) and the intersection is computed. However, the value `UNDEFINED` matches anything. Match if the result of the intersection is non-`NIL`; the intersection becomes the new feature value.

If unification succeeds, then 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`.

Although unification is applicable to all CommonORBIT objects, there is a special prototype `FEATURE-OBJECT` with convenient defaults which can be used as a proxy by other objects to be unified, as shown in Figure 5.2. For SG, the prototypical category is the object `SYNTACTIC-CATEGORY`. This object states, among other things, that the `CATEGORY` of the object is a required feature for unification. This prevents the unification of two different categories, e.g., S and PP.

5.2.4 Unification in Segment Grammar

In SG, it the following extensions c.q. restrictions are proposed:

1 *Disjunctive values.* While the values of features can only be atoms or feature structures in many unification-based formalisms, SG—like FUG—allows features to contain sets representing disjunctive values. Unification computes the intersection of two value sets and succeeds if the result is not empty[2]. E.g., the unification of (11) and (12) will be (13).

(11) [plural =,(+ -),person =,(2 3)]

(12) [plural =,+,person =,(1 3)]

(13) [plural =,+,person =,3]

2 *Non-recursive feature structures.* The recursiveness of most unification-based grammars is due to the fact that (1) they do not distinguish grammatical functions from 'true features'; they sometimes use a structured feature *agreement* (e.g. (4)). In SG, grammatical functions are not represented as features but rather as arcs on segments

[2] Atomic values are considered as singleton sets; sets are written in list notation.

(see also below). Furthermore, SG does not use a structured feature *agreement* but rather specifies features for agreement individually. Hence unification can *in principle* be restricted so that it is a non-recursive operation. In the present implementation, recursiveness is nevertheless possible. It is even so that features are *themselves* implemented as CommonORBIT objects, and are thus viewed as feature structures, but this recursiveness goes only one level; it is inessential and can be avoided altogether if necessary. Thus feature values could be restricted to symbols (atomic elements) or sets of symbols, which means that unification in SG is a considerably simpler operation than in most unification-based grammars[3].

3 *Disjunctive structures*. FUG takes seriously the idea of feature structures as the only stores of linguistic information, semantic as well as syntactic, and does not represent grammar rules other than in the form of feature structures. Since the parts of a feature structure operate conjunctively, the grammar must therefore also incorporate the notion of disjunction (or alternation). A FUG grammar is thus a large, disjunctive set of feature structures. SG does not incorporate this notion of disjunction, and indeed does not represent a grammar itself as a functional description. Unification in SG is applied to nodes rather than to segments. The amalgamation of segments by means of unification of their nodes is lexically guided; it is controlled by a process which is non-unification-based (cf. Chapter 7). Thus, SG applies unification more locally than FUG. The term *f-structure* (see below) in SG may therefore be somewhat confusing, for f-structures are *not* unified in the way feature structures (or functional structures, or functional descriptions) are unified in FUG.

5.3 Overview of Segment Grammar

Somewhat like a lexical-functional grammar (LFG; Kaplan & 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 represented as 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 grammatical relations between words and phrases, as traditionally expressed by 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. When SG is activated in an incremental processing mode, 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

[3] In a similar way, Proudian & Pollard (1985) do not use general graph structures but propose restricting HPSG structures to 'flat' feature matrices where each value is a set of atomic features. When such structures are represented as vectors of integers, they can be unified in a very fast way using the 'logical AND' machine instruction.

eventually uttered as speech sounds. This overall process is depicted in Figure 5.3. We will now be concerned with the elements which constitute the grammar.

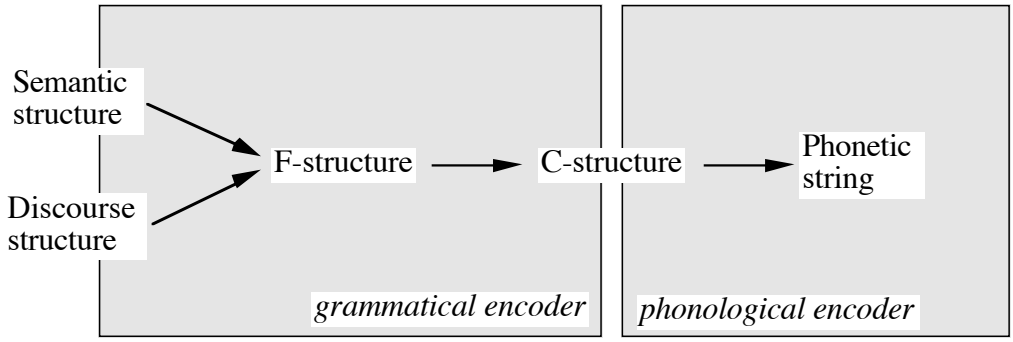

Figure 5.3
The generation of subsequent linguistic descriptions during the formulation stage

5.4 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[4].

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[5]: $F \times V \rightarrow \wp(W)$, F_L a labeling function: $V \rightarrow N \cup T$, and F_O a labelling function $E \rightarrow O$. The set of f-structures in G is defined as $\mathbb{F}_G = S \cup \{y \mid \exists\, y', y'' \in \mathbb{F}_G: y \in U(y',y'')\}$ where U is the universal unification function: $\mathbb{F}_G \times \mathbb{F}_G \rightarrow \wp(\mathbb{F}_G)$ which derives new f-structures by unifying a node of one f-structure with a node of another. Unification of nodes is introduced earlier in this chapter.

An f-structure is well-formed if observes the principles of completeness and coherence stated in (14) and (15). These principles refer to valence specifications which are projected from the lexicon.

(14) 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.

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

[5] $\wp$ denotes the powerset. Sets of feature values, as allowed by the function F_W, represent disjunctive values.

(15) 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.

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 F_L 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 in Chapter 7.

For a Segment Grammar G, *phonetic strings* are structures $\in P^*$. The phonetic strings in L_G are the sequences of terminal nodes of all possible c-structures in G.

5.5 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 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 5.4. In running text, segments are also written left-to-right (root-to-foot), e.g. S-SUBJECT-NP or NP-HEAD-NOUN.

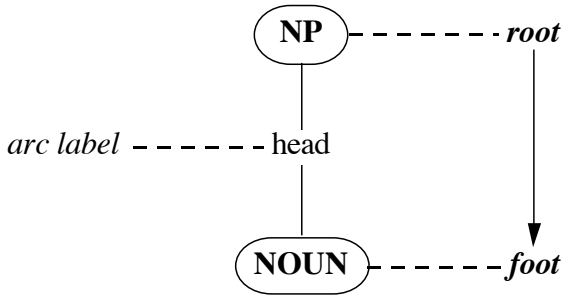

Figure 5.4
A syntactic segment

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 5.5.

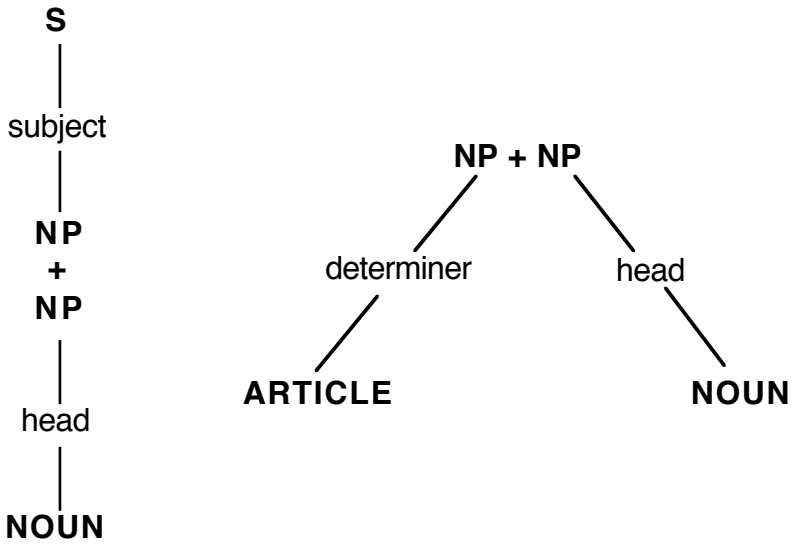

Figure 5.5
Concatenation (left) and furcation (right)

5.5.1 Features and agreement

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, then the feature has all possible values by default, in this case the set[6] (+ -). When two nodes are unified (in either concatenation or furcation), their features are also unified. This process, illustrated in Figure 5.6, is essentially the same as feature unification in other unification-based formalisms and will be further described later in this chapter. It consists of computing the union of all features in both nodes, and for each feature the intersection of the values in both nodes.

[6] Sets are represented here in list notation.

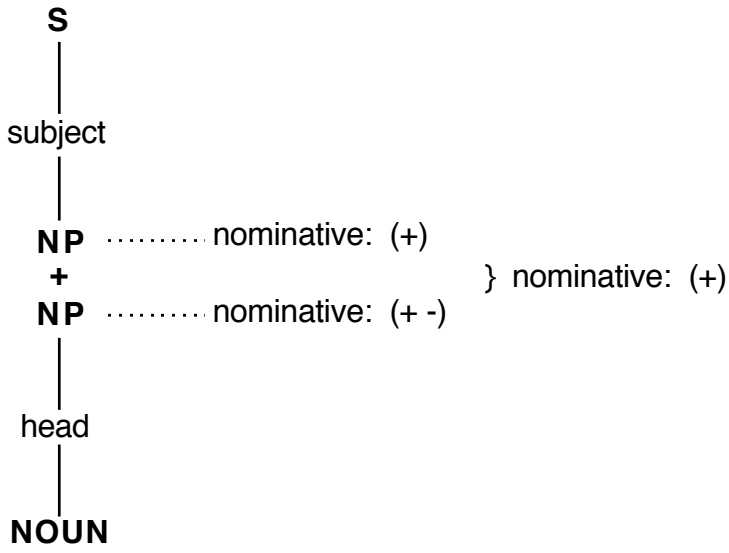

Figure 5.6
Feature unification

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 5.7.

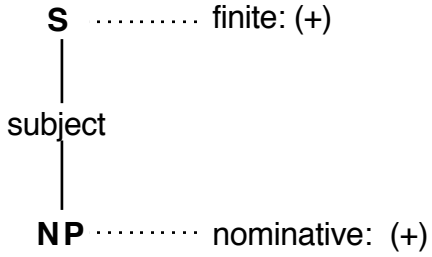

Figure 5.7
Co-occurrence of features in a segment

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 5.8.

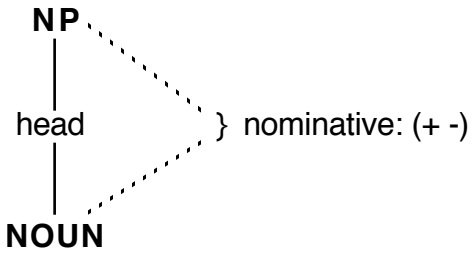

Figure 5.8
Feature sharing

The combination of feature sharing and unification amounts to 'feature transport'. By virtue of the sharing relationship in NP-HEAD-NOUN, the concatenation depicted in Figure 5.9 results in a feature change to the foot of the lower segment as well as to its root. Features are in fact not transported from one segment to another, but they are unified so that they are shared by nodes of different segments. In the case of multiple concatenations and/or furcations, features may well be shared by three or more nodes in the syntactic structure.

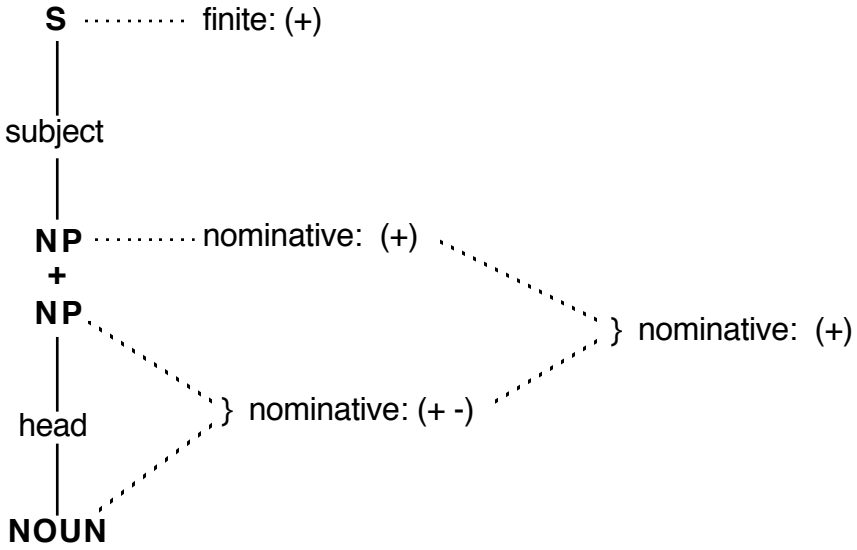

Figure 5.9
Unification of a shared feature

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 5.10.

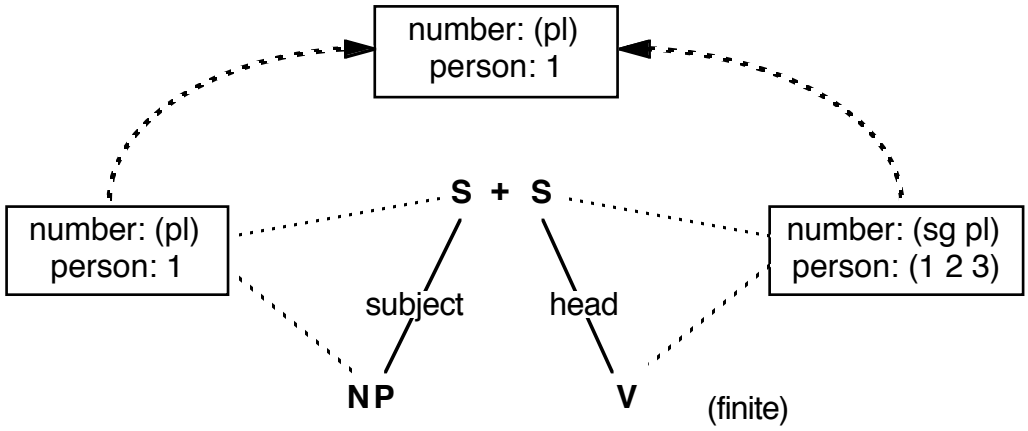

Figure 5.10
Agreement by means of feature sharing

Agreement in many unification-based formalisms can be specified between constituents at arbitrary distances by using *paths*, expressed as the coindexing in (4), or as the equations (5) and (6). Our approach is different by virtue of the fact that agreement can only be specified on the level of individual segments, namely as shared features between root and foot. SG *distributes* grammatical constraints among the various syntactic segments which occur in a language. In general, an 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 (cf. Chapter 7).

5.5.2 Feature sharing vs. feature coocurrence

It has already been shown that agreement can be enforced by specifying features which are shared between foot and root. Intuitively, such a relation can be described as, e.g. "the finite S and its subject NP agree with respect to the feature *person*". This is represented in Figure 5.11.

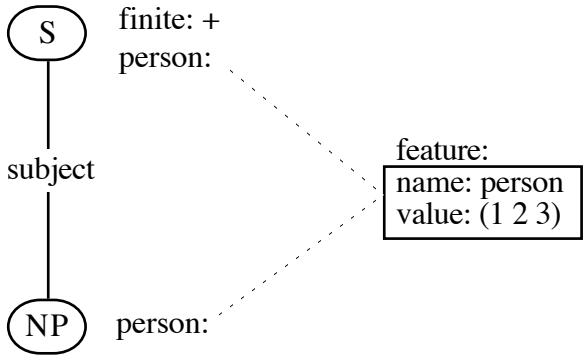

Figure 5.11
Feature sharing

Specifying shared features in a segment thus serves the same purpose as coindexing to indicate structure sharing in other unification-based formalisms. Any subsequent changes to the feature in either the foot or the root will also affect the other. Stating that *all* features be shared between root and foot in some segments would be a possible implementation of the Head Feature Principle (HFP) in HPSG. It is not possible to specify feature sharing between sister nodes in SG. Such agreement relations can only indirectly be enforced by specifying several sharing relations over furcated and concatenated segments.

A second method of enforcing syntactic constraints is more general. It can be specified that some feature values on the root of a segment cooccur with some feature values on the foot. Again, such specifications may only be given at the level of individual segments, and not at arbitrary distances, as other unification-based formalisms might allow. Intuitively, such a relation can be described as, e.g., "The indefinite singular neuter NP has an uninflected premodifier AP". This is represented in Figure 5.12.

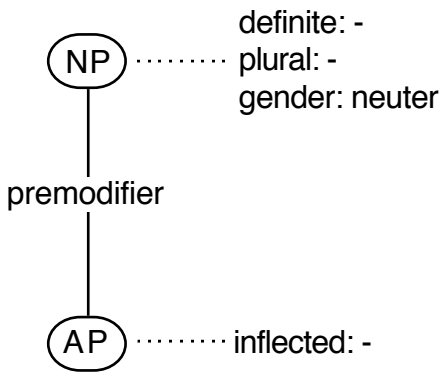

Figure 5.12
Feature cooccurrence on root and foot

On the one hand, feature cooccurrence specifications are more general, because feature sharing cannot establish a relation between features with a different name. On the other hand, they are less powerful than feature sharing, because feature sharing corresponds to one cooccurrence relation for every possible feature value. E.g., the information in Figure 5.11 can also be expressed by three separate cooccurrence relations, each containing a different value for the feature *person*. Thus, feature sharing is preferred where possible to avoid having to make a choice between several segments. Cooccurrence specifications seem to be useful for two purposes:

1 They can express constraints between very different kinds of features, e.g., between syntactic and morphological features, as in Figure 5.12.
2 They can express constraints which are often associated with grammatical relations, e.g., the fact that "the subject NP of a finite S is nominative". If this kind of information is exhaustively encoded by means of features, grammatical relations like *subject* become superfluous in SG.

5.5.3 Immediate dominance / linear precedence

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 (cf. Starosta, 1978). 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 language generation process as a whole 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, which was regarded as essential by Kempen (1987) may turn out to be disposable in SG.

Linear precedence (LP) is encoded by assigning to each foot node a feature POSITIONS 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 process of assigning left-to-right order in c-structures will be further explained in the next chapter.

5.6 An object-oriented implementation of Segment Grammar

SG is implemented as an object-oriented unification-based grammar by uniformly representing important syntactic segments and other grammar units, such as nodes (phrases, words) and features, as computational objects. Objects representing categories are viewed as feature structures which can be unified using a general unification operation. 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))
```

Rather than associating an atomic feature value directly with a category, objects representing features occur as values in categories. E.g., a NP has a feature for plural, which has all possible values:

```
(DEFOBJECT NP
  (PLURAL (A PLURAL-FEATURE))
  ...)
```

The reason for doing this is that features can themselves be interpreted as feature structures and thus they can also be unified. We can define 'mixins' which represent more specific feature values, e.g., an object PLURAL which contains only the value '+' for the plural feature:

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

Segments have aspects for the root, foot, and arc label, and also for the features which are shared between root and foot. Specific segments may *delegate* to more general ones. By way of example, the CommonORBIT definition of some segments, including a lexical segment, is given below. Also given is a representation of the Dutch word *watermeloen* (watermelon) which occurs in one of the segments.

```
(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. Only knowledge which is specific to a particular segment is listed in the definition of that segment. For example, in NP-HEAD-NOUN, categories restricting the root and foot slots are given. Other knowledge is inherited from more general segments. The

prototypical SYNTACTIC-SEGMENT, which acts as a proxy for specific segments such as NP-HEAD-NOUN, contains the most 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)))))
```

Segments are thus efficiently organized in a multiple inheritance hierarchy (cf. Chapter 5). E.g., knowledge common to both segments S-subject-S and S-subject-NP is stored in a general segment S-subject-*. Knowledge common to all subordinate clauses is stored in *-*-S. This is schematically represented in Figure 5.13:

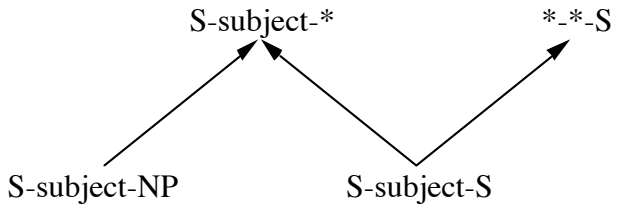

Figure 5.13
A specialization hierarchy of syntactic segments

Feature sharing can be modeled in various ways. It can be based on feature unification: upon the creation of a segment, the features to be shared in the foot and the root are unified. Thus, the definition of a segment contains a specification of which features must be unified. This is the representation which is currently used in the CommonORBIT implementation of SG. But an alternative implementation is possible. Rather than creating two features and unifying them, we could make just one feature and then assign it to the foot and the root. This can be achieved by using the lexical scope mechanism of Common Lisp[7]:

[7] A similar treatment using use of variables is found in the grammar proposed by Steels and De Smedt (1983) (cf. Chapter 4). It is also similar to the use of variables in Definite Clause Grammar.

```
(LET ((P (A PERSON-FEATURE))
      (PL (A PLURAL-FEATURE))
      (G (A GENDER-FEATURE))
      (NOM (A NOMINATIVE-FEATURE)))
  (DEFOBJECT NP-HEAD-NOUN
    SYNTACTIC-SEGMENT      ;delegate to prototypical segment
    (ROOT (A NP
             (PERSON P)
             (PLURAL PL)
             (GENDER G)
             (NOMINATIVE NOM)))
    (ARC 'HEAD)
    (FOOT (A NOUN
             (PERSON P)
             (PLURAL PL)
             (GENDER G)
             (NOMINATIVE NOM)))
    ((FOOT POSITIONS) '(6))       ;word order possibilities
    )
```

The nodes representing syntactic categories on the root and foot of a segment are also defined as CommonORBIT objects. The structured inheritance mechanism in CommonORBIT establishes delegation relations between the root of a segment and that in its proxies, and likewise between the foot of a segment and that of its proxies. For the segments defined above, these structured delegation relations are depicted in Figure 5.14. From this hierarchy, it can be seen that `WATERMELOEN` can obtain a value for the aspect `POSITIONS` by delegation to a higher segment.

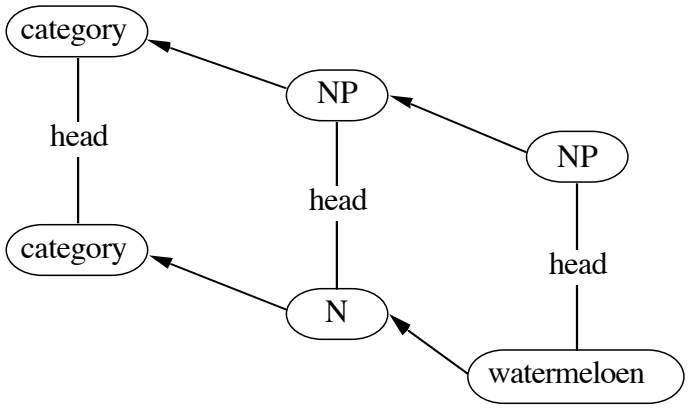

Figure 5.14
Structured inheritance in segments[8]

Phrasal categories, parts of speech and lexical entries are represented in the same delegation hierarchy. Consequently, the grammar and the lexicon are in a continuum: lexical segments are merely the most specific objects in the specialization hierarchy of segments, and words are the most specific objects in the specialization hierarchy of

[8] For readability, the nodes are labeled with their categories. However, it must be kept in mind that the nodes themselves are clients of these categories.

syntactic categories. By way of example, some delegation relations (slightly simplified) between linguistic units in SG are represented in Figure 5.15.

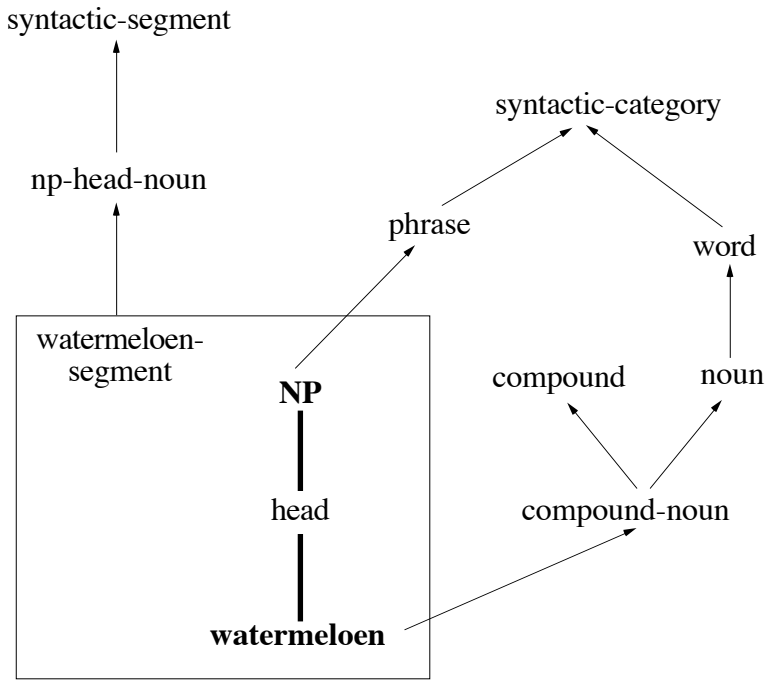

Figure 5.15
Some delegation relations in an object-oriented SG.

5.7 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.

5.7.1 Phrase Structure Grammar 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 in (16) is from Sells (1985).

(16) 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.:

(17) 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 (17a-d) cannot be made deterministically (cf. Kempen & Hoenkamp, 1987). 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. Although in GPSG, subcategorization indices will solve the choice between rules (17a-d), this does not work when several subcategorization frames are possible for one predicate and constituents may be added by incremental upward as well as downward expansions. By virtue of its orientation toward the representation of separate ID relationships, SG allows the incremental addition of sister nodes and chooses between subcategorization frames while the construction of the syntactic structure is already under way.

This problem with PS rules could, in theory, be obviated by redefining the grammar as in (18). But then the PS structures generated by the grammar would not be isomorphic to those generated by grammar (17): the grammars are only weakly equivalent.

(18) 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, where a relatively word order free language will have a relatively small grammar and lexicon.

It must be noted, however, that some kinds of categorial grammar allow considerably more flexibility. By choosing non-directional categories rather than right-directional and left-directional ones, some variations in word order can easily be accounted for. In combination with rules such as composition and category-raising, 'flexible' categorial grammars could even provide an elegant description of a free word order language such as Warlpiri (Bouma, 1986). Flexible categorial grammars allow constituents to be built for any part of a sentence, producing many possible derivations for a given sentence. No unique constituent structure is assigned to a sentence, but the order in which elements combine is rather free. Since syntactic rules operate in parallel with semantic rules, this suggests that incremental processing is facilitated not only for parsing (cf. Ades & Steedman, 1982; Bouma, 1989) but also for generation.

5.7.2 Tree Adjoining Grammar

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 as opposed to PS-rules, TAG is tree-based rather than string-based. FTAG, the recent "feature structures based" extension of TAG (Vijay-Shanker & 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 5.15a) can be said to correspond to the NP-modifier-AP segment (Figure 5.15b), although it must be noted that SG always creates an adjectival *phrase* rather than just an adjective.

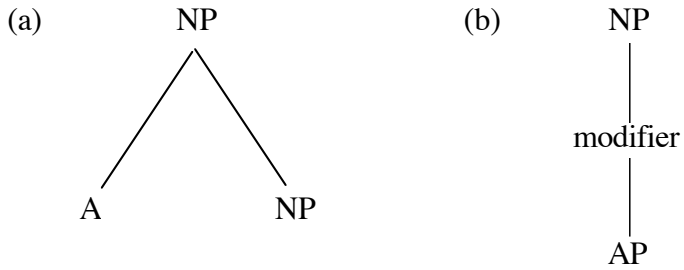

Figure 5.15
TAG auxiliary tree and SG segment

The difference between the two approaches is, that TAG allows the *recursive* insertion of A nodes, whereas SG allows the *iterative* addition of AP nodes. In terms of PS grammar, the corresponding rules are:

(19) a. NP → A NP (TAG)

b. NP → AP* N (SG)

The adjoining operation of the auxiliary tree for an adjective yields two NP nodes in the resulting structure (Figure 5.16a), whereas the corresponding composition of SG segments will result in only one NP node (Figure 5.16b). Whereas it can be argued that the hierarchical embedding of NPs in TAG allows an easier semantic interpretation in some cases (e.g. for the order of non-compositional adjectives: *an old strong man* vs. *a strong old man*) it also makes for considerable redundancy in cases where recursiveness does not affect semantic interpretation.

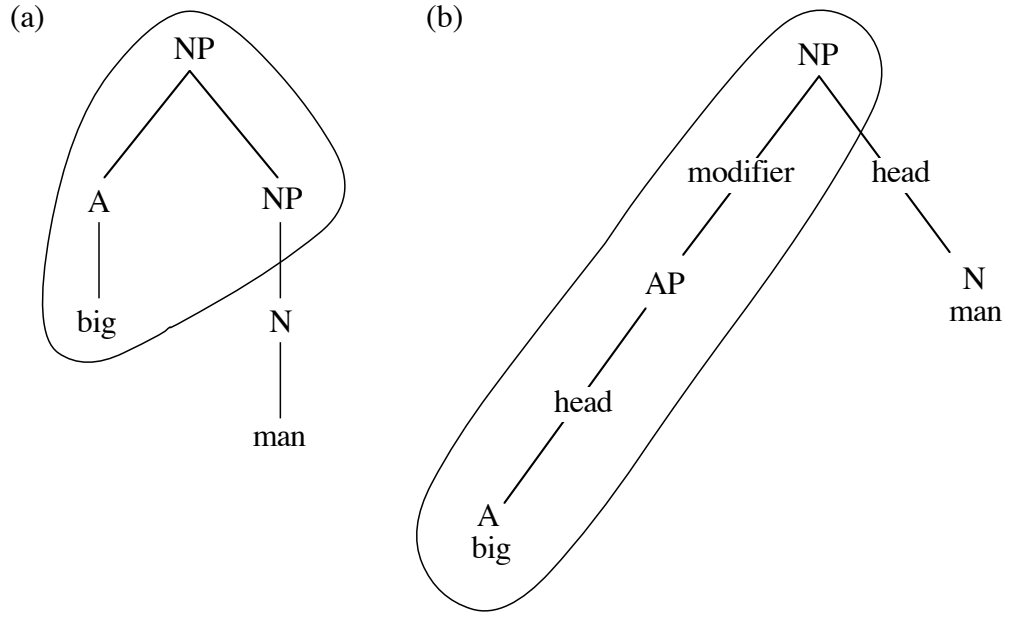

Figure 5.16
Results of TAG adjoining and SG unification

Word order and immediate dominance are factored in the TAG formalism, which provides considerable flexibility in the generation process of a sentence. TAG allows incremental generation, but only as defined by the adjoining operation, which means that it does not allow the addition of sister nodes *(furcation)* 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 an intransitive tree and expanding it into a transitive one by furcating with a sister node, as SG might allow (if valence 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(infinitive)-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.

5.7.3 Lexical-Functional Grammar

SG is perhaps closest to Lexical-Functional Grammar (LFG; Bresnan, 1982). Both formalisms assign an f-structure as well as a—not necessarily isomorphic—c-structure to a sentence. This distinction is felt to be a useful abstraction in both formalisms, and is used to account for such things as functional control (e.g. S-O raising, cf. Chapter 7). The structures which are assigned by both formalisms to sentences are often very similar. S-rules in LFG which are annotated with functional schemata contain essentially the same information as syntactic segments. A basic LFG rule such as (20a) could in principle be translated into two SG segments (20b).

(20) a. S → NP VP
↑ subj = ↓ ↑ = ↓

b. S-subject-NP (positions: (1))
S-head-VP (positions: (2))

However, upon closer inspection, a number of differences become apparent. Whereas in LFG, c-structures are generated independently by means of a PS grammar, SG derives c-structures directly from f-structures (this is explained further in Chapter 7). In LFG, which forbids operations on f-structures, there is no way to derive a c-structure from an f-structure, although derivation can be f-structure *driven* (cf. Wedekind, 1988).

On the representation level, LFG encodes functional relations as well as features uniformly in f-structures; functional structures are thus feature structures. In SG, however, only nodes are represented as feature structures; functional structures form a different graph. The advantage of the latter representation is that unification is a more local operation and can be simpler because it need not be recursive.

5.8 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).

In the form of features and word order information, segments contain all necessary information which is traditionally associated with grammatical functions such as subject, direct object, etc. Although grammatical function is retained in the form or arc labels, this is done for clarity, and not a single syntactic decision in the formulator need rely on arc labels.

The choice of unification as a mechanism for the composition of syntactic informational structures out of segments is motivated by the fact that unification is a general, adequately formalized formalism. A wide range of present-day linguistic theories is based on unification, which allows their explicit construction, computer implementation and mutual comparison.

Although other contemporary grammar formalisms (e.g. GPSG, CG, TAG, LFG) offer considerable flexibility for incremental parsing and generation, SG distinguishes itself from these formalisms mainly in that it views the segment as a domain of locality. 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, valence, LP rules) without having to resort to any additional global knowledge outside the segment definitions. We believe that this approach provides the grammar with more modularity because the set of segments for a language is easily extendable and modifiable.

I conclude that SG is *not* proposed as another formalism which is claimed to be formally more powerful or less powerful than other grammars, or accounts for different linguistic facts. Rather, it is a grammar where linguistic knowledge is organized so as to serve a specific language *processing* model. This model will be the subject of Part Two.

References

Ades, A. & Steedman, M. 1982. On the order of words. *Linguistics & Philosophy* 4, 517-518.

Bouma, G. 1986. Kategoriale Grammatica en het Warlpiri. *Glot* 8, 227-256.

Bouma, G. 1989. Efficient processing of flexible categorial grammar. In: *Fourth Conference of the European Chapter of the Association for Computational Linguistics, Manchester, 10-12 April 1989* (pp. 19-26). ACL.

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

Bresnan, J. 1978. A realistic transformational grammar. In: Halle, M., Bresnan, J. & Miller, G. (eds.) Linguistic theory and psychological reality. Cambridge, Mass.: MIT 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.

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.

Gazdar, G., Klein, E., Pullum, G. & 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.

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

Karttunen, L. 1984. Features and values. In: *Proceedings of Coling84* (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: Berkeley Linguistic Society.

Kay, M. 1985. Parsing in functional unification grammar. In: ?? (ed.) Natural language parsing: psychological, computational and theoretical perspectives (Chapter 7, pp. 251-278). Cambridge: Cambridge University Press.

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

Kempen, G. 1987. A framework for incremental syntactic tree formation. In: *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87), Milan* (pp. 655-660). Los Altos: Morgan Kaufmann.

McDonald, D. & Pustejovsky, J. 1985a. A computational theory of prose style for natural language generation. In: *Proceedings of the Second Conference of the European Chapter of the Association for Computational Linguistics, Geneva*.

McDonald, D. & Pustejovsky, J. 1985b. TAGs as a grammatical formalism for generation. In: *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics, Chicago*.

Moon, D. A. 1986. Object-oriented programming with FLAVORS. *Proceedings of OOPSLA '86*. ACM.

Pollard, C. & Sag, I. 1987. *Information-based syntax and semantics*. Stanford: CSLI.

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

Shieber, S.M. 1986. *An introduction to unification-based approaches to grammar*. CSLI Lecture Notes No. 4. Stanford: CSLI.

Starosta, S. 1978. The one per sent solution. In: Abraham, W. (ed.) *Valence, semantic case and grammatical relations* (pp. 459-576). Amsterdam:

Vijay-Shankar, K. & Joshi, A.K. 1985. Some computational properties of Tree Adjoining Grammars. In: *Proceedings of the 23rd annual meeting of the Association for Computational Linguistics, Chicago*. Association for Computational Linguistics.

Vijay-Shanker, K. & Joshi, A.K. 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.

Wedekind, J. 1988. Generation as structure driven derivation. In: *Coling-88: Proceedings of the 12th International Conference on Computational Linguistics, Budapest, 22-27 August 1988* (pp. 732-737). ACL.