Performance Grammar: a Declarative Definition

*Gerard Kempen**, *Karin Harbusch*†

∗ Cognitive Psychology Unit, Department of Psychology, Leiden University & MPI for Psycholinguistics, Nijmegen
† Computer Science Department, University Koblenz–Landau

Abstract

In this paper we present a definition of *Performance Grammar (PG)*, a psycholinguistically motivated syntax formalism, in declarative terms. PG aims not only at describing and explaining intuitive judgments and other data concerning the well–formedness of sentences of a language, but also at contributing to accounts of syntactic processing phenomena observable in language comprehension and language production. We highlight two general properties of human sentence generation, incrementality and late linearization, which make special demands on the design of grammar formalisms claiming psychological plausibility. In order to meet these demands, PG generates syntactic structures in a two-stage process. In the first and most important 'hierarchical' stage, unordered hierarchical structures ('mobiles') are assembled out of lexical building blocks. The key operation at work here is typed feature unification, which also delimits the positional options of the syntactic constituents in terms of so-called topological features. The second, much simpler stage takes care of arranging the branches of the mobile from left to right by 'reading–out' one positional option of every constituent.

In this paper we concentrate on the structure assembly formalism in PG's hierarchical component. We provide a declarative definition couched in an HPSG–style notation based on typed feature unification. Our emphasis throughout is on linear order constraints.

1 Introduction

Performance Grammar (PG) is a psycholinguistically motivated grammar formalism. It aims to describe and explain intuitive judgments and other data concerning the well–formedness of sentences of a language, but at the same time it hopes to contribute to accounts of syntactic processing phenomena observable during language comprehension and language production. Due to space limitations, we cannot detail the grammar design desiderata emerging from these phenomena. Here, we highlight two sentence production characteristics:

(1) *Late linearization*: The linear order of constituents is realized *after* the hierarchical (functional, grammatical) relations between them have been established.

(2) *Incrementality*: Speakers often construct and deliver sentences in a piecemeal fashion, risking premature utterance release and 'talking themselves into a corner'.

The first desideratum derives from speech error phenomena discovered by Garrett (1975). He identified two stages of syntactic processing in sentence generation: an early 'functional' and a later 'positional' stage. This distinction has since been adopted by most students of language production (e.g., see Bock and Levelt, 1994). We follow Garrett's lead by assuming that syntactic tree formation in PG is a two–stage process. First, an unordered hierarchical structure ('mobile') is assembled out of lexical building blocks. The key operation at work here is feature unification, which also delimits the positional *options* of the syntactic constituents. During the second stage, the branches of the mobile are arranged from left to right by a 'Read–out' module that realizes one positional option of every constituent.

Incremental sentence production does not imply that the *spatial* (left–to–right) order of the successive increments (sentence fragments, substrings) in a sentence correlates perfectly with the *temporal* order in which these fragments have been created and released into the output string. An obligatory constituent that is released later than, but needs a position before, a non-obligatory one may be skipped inadvertently and receive an illegal output position. Consider, for instance, a speaker who is grammatically encoding a finite adverbial subordinate clause and intends to append it to the main clause. If, for whatever reason, the complementizer that should open the subclause takes more time to construct than subject NP and finite verb, it will follow rather than precede the subject and verb in the output string. Therefore, incremental generators need special precautions to prevent constituents to be released prematurely. Grammar formalisms that, as required by the first psycholinguistic property, separate hierarchical from positional computations, typically use linear precedence (LP) rules to linearize the branches of unordered trees. LP rules specify the *relative position* of (pairs of) constituents (cf. the ID/LP format in Generalized Phrase Structure Grammar (GPSG); see, e.g., Gazdar *et al.*, 1985). However, a linearization method based exclusively on LP rules cannot meet the requirements of incremental sentence generation. In terms of the example, suppose the grammar uses LP rules such as "Complementizer < Subject", "Subject < Verb" and "Complementizer < Verb". This allows subject and verb to be ordered correctly when they are ready to be released and uttered more or less simultaneously. But no warning is issued if the complementizer has not yet been appended to the output string and runs the risk of being skipped or misplaced. One possible solution to this problem is to supplement LP rules with rules that somehow can reference the spatial layout of a constituent and assign subconstituents to *absolute positions* within this layout. For instance, suppose the spatial layout of a subordinate clause would define a bank of 'slots' to be filled by clause constituents. Then, a linearization constraint might stipulate that Slot #1 is obligatorily occupied by one constituent, in particular by a complementizer. This is the approach we have followed in our design of the PG formalism (cf. Kempen and Hoenkamp, 1987, for an early implementation).

To prevent misunderstandings, we do not deny that people are liable to uttering sentence fragments prematurely and, at times, do 'talk themselves into a corner'. But they do so much less frequently than expected on the basis of a generation system without any provisions against premature utterance release.

Although psycholinguists are primarily interested in processing phenomena, that is, in the *procedural* aspects of human syntactic processing, we considered that, in order to enhance the comparability between PG and competing grammar formalisms, a *declarative* definition of Performance Grammar is desirable. Here, we present such a definition couched in a notation based on typed feature unification, as in Head–driven Phrase Structure Grammar (HPSG). Our emphasis throughout is on linear order. In Section 2, we introduce PG's elementary data structures and structure assembly operations informally, using examples from English and Dutch. Section 3 is devoted to unification–based linearization. In Section 4, Performance Grammar is compared to other linguistic formalisms from the viewpoint of psychological plausibility. Section 5 contains evaluative comments and conclusions.

2 Performance Grammmar: a procedural introduction

2.1 Basic assumption

Performance Grammar (PG) is a tree assembly system generating pairs consisting of a hierarchy of *concepts* on the one hand, and a string of *lemmas* on the other. The conceptual structure of an output pair represents the meaning of the lemma string. A conceptual structure is a hierarchy of concepts connected by *thematic* relations (agent, recipient, theme), quantifier scope relations, etc. The individual lemmas in output pairs are annotated by morphological diacritics (case, gender, number, etc.) and map onto phonologically specified lexical items (*lexemes*). PG is fully lexicalized, that is, the elements out of which the various structures are composed, all originate from the lexicon; there are no rules that introduce additional elements.

The lexical entries used by the grammar are triples consisting of

(1) a *concept frame* comprising a concept together with its thematic and other relations; the concept represents the meaning of the entry;
(2) a *lexical frame* specifying the part–of–speech (word class) of a lemma and its subcategorization information (grammatical functions fulfilled by constituents, e.g., subject NP, head verb, modifier AP); the lemma itself functions as the head of the frame and as the entry's ID; and
(3) a *topology*, i.e., a fixed number of serial positions to be occupied by syntactic constituents in the lexical frame.

The members of a triple are complex units themselves, and the individual components of a member may be cross–linked to components of another member. For instance, thematic relations of a concept frame map onto the grammatical functions of a lexical frame, and vice–versa. This bidirectional mapping between components of lexical and concept frames is specified by so–called *synchronization links*. A topology is a one–dimensional array of slots. Each slot may serve as the 'landing site' of one (sometimes more than one) syntactic constituent. This mapping consists of bidirectional *linearization links*.

The construction of the grammar's output pairs proceeds in three steps.

(1) A hierarchy of lexical frames is assembled in parallel (in synchrony) with a hierarchy of concept frames. The latter hierarchy is one member of the output pair.
(2) The hierarchy of lexical frames includes branches whose terminal node is a lemma. These branches select a slot in one of the topologies accompanying the lexical frames. This happens in preparation of the final word order.

(3) A Read–out module traverses the topologies depth–first from top to bottom and from beginning to end, inspecting the linearization links deposited in the slots. If such a link references a lemma, it is appended to the current output string, together with its diacritics. Lemma strings delivered by the Read–out module are the other members of the grammar's output pairs.

The hierarchy of lexical frames emerging during the assembly process, i.e., an unordered syntactic tree, we call the *internal* structure of the utterance, in contrast with the two members of an output pair, which may be considered external structures.

We now discuss the formation of internal structures and lemma strings in more detail. Concept hierarchies will receive only cursory attention.

2.2 Hierarchical structure in PG

PG's internal syntactic structures are unordered trees composed of *lexical frames*. These are mobiles assembled from branches called *segments*. A segment consists of three nodes: the root is a phrasal node (Sentence, Noun Phrase, Adjectival Phrase, Prepositional Phrase); the foot node is categorial (i.e. phrasal or lexical); the node in the middle is *functional* (that is, labeled by a grammatical function, e.g. SUBJect, HeaD, MODifier). A lexical frame contains a fixed number of segments, all with different functional nodes but with the same root node. The roots of these segments are merged, giving rise to a tree with three layers of nodes. Exactly one segment of a frame has a lexical foot node: this is the head segment, whose middle node is labeled "HD". Each lexical frame is 'anchored' to exactly one lexical item: a *lemma* (printed below the lexical node serving as the frame's HeaD). A lexical frame encodes the word category (part of speech), subcategorization features, and morphological diacritics (person, gender, case, etc.) of its lexical anchor (cf. the elementary trees of Tree Adjoining Grammar (TAG; e.g. Joshi and Schabes, 1997).

All non–head segments have a phrasal foot node, which can be replaced ('substituted for') by a whole lexical frame. *Substitution* gives rise to hierarchical structures comprising, in principle, any finite number of lexical frames (Figure 1). Substitution, PG's sole composition operation, causes the foot node of one segment to merge with the root of another segment (see the filled circles in Figure 1). Associated with every categorial node (i.e., phrasal or lexical node) is a *feature matrix*, that is, a set of attribute–value pairs. Whenever two nodes are merged, their feature matrices are *unified*. In accordance with standard definitions of unification (for instance, see Sag and Wasow, 1999), unification implies checking the compatibility of feature values of the to–be–unified nodes (cf. agreement checks). If the values do not match, unification fails. Hierarchical syntactic structures derived by substitution are well–formed if and only if the phrasal foot nodes of all obligatory segments of the lexical frames involved successfully unify with the root of another frame. The hierarchy in Figure 1 is well–formed because the MODifier segments are not obligatory.

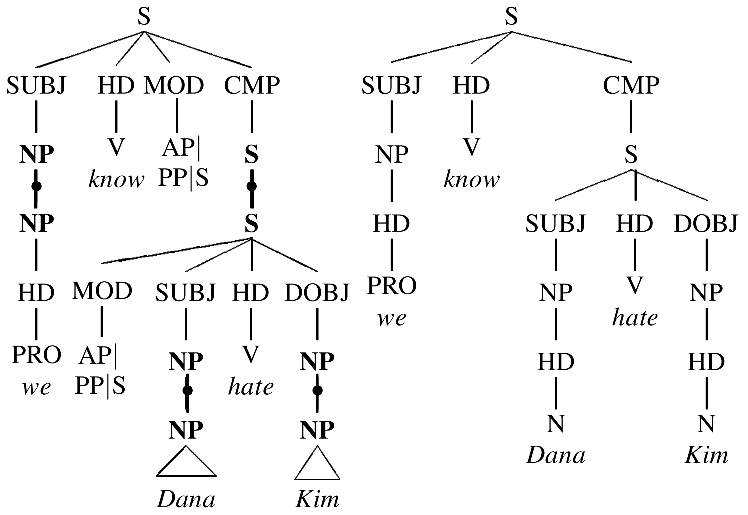


Figure 1: *Lefthand panel*: Simplified lexical frames underlying the sentences *We know Dana hates Kim* and *Kim we know Dana hates* (example from Sag and Wasow, 1999). Filled circles denote substitution and unification. (The feature matrices unified as part of the substitution operations are not shown.) *Righthand panel*: Hierarchy resulting after unification (merger) of foot and root nodes during substitution. Segments (branches) with phrasal leaves have been pruned. In both panels, order of segments is arbitrary.

2.3 Linear structure in PG

In order to assign a left–to–right order to the branches of hierarchical syntactic structures, PG uses so–called *linearization links* between categorial nodes in the frame hierarchy and positions in a *topology*, that is, a one–dimensional array of left–to–right *slots*. In this paper we will only be concerned with linearization links that assign a left–to–right order to the segments of verb frames (i.e., to the major constituents of finite and non–finite clauses). We propose that topologies of English and Dutch clauses contain exactly nine slots. Table 1 shows the slot names to be used in the topological features of the two target languages considered here. The slots labeled F*i* make up the Forefield (from Ger. *Vorfeld*); the M*j* slots belong to the Midfield (*Mittelfeld*); the E*k*s define the Endfield (*Nachfeld*; terms adapted from traditional German grammar; cf. Kathol, 2000). Table 2 illustrates which major clause constituents select which slot as their destination or 'landing site'. Notice, in particular, that the placement conditions refer not only to the grammatical function fulfilled by a constituent but also to its shape. E.g., while the Direct OBJect takes M3 as its default landing site, it selects F1 if it is a Wh–phrase or carries focus, and M2 if it is a personal pronoun (*it*). In terms of Figure 1, if *Kim* carries focus, it will occupy slot F1 of the topology associated with the complement clause headed by *hates*. In Section 3, we work out a formal method of assigning syntactic constituents to topology slots that is based on typed feature unification.

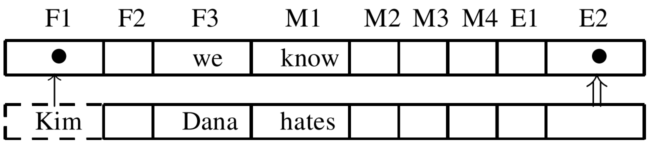
Table 1: Slot labels in clausal topologies.

English	F1	F2	F3	M1	M2	M3	M4	E1	E2
Dutch	F1	M1	M2	M3	M4	M5	M6	E1	E2

Table 2: Examples of topology slot fillers in English. MODifier constituents are not shown. Precedence between phrases landing in the same slot is marked by "<".

Slot	**Filler**
F1	(*Declarative main clause*: Topic or Focus) or (*Interrogative main clause*: Wh–constituent) or (*Complement clause*: Wh–constituent)—at most one constituent in F1
F2	*Complement clause*: CoMPLementizeR *that*
F3	SUBJect (if non–Wh)
M1	Pre–INFinitive *to* < HeaD verb (oblig.) < PaRTicle
M2	(*Interrogative main clause*: SUBJect (if non–Wh)) < (Direct OBJect (if personal pronoun))
M3	Indirect OBJect < (Direct OBJect (non–Wh))
M4	PaRTicle
E1	Non–finite Complement of 'Verb Raiser' (usually an Auxiliary)
E2	Non–finite Complement of 'VP Extra–position' verb or Finite Complement clause

How is the focussed Direct OBJect NP *Kim* 'extracted' from the subordinate clause and 'moved' into the main clause? Movement of phrases between clauses is due to *lateral topology sharing* (i.e. *left–* and/or *right–peripheral* sharing). If a sentence contains more than one verb, each of the verb frames concerned instantiates its own topology. This applies to verbs of any type, whether main, auxiliary or copula. In such cases, the topologies are allowed to *share* identically labeled lateral slots, conditionally upon several restrictions to be explained shortly. *After two slots have been shared, they are no longer distinguishable; in fact, they are the same object.* In the example of Figure 1, the embedded topology shares its F1 slot with the F1 slot of the matrix clause. This is indicated by the dashed borders of the bottom F1 slot:



Sharing F1 effectively causes the embedded focussed Direct Object *Kim* to move into slot F1 of the matrix, and thereby to open the sentence (black dot in F1 above the single arrow). The dot in E2 above the double arrow marks the position selected by the complement clause (or, rather, the root S–node of that clause). In order to avoid confusion with current terminology, we use the term *promotion* to denote the upward movement caused by lateral topology sharing.

The overt surface order is determined by a *Read–out module* that traverses the hierarchy of topologies in a left–to–right, depth–first manner. Any lexical item it 'sees' in a slot is appended to the output string and tagged as already processed. E.g., the Read–out module detects *Kim* while scanning the matrix topology and skips this NP during its traversal of the embedded topology. See Figure 2 for the ordered tree corresponding to *Kim we know Dana hates.*

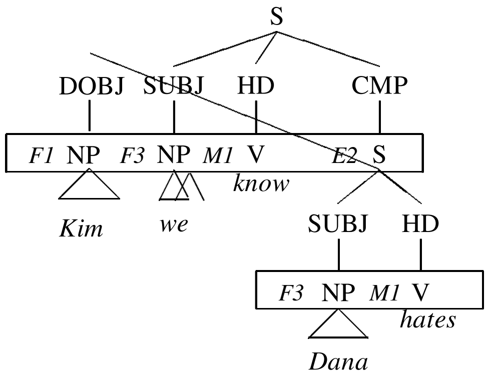


Figure 2: Fronting of Direct Object NP *Kim* due to promotion (cf. Figure 1).

The number of lateral slots an embedded topology shares with its upstairs neighbor is determined by the parameters *LS* (left–peripherally shared area) and *RS* (right–hand share). The two laterally shared areas are separated by a non–shared central area. The latter includes the slot occupied by the HeaD of the lexical frame (i.e., the verb in case of a verb frame) and possibly additional slots. The language–specific parameters *LS* and *RS* are defined in the lexical entries of complement–taking verbs, and dictate how the topology associated with the foot of S–CMP–S segments gets instantiated. For instance, the lexical entry for *know* (Figure 1) states that *LS*=1 if the complement clause is finite and declarative. This causes the two S–nodes of the CoMPlement segment to share one left–peripheral slot, i.e. F1. If the complement happens to be interrogative (as in *We know who Dana hates*), *LS*=0, implying that the F1 slots do not share content and *who* cannot escape from its clause. See Table 3 for the *LS* and *RS* values in English and Dutch. *LS* and *RS* are set to zero by default; this applies to the root S of main clauses and adverbial subordinate clauses.

In Figure 4, we show linearization at work on example (1), a Dutch sentence whose clause hierarchy is depicted in Figure 3. Table 4 lists the slots selected by various Dutch constituent types. Slot F1 of the main clause remains empty because this clause is a yes/no rather than a Wh–question. In the main clause, the HeaD verb goes to M1, in subordinate clauses to M6. The root S–node of the finite CoMPlement clause lands in E2. Within the finite complement, the CoMPlementizeR *dat* lands in slot M1 while the SUBJect selects M2. The non–finite complement of *heeft* opts for E1, its Direct OBJect chooses M3.

(1) *Denk je dat hij de auto heeft gerepareerd?*

Table 3: Size of the left– and right–peripheral shared topology areas (*LS* and *RS*) in diverse complement constructions. The notation ”4:6” indicates that the value is an integer between 4 (minimum) and 6 (maximum).

Clause type	**English**	**Dutch**
Interrogative	*LS*=0 RS=0	*LS*=0 RS=1
Declarative & Finite	*LS*=1 RS=0	*LS*=1 RS=1
Decl. & Non–Finite, VP Extraposition	*LS*=3 RS=0	*LS*=1 RS=1
Decl. & Non–Finite, Verb Raising	*LS*=3 RS=0	*LS*=4:6 RS=1
Decl. & Non–Finite, Third Construction	n.a.	*LS*=1:6 RS=1

think you that he the car has repaired
‘Do you think that he has repaired the car?’

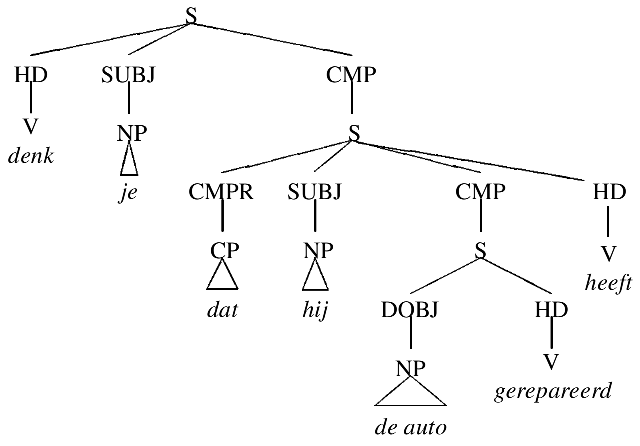


Figure 3: Verb frame hierarchy underlying example (1). The root S–node of the verb frame associated with *heeft* and the CoMPlement S–node of *denk* have merged as a result of unification, and so have the CMP–S of *heeft* and the root S–node dominating *gerepareerd*. Left–to–right order of branches is arbitrary.

3 Linear order constraints in typed feature unification

In our descriptions of typed feature unification in PG, we adopt terminology used in HPSG (for instance, see Sag and Wasow, 1999). Psycholinguistically motivated details such as non–destructive unification (cf. Vosse and Kempen, 2000) are not of interest here. Feature unification as outlined in Section 2.2 works in the usual manner and is not addressed any further.

Table 4: Examples of topology slot fillers in Dutch. Precedence between constituents landing in the same slot is marked by "<".

Slot	Filler
F1	(*Declarative main clause*: SUBJect, Topic or Focus) or (*Interrogative clause*: Wh–constituent) or (*Complement clause*: Wh–constituent)—exactly one constituent in F1
M1	(*Main clause*: HeaD verb) or (*Complement clause*: CoMPLementizeR *dat/om*)
M2	SUBJect NP (if non–Wh) < Direct OBJect (if personal pronoun)
M3	Direct < Indirect OBJect (if non–Wh)
M4	PaRTicle
M5	Non–finite CoMPlement of Verb Raiser
M6	*Subordinate clause*: Pre–INFinitive *te* < HeaD verb
E1	Non–finite Complement of Verb Raiser
E2	Non–finite Complement of VP Extraposition verb or Finite Complement

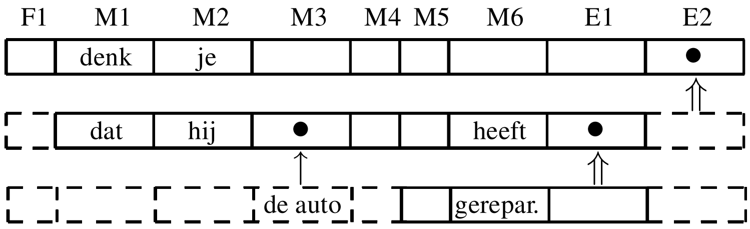


Figure 4: Linearization of example (1). The large left–peripherally shared area (*LS*=5) in the lower topology causes 'clause union' and promotion of Direct OBJect *de auto*.

The linearization method outlined in Section 2.3 consists of two parts. One part takes care of relations between clausal topologies, including lateral topology sharing. The other part deals with a clause–internal matter: the mapping from clause constituents onto the slots of the topology associated with the clause (*phrase–to–slot mapping*). We discuss these parts in turn.

3.1 Topologies as typed features

In this Section, we present a rule system for lateral topology sharing which is couched in a typed feature logic and deals with a broad variety of $\bar{A}$–movement phenomena in English and Dutch[1]. For each of these languages we define 9 slot types. They carry the labels introduced in Table 1 (e.g., *F1t, F2t, F3t, M1t, M2t,*

[1]A–movement (e.g. Subject–to–Subject and Subject–to–Object Raising) is dealt with in PG's hierarchical component and will not be discussed here. We refer to Harbusch and Kempen (2002) for a PG treatment of $\bar{A}$–movement in German.

M3t, M4t, E1t, E2t for English). Slots are attributes that take a list[2] of lemmas or constituents (e.g. HeaD–v, SUBJect–NP, CoMPlement–S) as their value. Slots are initialized with the value *empty list* (e.g., $[^{F1t}\ \mathrm{F1}\ \langle\rangle]$).

A slot type may impose a constraint on the *cardinality* (the number of members) of the list serving as its value. Cardinality constraints are expressed as subscripts of the value list. E.g., the subscript "c=1" in $[^{F1t}\ \mathrm{F1}\ \langle\rangle_{c=1}]$ states that the list serving as F1's value should contain exactly one member. Cardinality constraints are checked after all constituents that currently need a place have been appended.

We define a clausal topology as a list of slot types that serves as the value of the topology ("TPL") feature associated with S–nodes:

$$\mathrm{S}\ [\mathrm{TPL}\ \ \langle \mathrm{F1t, F2t, F3t, M1t, M2t, M3t, M4t, E1t, E2t}\rangle]$$

for English, and

$$\mathrm{S}\ [\mathrm{TPL}\ \ \langle \mathrm{F1t, M1t, M2t, M3t, M4t, M5t, M6t, E1t, E2t}\rangle]$$

for Dutch. Depending on the values of sharing parameters *LS* and *RS* (see Section 2.3), the list is divided into the left–peripheral area (comprising zero or more slot types), the central area (which includes at least one slot for the HeaD verb), and the right–peripheral area (possibly empty). Topology sharing is licensed exclusively to the lateral areas. *LS* and *RS* are set to zero by default; this applies to the root S of main clauses and adverbial subordinate clauses. The root S of a complement clause obtains its sharing parameter values from the foot of the S–CMP–S segment belonging to the lexical frame of its governing verb. For example, the lexical entry for *know* states that the complement of this verb should be instantiated with *LS*=1 if its clause type is declarative (CTYP=Decl). This causes the first slot (F1) of the topologies associated with the S–nodes in the S–CMP–S segment of *know*'s lexical frame to receive a coreference tag:

$$\begin{array}{cl}
\mathrm{S} & \begin{bmatrix}\mathrm{TPL} & \langle \boxed{1}\,\mathrm{F1, F2, ..., E2}\rangle\end{bmatrix} \\
| & \\
\mathrm{CMP} & \\
| & \\
\mathrm{S} & \begin{bmatrix}\mathrm{TPL} & \langle \boxed{1}\,\mathrm{F1, F2, ..., E2}\rangle \\ \mathrm{CTYP} & \mathrm{Decl}\end{bmatrix}
\end{array}$$

If, as in the example of Figure 1, *know*'s complement is indeed declarative, the foot of the complement segment can successfully unify with the root of the *hate* frame (the ensueing configuration is depicted in Figure 5 at the end of Section 3.2). As a consequence, the F1 slot of the complement clause is the same object as the F1 slot of the main clause, and any fillers will seem to have moved up one level in

[2]As for notation, list elements are surrounded by triangular brackets, and "⟨⟩" represents the empty list. Furthermore, we assume that topological (TPL) features are always instantiated with the complete list of empty slots, and that unification of two TPL features succeeds if their list of attributes (slot names) is identical. In case of successful unification, the values of the corresponding slot attributes are combined by the *append* operation, which is rendered by the symbol "∘". The expression "L1 ∘ L2" represents the list composed of the members of L1 followed by the members of L2. If L1 is the empty list, "L1 ∘ L2" evaluates to L2.

the clause hierarchy. The result is left–peripheral topology sharing of the first slot, with promotion of its content, as depicted in Figure 2.

3.2 Phrase–to–slot mapping

How does PG set up the linearization links that enable the constituents of a clause to access a legitimate landing site in the topology associated with that clause? To illustrate, consider the placement of non–pronominal SUBJect and Direct OBJect NPs, and of finite and non–finite verbs in Dutch. (The real constraints are more complex than rendered here.) As already indicated in Table 4, the SUBJect NP[3] lands in F1 or M2:

$$\begin{array}{ll} \mathrm{S} & \left[\mathrm{TPL}\quad \left\{\langle \mathrm{F1} \circ \boxed{1}, \mathrm{M1}, \ldots, \mathrm{E2}\rangle, \langle \mathrm{F1}, \mathrm{M1}, \mathrm{M2} \circ \boxed{1}, \ldots, \mathrm{E2}\rangle\right\}\right] \\ | & \\ \mathrm{SUBJ} & \\ | & \\ \mathrm{NP} & \left[\mathrm{TPL}\quad \boxed{1}\right] \end{array}$$

The tokens of coreference tag $\boxed{1}$ following symbol "$\circ$" in slots F1 and M2 license appending the entire feature complex of the SUBJect NP to the current content of these slots. (If F1 has remained empty since initialization, $\mathrm{F1} \circ \boxed{1} = \langle\rangle \circ \boxed{1} = \boxed{1}$.) The choice between the two landing sites will be made by the Read–out module at a later point in time. The full Direct OBJect NP selects M3. However, when this NP carries focus, F1 is an additional option:

$$\begin{array}{ll} \mathrm{S} & \left\{\left[\mathrm{TPL}\quad \left\{\langle \mathrm{F1} \circ \boxed{1}, \ldots\rangle, \langle \ldots, \mathrm{M3} \circ \boxed{1}, \ldots\rangle\right\}\right], \left[\mathrm{TPL}\quad \langle \ldots, \mathrm{M3} \circ \boxed{1}, \ldots\rangle\right]\right\} \\ | & \\ \mathrm{DOBJ} & \\ | & \\ \mathrm{NP} & \begin{bmatrix} \mathrm{FOC} & + \\ \mathrm{TPL} & \boxed{1} \end{bmatrix} \end{array}$$

The finite HeaD verb selects slot M1 in main clauses, M6 in subordinate clauses:

$$\begin{array}{ll} \mathrm{S} & \left\{\begin{bmatrix} \mathrm{CTYP} & \mathrm{MainCl} \\ \mathrm{TPL} & \langle \ldots, \mathrm{M1} \circ \boxed{1}, \ldots\rangle \end{bmatrix}, \begin{bmatrix} \mathrm{CTYP} & \mathrm{SubCl} \\ \mathrm{TPL} & \langle \ldots, \mathrm{M6} \circ \boxed{1}, \ldots\rangle \end{bmatrix}\right\} \\ | & \\ \mathrm{HD} & \\ | & \\ \mathrm{V} & \left[\mathrm{LEMMA}\quad \boxed{1}\right] \end{array}$$

[3]Every phrasal node has its own TPL feature, with its own list of slot names. Due to space limitations, we cannot address the ordering of constituents of NPs, PPs, etc. Furthermore, in the graphical representation of feature matrices, feature values printed between curly brackets and separated by commas denote alternative (disjunctive) options.

Here, the coreference tag forces the slot choice immediately upon initialization: If the segment belongs to a main clause, the TPL feature of the root node is initialized only with the first of the two placement alternatives; in case of a subordinate clause, only the second alternative survives.

Suppose now that the three segments we have just seen (SUBJ, DOBJ and HD) belong to the same main clause, and that the Direct OBJect is not focussed. In that case, the final position of the SUBJect segment is uniquely determined: Although it has outgoing linearization links to F1 as well as M2, only SUBJ–HD–DOBJ order will yield a successful derivation due to the cardinality constraint of F1 (exactly one phrase landing there). If the Direct OBJect does carry focus, it also has two outgoing linearization links (to F1 and M3). Now there are two ways to comply with F1's cardinality constraint: not only SUBJ–HD–DOBJ but also its mirror image DOBJ–HD–SUBJ (cf. *Jan wil dit boek* 'Jan wants this book' and *Dit boek wil Jan* 'This book Jan wants').

In order to illustrate the combined effect of topology sharing and phrase–to–slot mapping, we return to the English example of Figure 1. Figure 5 depicts the structure resulting under the assumption that the Direct OBJect is focussed. Notice that the feature structure associated with the CoMPlement–S node leaves *Kim* two placement options. We assume the Read–out module selects one alternative on the basis of pragmatic, discourse and other context factors. For instance, the focussed Direct OBJect may prefer to go to slot M3 in spoken output where focussed elements can be accented; in written output, fronting of focussed elements may be preferable. In incremental generation mode, fronting of a focussed Direct OBJect may be ruled out if Read–out already has advanced beyond slot F1.

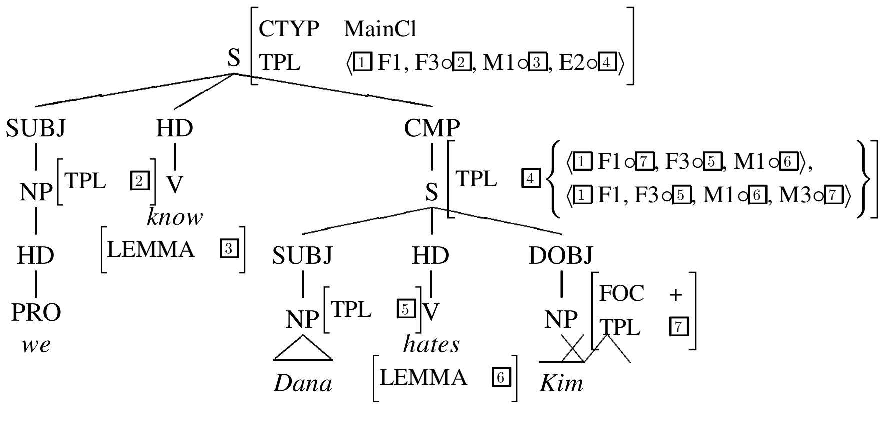


Figure 5: Unification–based lateral topology sharing and phrase-to-slot mapping applied to the example of Figure 1. Empty slots in the TPL features are not shown.

4 Comparison with other grammar formalisms

In the introductory Section we highlighted two general properties of human sentence production and discussed the requirements they impose on the design of a grammar formalism that claims psychological plausibility. Actually, grammar formalisms of linguistic origin rarely deal with psychological (performance) issues in any detail, concentrating on the description and explanation of linguistic competence. However, considerations of theoretical parsimony justify assessing the extent to which existing linguistic grammar formalisms in fact meet psychological demands. For, if one or more formalisms would receive a positive evaluation, no additional formalism would be needed. In this Section we therefore apply the performance criteria of late linearization and incrementality to mainstream linguistic models of grammar.

Many formalisms employ phrase–structure rules, whether or not supplemented by transformations. Because these rules conflate hierarchical and linear structure, they are at variance with the requirement of late linearization. Converting phrase–structure rules into ID/LP format in order to separate hierarchal (Immediate Dominance, ID) and linear (Linear Precence, LP) aspects is an important step forward, unless other properties of the grammar preclude *late* assignment of linear order. Reliance on transformations that presuppose and modify left–to–right order of phrases is one such property. Non–transformational grammars, in particular Categorial Grammars (CG), GPSG/HPSG, and Tree Adjoining Grammars (TAG), are therefore more likely candidates to evolve into psychological models of grammar.

Non-transformational grammars have the additional advantage of being more easily amenable to incremental production, at least insofar as transformations presuppose syntactic trees that dominate *complete* sentences. Incremental syntactic generators have indeed been developed on the basis of HPSG (e.g., Neumann and Finkler, 1990 and Abb *et al.*, 1993) and TAG (e.g., Harbusch *et al.*, 1991).

TAG, HPSG and CG have also been advocated as meeting important psychological requirements (e.g., Ferreira, 2000, on TAG; Sag and Wasow, 1999, Chapter 9 on HPSG as a "Realistic Grammar"; Steedman, 2000, on CG). Kathol (2000) developed an HPSG variant called "Linear Syntax", which comes close to PG (although Kathol does not make any psychological claims). It has separate separate hierarchical and linearization components, the latter based on topological fields.

However, although these models can generate sentences incrementally, they lack provisions against premature utterance release. Kathol's linearization method is exlusively based on linear precedence rules, i.e. on relative positions (*o.c.*, p. 76–80). The TAG–based incremental generator developed by Harbusch *et al.* does include provisions against premature utterance release but in the form of a special control mechanism built around the grammar. In PG, such provisions are an integral part of the formalism: The topologies provide placeholders for all types of constituents looking for a landing site; slots serving as placeholder for obligatory constituents carry a cardinality tag so that the Read–out module cannot skip them if the obligatory filler is missing.

We conclude that PG complies better with the two general sentence production constraints addressed in this paper than any of the formalisms of linguistic origin.

5 Discussion

Elsewhere, we have presented detailed accounts of complex and puzzling linear order phenomena in Dutch and German, in particular verb clustering in Dutch and German, and scrambling in German (see two forthcoming papers by Kempen and Harbusch). We have shown, furthermore, that striking contrasts between $\bar{\mathrm{A}}$–movement phenomena in English, Dutch and German reduce to different settings of a few numerical parameters that control lateral topology sharing (Harbusch and Kempen, 2002). Space limitations prevent us from seriously discussing PG's additional psycholinguistic virtues; but see Vosse and Kempen (2000) for a dynamic model of human syntactic parsing built around a very similar formalism.

As already pointed out above, the choice between ordering options left open by PG's hierarchical component is made by the Read–out module. So far we have assumed that this component processes the slots of a topology strictly from left to right. However, it seems reasonable to endow it with a limited amount of preview, thereby enabling it to base its sequencing decisions on the contents of slots further down the topology. For instance, consider the placement of particles in English, which according to Table 2 are free to land in M1 (immediately after the HeaD verb) or in M4 (after the Direct Object). This accounts for the acceptability of both *She called up her boyfriend* and *She called her boyfriend up*. However, if the Direct OBJect is an unstressed personal pronoun, it should precede the particle (**She called up him*). We therefore suggest to delegate to the Read–out module linearization decisions that depend on constituent length/weight.

We realize that the division of linearization tasks between the hierarchical component and the Read–out module is far from clear, and that further empirical (psycho–)linguistic data will rule on this matter. The important points to be stressed here are, first, that PG's hierarchical component does not output linearized structures and, second, that the hierarchical and Read–out components are in a master–slave relationship. At all times, the latter should operate within the space of linearization options delineated by the former.

Acknowledgement

We are indebted to Camiel van Breugel (Leiden Institute for Advanced Computer Science) who implemented in JAVA a *Performance Grammar Workbench (PGW)* which runs as an applet within modern web browsers. At the time of writing (June 2002), a Dutch PGW version is available on the internet (`http://www.liacs.nl/home/cvbreuge/pgw/`).

References

Abb, B., Günther, C., Herweg, M. and Lebeth, K. (1993), Incremental syntactic and phonological encoding — An outline of the SYNPHONICS formulator, *Technical report*, University of Hamburg, Hamburg, Germany.

Bock, J. K. and Levelt, W. J. (1995), Language production: Grammatical encoding, *in* M. Gernsbacher (ed.), *Handbook of Psycholinguistics*, Academic Press, New York, NY, USA, pp. 945–984.

Ferreira, F. (2000), Syntax in language production: An approach using tree-adjoining grammars, *in* L. Wheeldon (ed.), *Aspects of Language Production*, MIT Press, Cambridge, MA, USA.

Garrett, M. F. (1975), The analysis of sentence production, *in* G. Bower (ed.), *The psychology of learning and motivation*, Academic Press, New York, NY, USA, pp. 133–177.

Gazdar, G., Klein, E., Pullum, G. K. and Sag, I. (1985), *Generalized Phrase Structure Grammar*, Harvard Press, Cambridge, MA, USA.

Harbusch, K. and Kempen, G. (2002), A quantitative model of word order and movement in English, Dutch and German complement constructions, *Proceedings of the 19th International Conference on Computational Linguistics (COLING-2002)*, Taipei, Taiwan.

Harbusch, K., Finkler, W. and Kilger, A. (1991), Incremental syntax generation with tree adjoining grammars, *Proceedings of the "Fourth International GI Congress: Knowledge–based Systems - Distributed Artificial Intelligence"*, Springer, Heidelberg, Germany, pp. 363–374.

Joshi, A. K. and Schabes, Y. (1997), Tree Adjoining Grammars, *in* G. Rozenberg and A. Salomaa (eds), *Handbook of Formal Languages*, Vol. 3, Springer, Berlin, Germany, pp. 69–214.

Kathol, A. (2000), *Linear Syntax*, Oxford University Press, New York, NY, USA.

Kempen, G. and Harbusch, K. (2002a), Dutch and German verb clusters in Performance Grammar, *in* P. Seuren and G. Kempen (eds), *Verb clusters in Dutch and German*, Benjamins, Amsterdam, The Netherlands. Forthcoming.

Kempen, G. and Harbusch, K. (2002b), Word order scrambling as a consequence of incremental sentence production, *in* H. Härtl, S. Olsen and H. Tappe (eds), *The syntax–semantics interface: Linguistic structures and processes*, De Gruyter, Berlin, Germany. In press.

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

Neumann, G. and Finkler, W. (1990), Head–Driven Incremental and Parallel generation, *Proceedings of the 13th International Conference on Computational Lingustics (COLING–90)*, Vol. 2, Helsinki, Finnland, pp. 288–293.

Sag, I. A. and Wasow, T. (1999), *Syntactic theory: a formal introduction*, CSLI Publications, Stanford, CA, USA.

Steedman, M. (2000), *The Syntactic Process*, MIT Press/Bradford Books, Cambridge, MA, USA.

Vosse, T. and Kempen, G. (2000), Syntactic structure assembly in human parsing: A computational model based on competitive inhibition and lexicalist grammar, *Cognition 75*, 105–143.