A generation-oriented workbench for Performance Grammar: Capturing linear order variability in German and Dutch

Karin Harbusch[1], Gerard Kempen[2,3], Camiel van Breugel[3], Ulrich Koch[1]

[1]Universität Koblenz-Landau, Koblenz
{harbusch, koch}@uni-koblenz.de

[2]Max Planck Institute for Psycholinguistics, Nijmegen
gerard.kempen@mpi.nl

[3]Department of Psychology, Leiden University
cvbreuge@liacs.nl

Abstract

We describe a generation-oriented workbench for the Performance Grammar (PG) formalism, highlighting the treatment of certain word order and movement constraints in Dutch and German. PG enables a simple and uniform treatment of a heterogeneous collection of linear order phenomena in the domain of verb constructions (variably known as Cross-serial Dependencies, Verb Raising, Clause Union, Extraposition, Third Construction, Particle Hopping, etc.). The central data structures enabling this feature are clausal "topologies": one-dimensional arrays associated with clauses, whose cells ("slots") provide landing sites for the constituents of the clause. Movement operations are enabled by unification of lateral slots of topologies at adjacent levels of the clause hierarchy. The PGW generator assists the grammar developer in testing whether the implemented syntactic knowledge allows all and only the well-formed permutations of constituents.

1 Introduction

Workbenches for natural-language grammar formalisms typically provide a parser to test whether given sentences are treated adequately — D-PATR for Unification Grammar (Karttunen, 1986) or XTAG for Tree-Adjoining Grammars (Paroubek *et al.,* 1992) are early examples. However, a parser is not a convenient tool for checking whether the current grammar implementation licenses all and only the strings qualifying as well-formed expressions of a given input. Sentence generators that try out all possible combinations of grammar rules applicable to the current input, are better suited.

Few workbenches in the literature come with such a facility. LinGO (Copestake & Flickinger, 2000), for Head-Driven Phrase Structure Grammar, provides a generator in addition to a parser. For Tree Adjoining Grammars, several workbenches with generation components have been built: InTeGenInE (Harbusch & Woch, 2004) is a recent example.

Finetuning the grammar such that it neither over- nor undergenerates, is a major problem for semi-free word order languages (e.g., German; cf. Kallmeyer & Yoon, 2004). Working out a satisfactory solution to this problem is logically prior to designing a generator capable of selecting, from the set of all possible paraphrases, those that sound "natural," i.e., the ones human speakers/writers would choose in the situation at hand (cf. Kempen & Harbusch, 2004).

Verb constructions in German and Dutch exhibit extremely intricate word order patterns (cf. Seuren & Kempen, 2003). One of the factors contributing to this complexity is the phenomenon of *clause union*, which allows constituents of a complement clause to be interspersed between those of the dominating clause. The resulting sequences exhibit, among other things, cross-serial dependencies and clause-final verb clusters. Further complications arise from all sorts of 'movement' phenomena such as fronting, extraction, dislocation, extraposition, scrambling, etc. Given the limited space available, we cannot describe the Performance Grammar (PG) formalism and the linearization algorithm that enables generating a broad range of linear order phenomena in Dutch, German, and English verb constructions. Instead, we refer to Harbusch & Kempen (2002), and Kempen & Harbusch (2002, 2003).

Here, we present the generation-oriented PG Workbench (PGW), which assists grammar developers, among other things, in testing whether the implemented syntactic and lexical knowledge allows all and only well-formed permutations.

In Section 2, we describe PG's topology-based linearizer implemented in the PGW generator, whose software design is sketched in Section 3. Section 4 shows the PGW at work and draws some conclusions.

2 Linearization in PG and PGW

Performance Grammar (PG) is a fully lexicalized grammar that belongs to the family of tree substitution grammars and deploys disjunctive feature unification as its main structure building mechanism. It adheres to the ID/LP format (Immediate Dominance vs. Linear Precedence) and includes separate components generating the hierarchical and the linear structure of sentences. Here, we focus on the linearization component.

PG's hierarchical structures consist of unordered trees composed of elementary building blocks called *lexical frames*. Every word is head of a lexical frame, which specifies the subcategorization constraints of the word. Associated with every lexical frame is a *topology*. Topologies serve to assign a left-to-right order to the branches of lexical frames. In this paper, we will only be concerned with topologies for verb frames (clauses). We assume that clausal topologies of Dutch and German contain exactly nine slots — see (1).

(1) Wat wil je dat ik doe? / what want you that I do /'What do you want me to do?'

F1	M1	M2	…	M6	E1	E2
●	wil	je				●
↑						⇑
Wat	dat	ik			doe	

The slot labeled F1 makes up the Forefield (from Ger. *Vorfeld*); slots M1-M6 make up the Midfield (*Mittelfeld*); slots E1 and E2 define the Endfield (*Nachfeld*). Every constituent (subject, head, direct object, complement, etc.) has a small number of placement options, i.e. slots in the topology associated with its "own" clause.

How is the Direct Object NP *wat* 'what' 'extracted' from the complement clause and 'promoted' into the main clause? Movement of phrases between clauses is due to *lateral topology sharing*. If a sentence contains more than one verb, each of their lexical frames instantiates its own topology. This applies to verbs of any type — main, auxiliary or copula. In such cases, the topologies are allowed to *share* identically labeled lateral (i.e. left- and/or right-peripheral) slots, conditionally upon several restrictions (not to be explained here; but see Harbusch & Kempen, 2002)). *After two slots have been shared, they are no longer distinguishable; in fact, they are unified and become the same object.* In example (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 the F1 slots effectively causes the embedded Direct Object *wat* to be *preposed* into the main clause (black dot in F1 above the single arrow in (1)). The dot in E2 above the double arrow marks the position selected by the finite complement clause.

The overt surface order is determined by a *read-out module* that traverses the hierarchy of topologies in left-to-right, depth first manner. E.g., *wat* is already seen while the reader scans the higher topology.

3 A sketch of PGW's software design

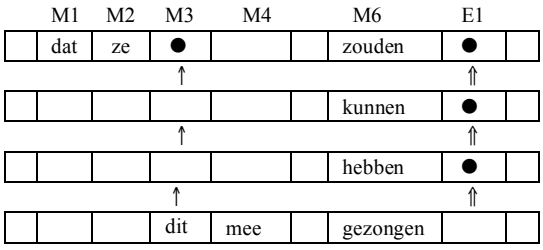
The PGW is a computational grammar development tool for PG. Written in Java, it comes with an advanced graphical direct-manipulation user interface. All lexical and grammatical data have been encoded in a *relational database* schema. This contrasts with the predominance of *hierarchical databases* in present-day computational linguistics. Relational lexical databases tend to be easier to maintain and update than hierarchical ones, especially for linguists with limited programming experience. The software was designed with an eye toward easy cross-language portability of the encoded information. For German we developed a *lexicon converter* that maps the German *CELEX* database automatically to the PGW format (Koch, 2004).

4 Generating verb constructions in Dutch and German

In order to convey an impression of the capabilities of the PGW, we show it at work in generating verb constructions that involve rather delicate linearization phenomena: "Particle Hopping" in Dutch (2), and "Scrambling" in German (3).

The finite complement clause (2) includes the verb *meezingen* 'sing along with,' where *mee* 'with' is a preposition functioning as separable particle. The three other verbs are auxiliaries. According to a topology sharing rule for Dutch, clauses headed by auxiliaries are free to share 4, 5 or 6 left-peripheral slots of their own topology with that of its complement. The most restrictive sharing option is shown in (2).

(2) *... dat ze dit (lied) zouden kunnen hebben meegezongen*
... that they this (song) would be-able-to have along-sung
'... that they might have sung along this (song)'



The Direct Object NP *dit (lied)* 'this (song)' lands in M3 of the lowest topology. As this slot belongs to the four left-peripheral ones, it is always shared and its content gets promoted all the way up into the highest clause (see single arrows). Particle *mee* always lands in the fifth slot (M4), i.e. in the optionally shared area. Hence, its surface position depends on the actual number of shared left-peripheral slots. In (2), with minimal slot sharing, *mee* stays in its standard position immediately preceding the head verb. In case of non-minimal topology sharing, the particle may move leftward until (but no farther than) the direct object, thus yielding exactly the set of grammatical placement options.

The quality of PGW's treatment of Scrambling in German can be assessed in terms of a set of 30 word order variations of sentence (3), discussed by Rambow (1994), who also provides grammaticality ratings for all members of the set. Seuren (2003) presents similar grammaticality judgments obtained from an independent group of native speakers. As the rating scores appeared to vary considerably (cf. (3a) and (3b)), we checked which permutations are actually generated by the PGW. It turned out easy to find a set of topology sharing values that generates all and only the paraphrases with high or satisfactory grammaticality scores.

In conclusion, although the evaluation data are very limited as yet, we believe they justify positive expectations with respect to the potential of a topology based linearizer to approximate closely the grammaticality judgements of native speakers and thus to avoid over- and undergeneration.

(3) a. *... weil niemand das Fahrrad zu reparieren zu versuchen verspricht*
because nobody the bike to repair to try promises
'… because nobody promises to try to repair the bike'
b. **...weil zu versuchen das Fahrrad niemand zu reparieren verspricht*

References

Copestake, A. and Flickinger, D. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. *Procs. of the 2nd Internat. Conf. on Language Resources and Evaluation (LREC),* Athens.

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

Harbusch, K. and Woch, J. 2004. Integrated natural language generation with Schema-TAGs. In: Habel, C., Pechmann, T. (eds.) *Language Production.* Berlin: Mouton De Gruyter.

Karttunen, L. 1986. D-PATR: A Development Environment for Unification Grammars. Tech. Rep. CSLI-86-61. CSLI, Stanford, CA.

Kempen, G. and Harbusch, K. 2002 Performance Grammar: A declarative definition. In Nijholt, A., Theune, M., Hondorp, H. (eds.), *Computational Linguistics in the Netherlands 2001.* Amsterdam: Rodopi, 146-162.

Kempen, G. and Harbusch, K. 2003. Dutch and German verb constructions in Performance Grammar. In Seuren & Kempen, 2003.

Koch, U. 2004. *The Specification of a German Performance Grammar from CELEX Data in Preparation for a German Performance Grammar Workbench*, Master's Thesis at the Computer Science Department, U. Koblenz-Landau.

Paroubek, P., Schabes, Y., and Joshi, A.K. 1992. XTAG – A Graphical Workbench for Developing Tree-Adjoining Grammars. *Procs. of the 3rd Applied Natural Language Processing Conference (ANLP)*, Trento, 216-223.

Rambow, O. 1994. *Formal and Computational Aspects of Natural Language Syntax.* Ph.D. Thesis, U. Pennsylvania, Philadelphia.

Seuren, P.A.M. 2003. Verb clusters and branching directionality in German and Dutch. In (Seuren & Kempen, 2003), 247-296.

Seuren, P. and Kempen, G. (eds.). 2003. *Verb Constructions in Dutch and German.* Amsterdam: Benjamins.