Annotation, storage, and retrieval of mildly recursive structures

Stefan Evert, Hannah Kermes
Institut für Maschinelle Sprachverarbeitung,
Azenbergstr. 12, 70174 Stuttgart, Germany
{evert,kermes}@ims.uni-stuttgart.de

1 Introduction

This paper describes an unusual approach to the partial syntactic analysis of unrestricted German text. Unlike most other chunk parsers, which are specially designed and implemented for the single purpose of annotating syntactic structures, YAC is the result of a slow evolution from on-line to off-line analysis. As we formulated increasingly complex queries in the CQP query language, some of which had to recognise nested syntactic structures and test morpho-syntactic agreement within noun phrases, we felt the need to "write back" the detected structures into the corpus for re-use in further queries.

After three years of development, YAC has become a versatile and robust chunk parser. Technologically, it is still based on the IMS Corpus Workbench and its query processor CQP. The YAC grammar consists of rules written in the CQP query language, similar to a context-free grammar without recursion (although the CQP query that makes up the "body" of a rule is by far more powerful than the right-hand side of a production in a context-free grammar), complemented by Perl scripts used for additional filtering and post-processing of the query results. The non-recursive rules can be applied repeatedly to gradually build up hierarchical structures. Although this approach is similar in principle to a cascaded finite-state parser (Abney 1996b), the CQP query language and especially the Perl code used in post-processing are more expressive, and quite often more convenient than standard finite-state technology. Other chunk parsers for German and their relation to YAC are discussed in (Kermes and Evert 2003) and (Kermes 2003).

(Kermes and Evert 2003) give a general overview of the architecture of YAC, as well as the annotated structures and additional features. They also discuss the particular advantages of the parser's design (e.g. that all grammar rules can be executed as on-line queries for debugging purposes) and sketch some applications of the richly annotated chunk analyses. This paper, on the other hand, is mainly concerned with the technical details of YAC's implementation

Section 2 introduces the IMS Corpus Workbench, and discusses its strengths and limitations. Section 3 describes the slow evolution from a collection of corpus queries to a robust and powerful chunk parser. Section 4 discusses the mildly recursive nature of German noun chunks and consequences for the representation and grammar formalism. A detailed description of the YAC implementation can be found in Section 5. Section 6 describes the XML output format used by YAC and summarises some extensions to the CWB for the storage, retrieval, and display of mildly recursive structures.

2 The IMS Corpus Workbench

The IMS Corpus Workbench (CWB) is a highly specialised database software for large text corpora. When it was originally designed in the early 1990's, the main goal was to provide efficient access to text collections of 100 million words and above on the computer hardware that was then available (Christ 1994). This emphasis on performance and efficient memory usage led to a read-only, token-based data model with a static word form lexicon and index. The data model allows an arbitrary number of token-level annotations (*positional attributes*, or *p-attributes* for short), each of which has its own (static) type lexicon and index. Limited support for non-recursive document-structure markup (sentences, paragraphs, documents, etc.) is provided by *structural attributes* (*s-attributes* for short), and parallel corpora can be aligned at the sentence level or a similar granularity. Corpora are stored in a platform-independent binary format, which applications can access directly using memory-mapping techniques. Thus data files need not be loaded into memory, reducing start-up times and ensuring optimal performance even when the data size of a corpus exceeds the amount of physical memory.

A key advantage of the CWB are its compression techniques for p-attributes, which reduce the data size of each attribute by 50% and more. The sequence of tokens or token-level annotations represented by a p-attribute is Huffman coded, using frequency information from the attribute's fixed lexicon. For the lookup index, a special variable-length encoding is used, which assumes that the instances of a

particular lexicon entry are evenly distributed across the corpus. Typical data size after compression is approx. 30 bits/token for lexical attributes (characterised by a large lexicon with many low-frequency entries) such as the tokens themselves and the lemma annotations. Categorical attributes with a small, fixed inventory of annotated codes (examples are part-of-speech tags and morpho-syntactic agreement features) are compressed more efficiently and require approx. 10 bits/token. Binary attributes, which encode yes/no distinctions, are usable only when compressed to a size of 4 bits/token. To flesh out these numbers, consider a 100-million word corpus. Every uncompressed p-attribute requires some 790 Mbytes of disk space (which need to be loaded into memory, at least temporarily, when a query is processed). Compression reduces this size to approx. 360 Mbytes for a lexical attribute, 120 Mbytes for a categorical attribute, and only 50 Mbytes for each binary attribute. The maximal size of a CWB corpus is dictated by technical constraints of the 32-bit operating systems for which it was designed. Depending on the number of attributes and their compression rates, corpus sizes of up to 500 million tokens are feasible. Without compression, the limit would be close to 100 million tokens instead.

The central component of the CWB is an interactive corpus query processor (CQP), which displays its KWIC-like[1] output in a terminal window. Alternatively, the query results can be saved to text files in one of several output formats, including HTML. In addition to CQP, the CWB includes several utility programs for the encoding (i.e. conversion into the binary CWB format), indexing, and compression of textual input, as well as tools that extract lexical and distributional information from a CWB-encoded corpus. Direct access to corpora in the binary CWB format is provided by an undocumented C library.

```
<text id=42 lang="English">
<s>An easy example.</s>
<s>Just the easiest examples!</s>
</text>
```

Figure 1. Example of text with document-structure markup (XML).

corpus position	word (form)	pos	lemma
(0)	<text> “id=42 lang="English"”		
(0)	<s>		
0	An 0	DET 0	a 0
1	easy 1	ADJ 1	easy 1
2	example 2	NN 2	example 2
3	. 3	PUN 3	. 3
(3)	</s>		
(4)	<s>		
4	Just 4	ADV 4	just 4
5	the 5	DET 0	the 5
6	easiest 6	ADJ 1	easy 1
7	examples 7	NN 2	example 2
8	! 8	PUN 3	! 6
(8)	</s>		
(8)	</text>		

Figure 2. The tabular CWB corpus format (with token-level annotations)

Figure 1 gives a short, but typical example of textual input with document-structure markup in the form of XML tags. When this text is encoded as a CWB corpus, it is transformed into a tabular format as sketched in Figure 2. In this format, rows correspond to tokens and columns to token-level annotations, i.e. positional attributes. The token itself, i.e. the surface word form, is treated as a token-level annotation and encoded in the special “word” attribute. (The small numbers in Figure 2 indicate lexicon IDs for each p-attribute.) Each token is identified by a unique *corpus position*, starting from 0 for the first token in the corpus. XML tags are recognised by the encoding tool and converted into structural attributes, which can be thought of as sequences of zero-width start and end tags interspersed with the token sequence. These tags attach to the following or preceding tokens, respectively, and are shown as additional rows in Figure 2 (with the “implied” corpus position in parentheses). XML attributes in start tags can optionally be annotated as a single string value (as shown for the <text> tag in the first row), or parsed into attribute-value pairs (this recent improvement is briefly described in Section 6.3).

The CQP query language uses regular expressions over token descriptions (*patterns*) to model contiguous sequences of tokens. A *pattern* specifies constraints on the annotations (i.e. p-attributes) of matching tokens. All annotations are treated as strings, which can be matched literally, against a list of fixed values (read from a file), or against a character-level regular expression, which follows the POSIX standard and supports case- and diacritic-insensitive matching. XML-style tags matching the start and end points of s-attribute regions can be freely interspersed with token patterns in a CQP query.

[1] KWIC stands for “key word in context”, the output format of choice for most applications in lexicography.

Labels can be assigned to patterns and refer to (the corpus position of) the matching token, so that additional constraints can be specified that compare the annotations of different tokens. Queries like

(1) `a:[pos = "NN"] [pos="APPR"] b:[pos="NN"] :: a.lemma = b.lemma;`

which matches two instances of the same noun separated by a preposition (e.g. *Tag für Tag*; day after day) extend the expressiveness of the query language beyond that of ordinary regular expressions. All features of the CQP query language and further "interactive" commands of CQP are described in (Evert 2003), which is written in the form of a tutorial with numerous example queries.

Binary versions of the IMS Corpus Workbench are available free of charge for non-commercial use. Currently, supported platforms are SUN Solaris 8 on SPARC processors and Linux (Kernel 2.4 or newer) on Intel i386-compatible processors. Further details and registration information can be found on the CWB homepage at http://www.ims.uni-stutgart.de/projekte/CorpusWorkbench/ .

Many recent developments of the CWB were inspired by the work on YAC. In addition to the improved XML support, this includes string-replacement macros with up to 10 interpolated arguments, various convenience features in CQP (command-line editing, better highlighting of query results, progress information, and the possibility to interrupt slow queries), and feature set attributes. Feature sets encode ambiguous annotation values (e.g. part-of-speech ambiguities not resolved by the tagger, or ambiguous lemmatisation possibilities) in a special string notation, delimited by | characters. The new comparison operators `contains` and `matches` test whether a feature set contains a particular value and match all elements of the set against a regular expression, respectively. Features set have proven particularly useful for the annotation of morpho-syntactic features in German and other inflectional languages. Example (2) shows a short German noun phrase annotated with the morpho-syntactic information relevant for agreement within the NP. For each token, all possible combinations of case, gender, and number (as determined by a morphological analyser) are combined into a feature set.

(2) a. den |*Akk:M:Sg*|Dat:F:Pl|Dat:M:Pl|Dat:N:Pl|
b. vierten |Akk:F:Pl|Akk:M:Pl|*Akk:M:Sg*|Akk:N:Pl
|Dat:F:Pl|Dat:F:Sg|Dat:M:Pl|Dat:M:Sg|Dat:N:Pl|Dat:N:Sg
|Gen:F:Pl|Gen:F:Sg|Gen:M:Pl|Gen:M:Sg|Gen:N:Pl|Gen:N:Sg
|Nom:F:Pl|Nom:M:Pl|Nom:N:Pl|
c. Platz |*Akk:M:Sg*|Dat:M:Sg|Nom:M:Sg|

In this case, the head noun *Platz* is ambiguous between nominative, dative, and accusative case. The determiner is also ambiguous between dative and accusative. However, when the entire NP is taken into account, the only consistent analysis is accusative, masculine, singular (shown in italics). This information can be obtained in a CQP query through a combination of labels, a fast built-in function that computes the intersection of feature sets, and the `matches` comparison operator.

3 From on-line to off-line analysis

The work that led up to the implementation of YAC began with an effort to improve the CQP queries used by (Eckle 1999) to extract evidence for verb subcategorisation frames, and write similar queries for other lexical and syntactic phenomena. Due to the complexity of German noun phrases (cf. the example below), the generalised queries quickly became highly complicated and inconveniently long.

(3)

APPR	ADJA	$,	APPR	ART	NN	ART	NN	ADJA	NN
mit	kleinen	,	über	die	Köpfe	der	Apostel	gesetzten	Flammen
with	small		above	the	heads	of the	apostles	set	flames

'with small flames set above the heads of the apostles'

(4)
```
[pos="APPR"] [pos="ART"]?
( ( ( [pos="APPR"] [pos="ART"]?
      ( [pos="ADJA"] ("," [pos="ADJA"])* )? [pos = "NN|NE"]+
      ( [pos="ART"]? ([pos="ADJA"] ("," [pos="ADJA"])*)? [pos="NN|NE"]+ )* )*
    [pos = "ADJA"] )
  ( ","
    ( [pos="APPR"] [pos="ART"]?
      ( [pos="ADJA"] ("," [pos="ADJA"])* )? [pos = "NN|NE"]+
      ( [pos="ART"]? ([pos="ADJA"] ("," [pos="ADJA"])*)? [pos="NN|NE"]+ )* )*
    [pos = "ADJA"] )* )?
[pos = "NN|NE"]+ ;
```

Example (4) shows a CQP query matching the prepositional phrase in (3). Even though this query omits tests for morpho-syntactic agreement and many generalisations that would be necessary to match

similar phrases, it is hardly legible and virtually impossible to maintain. It is obvious, though, that a certain sequence of patterns – matching a simple noun chunk, which is underlined in (4) – is repeated over and again. When this pattern sequence is captured in a reusable, named rule, the query becomes much more readable. We used the CQP macro language to define a wide range of rules like those for simple noun and prepositional chunks shown in (5) below.

(5) SimpleNC ⇒
```
   [pos="ART"]? ([pos="ADJA"] ("," [pos="ADJA"])* )? [pos="NN|NE"]+
```
SimplePC ⇒
```
   [pos="APPR"] /SimpleNC[]
```

With these rules, the query (4) reduces to

(6)
```
[pos="APPR"] [pos="ART"]?
( ( /SimplePC[] /SimpleNC[]* )* [pos = "ADJA"] )
  ( "," ( /SimplePC[] /SimpleNC[]* )* [pos = "ADJA"] )* )?
[pos = "NN|NE"]+ ;
```

`/SimpleNC[]` (and `/SimplePC[]`) is the syntax used to invoke a macro (i.e. a rule) without arguments in the CQP query language. Extensions to the rule for simple NCs (for instance, allowing optional adverbs modifying adjectives) will also automatically apply to simple PCs and all parts of (6).

Although the macro language simplifies the formulation and maintenance of queries, the macros are still expanded to the full complex expressions at run-time, with a substantial speed impact on query evaluation. It would thus be desirable to "memorise" partial matches corresponding to the macro invocations in (6). When such simple NCs and PCs are annotated as non-recursive structural attributes (named "nc" and "pc" in the example below), the macro invocations are replaced by constructions like "`<nc> []* </nc>`", which matches a pre-annotated noun chunk of arbitrary length.

(7)
```
[pos="APPR"] [pos="ART"]?
( ( <pc> []* </pc> (<nc> []* </nc>)* )* [pos = "ADJA"] )
  ( "," ( <pc> []* </pc> (<nc> []* </nc>)* )* [pos = "ADJA"] )* )?
[pos = "NN|NE"]+ ;
```

Since the simple NCs and PCs need not be re-analysed whenever (7) or a similar query is executed, the performance of the query processor improves substantially. We understand off-line analysis thus as the write-back of partial query results into a corpus for later re-use. Any potentially useful part of a query expression is eligible for such write-back. This is markedly different from the other chunk parsers, whose grammars are expressly designed to identify linguistically motivated chunks or phrases.

4 Mildly recursive structures

In the classical definition, chunks are non-recursive, and consequently non-overlapping structures. For instance, (Abney 1996a) defines a chunk as the "non-recursive core of an intra-clausal constituent". Because of this non-recursivity, finite-state technology (possibly in the form of regular expressions) can be used to identify chunks, and the structural attributes of the CWB suffice for their representation. Unfortunately, as (Kermes and Evert 2003) argue, this chunk concept is hardly adequate for German. Example (8) shows a prepositional phrase with recursive centre-embedding of NPs and PPs, including post-head embedding of the genitive NP *der Apostel*.

(8) $[_{PP}$ mit $[_{NP}$ $[_{AP}$ kleinen], $[_{AP}$ $[_{PP}$ über $[_{NP}$ die Köpfe $[_{NP}$ der Apostel]]] gesetzten] Flammen]]

Consequently, (Kermes 2003) suggests an extended chunk definition that includes these kinds of recursion, but does not require highly ambiguous PP-attachment decisions to be made. In the following, we will sometimes refer to such extended chunks as *phrases* (NP, PP, etc.). Extended chunks are fully recursive structures, with an in principle unlimited number of embeddings. It would thus seem that the full power of a context-free grammar is necessary to detect such chunks, and a hierarchical data model (more or less equivalent to XML) is needed for their representation. However, the picture is somewhat different when we look at each phrase type on its own, i.e. the embedding of noun phrases in larger NPs, prepositional phrases in larger PPs, etc. In practice, such embeddings will rarely be more than two levels deep. Table 1 presents evidence for this claim obtained from the *Frankfurter Rundschau* corpus, containing approx. 40 million words of newspaper text from the early 1990's. Only 1,144 out of more than 12 million NPs identified by YAC are nested more than twice, and most of the NPs are nested only once. PPs show an even lower degree of recursion, with only 20 phrases nested twice.

nesting level	NPs	PPs
0	10,647,350	3,895,664
1	1,480,170	28,675
2	38,238	20
> 2	1,144	—

Table 1. Nesting level of (extended) noun chunks in the *Frankfurter Rundschau* corpus.

The "mild" degree of recursion found in real language data can be resolved explicitly, regarding both identification and annotation of the extended chunks. The non-recursive rules introduced in Section 3 support a single level of embedding. Queries like (6) could again be made into rules for "semi-simple" NPs and PPs, which are then used to formulate queries for "complex" NPs and PPs with two levels of embedding. Such queries should be able to identify all PPs in the *Frankfurter Rundschau* corpus, and would miss only about one in 10,000 NPs. A much more flexible strategy for the identification of mildly recursive structures writes back smaller chunks and phrases into the corpus as partial query results. Query (7) can then be used as a rule for PPs at *any* level of embedding: as soon as all the nested NPs and PPs are annotated in the corpus, (7) recognises PPs of arbitrary complexity. Thus, instead of having a set of similar rules for each nesting level, which are expanded into a huge regular expression by the query processor, (7) and analogous rules for NPs and APs can be applied iteratively until the desired degree of nesting is reached, or until no larger structures are found.

(9) $[_{PP}$ mit $[_{NP}$ $[_{AP}$ kleinen], $[_{AP}$ $[_{PP1}$ über $[_{NP1}$ die Köpfe $[_{NP2}$ der Apostel]]] gesetzten] Flammen]]

The hierarchical structure of the extended chunk *annotations* can also be resolved explicitly. Table 1 suggests that for most applications it will be sufficient to annotate only maximal phrases of each category, which account for the majority of the corpus data. In order to represent nested phrases, an additional phrase category is introduced for each level of embedding. Example (9) shows NPs embedded once as "NP1", NPs embedded twice as "NP2" etc. This representation of mildly recursive structures is compatible with the non-recursive structural attributes of the IMS Corpus Workbench, given that each of the maximal and embedded phrase types is encoded as a separate, independent s-attribute. Embedded phrases just *happen* to be nested inside the corresponding larger phrases.

5 The architecture of YAC

In order to implement a chunk parser based on the principle of off-line analysis, we needed a scripting mechanism to automate the process of executing grammar rules in the form of CQP queries, and to write back the (partial) query results to the corpus in the form of new or updated structural attributes. Since CQP is only available as a stand-alone tool for interactive work and does not provide any scripting features, Perl was an ideal choice as a "glue" language. In addition to its support for communication with other programs (required both for CQP and the newly developed tools for the write-back annotation of s-attributes), Perl provides a wide range of string manipulation functions, which proved useful in the post-processing of query results. As a necessary prerequisite, we had to develop Perl interfaces to CQP and the C library for direct corpus access. These Perl modules are now the main programming interfaces of the CWB.

Although it was suggested in the previous section that a single set of rules, applied iteratively, would be sufficient to build complex structures, it turned out to be advantageous to divide the chunking process of YAC into three levels as illustrated in Figure 3. The *first level* introduces lexical information and annotates base chunks. The *second level* is the main parsing level. Here, generic phrase structure rules similar to the one shown in (7) are applied iteratively, with results written back to the corpus after each iteration. At any given time, only the maximal phrases found up to this point are annotated in the corpus: when a larger phrase is detected, all embedded phrases of the same type are discarded. The *third level* acts as a finishing level (cf. Section 5.3)

Each level is implemented by a Perl script, based on a library of call-back functions. When the macro representing a grammar rule has been executed, the query matches are automatically collected and can be post-processed with arbitrary Perl code (most importantly, post-processing includes the partial dis-ambiguation of morpho-syntactic annotations). Then, the results are added to the appropriate structural attribute for the phrase type at hand. In this process, existing phrases are automatically discarded when they are embedded within a newly identified region, so that the required non-recursivity of s-attributes is always guaranteed.

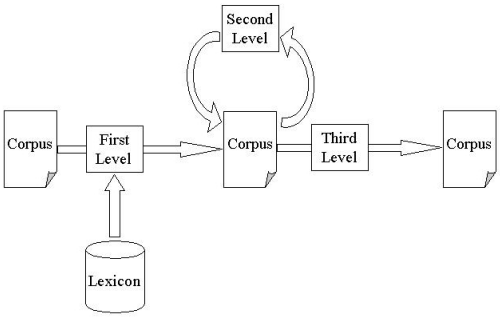

Figure 3. The three-level annotation process of YAC.

When the parsing process is completed, only maximal phrases of each type are annotated in the corpus, and no information about the nesting of PPs and NPs is encoded in the structural attributes. However, as the example in (10) shows, a maximal PP can be embedded in a maximal NP, and vice versa.

(10) [$_{NP}$ Die [$_{PP}$ auf ihren Artikel] stolze Linguistin] spricht [$_{PP}$ mit [$_{NP}$ einem Kollegen]] .
the of her paper proud linguist is talking with a colleague
'the scientist who is proud of her paper is talking to a colleague'

In order to recover the full hierarchical structure of extended chunks, all intermediate phrases constructed throughout the chunking process are collected in a text file. During an additional post-processing step, phrases of all categories are combined into a single hierarchy, which is then serialised to an XML file (cf. Section 6.1). Some care has to be taken in order to distinguish between truly nested phrases and incomplete phrases, which are annotated at some stages in the parsing process and should not be included in the final hierarchical analysis. Section 5.4 explains how head markers are used to make this distinction.

In principle, YAC can be applied to CWB-encoded corpora of any size. However, for corpora containing many millions of tokens, the constant re-encoding and re-loading of structural attributes (which may contain many millions of regions) adds up to a massive overhead. Therefore, large corpora are split up into slices of up to one million tokens, each of which is processed separately. The results from all chunks (including the reconstructed hierarchical analyses) are collected and can then be annotated in the original corpus. This slicing strategy also helps to keep the memory footprint of YAC moderately sized, allowing us to run multiple parsing processes in parallel. YAC achieves a typical parsing speed of above one million tokens per hour, depending on platform and CPU performance.

In the following, the three parsing levels and the hierarchy-building step are described in detail.

5.1 The first level

In contrast to the second level, the first level runs only once. It can be seen as a preliminary stage for the main parsing process. As such, it serves several purposes: (i) it annotates base-chunks, (ii) it annotates chunks with a specific internal structure, and (iii) it introduces lexical-semantic properties.

Base-chunks are kernels of phrasal structures or small non-recursive chunks. The chunks are used as a basis for the further annotation process. Chunks with specific internal structures are base-chunks that do not follow the general rule pattern of their category. Consequently, they need special rules, which are valid for a certain subclass of words only. A specific internal structure is usually triggered by the lexical or semantic properties of the head. In other words, the properties of the head have an impact on its behaviour, i.e., the head can select specific modifiers or can build multi-word units. Temporal nouns, e.g., can take specific noun modifiers in pre-head position (*Ende September*; end of September) and specific adverbial and year dates in post-head position (*Jahre später*; years later; *Juli 1999*; July 1999). Adjacent proper nouns (NE) can be subsumed to named entities (*Johann Sebastian Bach*). The dependencies and relations within these chunks are local, i.e., they do not involve long distance relations or complex (recursive) embedding. Thus, it is sufficient to apply these rules only once.

In addition to base-chunks and chunks with a specific internal structure, lexical and semantic information is introduced in the first level, and the base-chunks annotations are enriched with properties of the phrasal heads. Lexical information can be introduced in two different ways: (i) within the grammar rules themselves, (ii) during post-processing by the Perl script. When a lexical property triggers a specific internal structure, it is included directly in the respective rule in the form of a word list. Otherwise, the word lists are compared to the head lemma of the phrase after executing the general

rules for the given category. If the lemma is found in one of the word lists, the lexical information is added in the form of a set of features associated with the chunk.

5.2 The second level

The second level is the main parsing level. It includes relatively general and simple rules for each phrasal category, which are applied repeatedly to model complex (recursive) embedding. Example (11) below gives schematic descriptions of some (simplified) second level rules.

(11) a. AP → AdvP? PP? AC
b. NP → Determiner? Cardinal? AP? NC
c. PP → Preposition NP

The AP pattern consists of an optional AdvP, an optional PP, and an AC as kernel of the phrase. The NP rule includes an optional determiner, an optional cardinal, an optional AP, and an NC as kernel. The PP rule contains a preposition and an NP.

The rules are applied iteratively to build up larger and larger structures, which can involve complex (recursive) embedding. Thereby, the complexity of phrases is achieved by the embedding of complex structures (as in query (7)) rather than by complex rules (as in queries (4) and (6)). The same rules that build simple chunks and phrases can be used for complex phrases – the difference lies simply in the complexity of the embedded structures. The NPs below differ greatly in their complexity, but both examples can be built with the same rules (11a) and (11b).

(12) a. [$_{NP}$ eine [$_{AP}$ verständliche] Sprache]
an understandable language
b. [$_{NP}$ eine [$_{AP}$ [$_{PP}$ für den Anwender] verständliche] Sprache]
a for the user understandable language
'a language understandable for the user'

The rules of each phrasal category are combined into a rule block, after which CQP is restarted to re-load the newly annotated structures. Thus, the results of each rule block are immediately available to the following rule blocks: the rules can build not only on the output of the first level and previous iterations of the second level, but also on the output of previous rule blocks. This strategy reduces the number of iterations needed in the second level by a factor of up to three, since the bracketed phrases in (12b) can be annotated in a single pass (rather than the three passes which would otherwise be necessary). Note that only the largest annotated structure of each phrasal category is available, though.

In contrast to many other approaches, YAC was designed to exploit various special cases to improve its analysis. In specific "secure" contexts, constraints that are usually imposed on the generic rules can be relaxed. For instance, the list of acceptable post-head modifiers can be extended to include PPs when their attachment is disambiguated by parentheses or quotation marks in the text, cf. (13).

(13) a. [$_{NP}$ " Einladung [$_{PP}$ zur Enthauptung] "]
invitation to the decapitation
b. [$_{NP}$ die schlagfertige " Frau [$_{PP}$ mit den Hüten] "]
the quick-witted woman with the hats

The number of iterations in the second level is in principle unlimited. The level can be repeated until no more larger structures are found. Experience has shown, though, that three iterations are sufficient to cover the complexity of all but the most contrived examples.

The fact that the results of the rules are post-processed by Perl scripts allows the overgeneralisation and underspecification of parts of the grammar. Constraints and filters can then be applied to the results, either written directly in Perl or using the interactive subsetting and search functions of CQP. Only structures that satisfy these additional constraints and pass the filters are annotated in the corpus. Complex APs embedding PP or NP structures, e.g., are only built in a "secure" context, where they are included in a larger NP with at least one other element preceding the AP (cf. (14) below). This particular constraint is tested by Perl code that scans the context of the hypothesised AP.

(14) [$_{NP}$ die [$_{AP}$ [$_{PP}$ von der Gemahlin] [$_{PP}$ gegen das Fußpilzrisiko] empfohlenen]] Socken
the of the wife against athlete's foot recommended socks
'the socks recommended by the wife against athlete's foot'

5.3 The third level

The third level can be seen as a finalising level and serves multiple purposes: (i) different phrasal categories are united under one label, (ii) coordination of maximal chunks is performed, (iii) decisions are made which need full knowledge of the chunks, and (iv) the category of certain chunks is changed.

During the parsing process some phrasal categories are split up into several sub-categories. NPs, e.g., are split into NPs with determiner (NP), NPs without determiner (NCC), and base noun chunks (NC). After the parsing process is completed, all three sub-categories are combined into the category NP.

Adverbial and predicatively used adjectives share the same PoS-tag in the tag set used by YAC. They can only be differentiated by their syntactic context when the chunking process has been completed. Adverbially used APs are embedded as modifiers in other APs or NPs, whereas predicatively used APs stand alone (as immediate children of a sentence or clause). The category of adverbially used APs is changed accordingly to AdvP, and predicatively used APs are marked with the additional feature *pred* in contrast to attributive APs, which are marked with the feature *attr*.

5.4 Constructing a hierarchical analysis

Since the CWB does not support recursive or overlapping structures of the same category, the full hierarchical analysis of extended chunks has to be built after the actual parsing process. In order to do so, the intermediate and maximal structures from all stages of the parsing process are collected in a temporary text file. Afterwards, a Perl script re-orders this list with respect to the position of each structure in the corpus, so that nested phrases immediately follow the containing phrase. It is then easy to combine phrases of all categories into a single hierarchy. Since *all* intermediate structures are collected, there can be more than one version of the same phrase, representing different phases of its construction as the examples (15a-c) show. Only the largest version is to be included in the hierarchical structure. In order to determine whether a smaller phrasal structure is an embedded phrase or only a shorter version of the same phrase, the position of its head is used as a reference point. In other words, only the largest one of several overlapping structures with the same head is included in the hierarchy, while the smaller versions are discarded. Overlapping structures with different head positions, on the other hand, are both included in the hierarchy as nested phrases. The examples below show overlapping NPs collected during the parsing process. The token representing the head position is underlined in each case.

(15) a. [$_{NP}$ Faszination]
fascination
b. [$_{NP}$ gewisse <u>Faszination</u> des Schattens]
certain fascination of the shadow
c. [$_{NP}$ eine gewisse <u>Faszination</u> des Schattens]
a certain fascination of the shadow
d. [$_{NP}$ des <u>Schattens</u>]
of the shadow

The first three NPs share the same head (*Faszination*), whereas the last one has a different head (*Schatten*). Thus, the first two NPs (15a-b) are recognised as incomplete versions of (15c), while (15d) proves to be an embedded NP. The resulting hierarchical structure includes only (15c) and (15d):

(16) [$_{NP}$ eine gewisse <u>Faszination</u> [$_{NP}$ des <u>Schattens</u>]]
a certain fascination of the shadow

6 Storage, retrieval, and display

6.1 The XML output format

The results of the hierarchy construction step are converted into an XML file whose element tree is isomorphic to the hierarchical syntactic analysis produced by YAC. Each phrase type is represented by appropriately named XML elements, e.g. `<np>` for noun phrases and `<vc>` for verbal complexes. The various annotations of phrases (typically including features, head lemma, and partially disambiguated morpho-syntactic information) are encoded as attributes of the respective XML elements. Tokens are represented by `<t>` elements and may include the values of one or more positional attributes from the original corpus (in embedded `<a>` elements). Figure 4 shows a slightly simplified example of YAC's XML output for the input sentence *Lisa trocknet ihre wegen des Regens nassen Füße ab* ('Lisa is

rubbing her feet dry, which are wet because of the rain'). The <t> elements include word form, part-of-speech, and lemma annotations, the word forms being underlined as a guide for the reader.

The precise document structure of the YAC output format depends on the phrase types and annotations defined in the grammar, and does therefore not adhere to a fixed DTD. Although authors such as (Mengel and Lezius 2000) would certainly see this flexibility as a disadvantage, we have found the format reasonably human-readable, as well as convenient and efficient for further processing, especially with XSLT stylesheets. Such stylesheets can be used to transform the output of YAC into HTML or other suitable formats for viewing, and to extract lexical and syntactic information.

```
<s id="s1234">
  <np f="|ne|nogen|" h="Lisa" agr="|Akk:F:Sg|...">
    <t><a>Lisa</a> <a>NE</a> <a>Lisa</a></t>
  </np>
  <vc f="|norm|" h="abtrocknen">
    <t><a>trocknet</a> <a>VVFIN</a> <a>trocknen</a></t>
  </vc>
  <np f="|norm|" h="Fuß" agr="|Akk:M:Pl|...">
    <t><a>ihre</a> <a>PPOSAT</a> <a>ihr</a></t>
    <ap f="|attr|pp|" h="naß" agr="|Akk:F:Pl|...">
      <pp f="|norm|" h="wegen:Regen" agr="|Gen:M:Sg|Gen:N:Sg|">
        <t><a>wegen</a> <a>APPR</a> <a>wegen</a></t>
        <np f="|norm|" h="Regen" agr="|Gen:M:Sg|Gen:N:Sg|">
          <t><a>des</a> <a>ART</a> <a>d</a></t>
          <t><a>Regens</a> <a>NN</a> <a>Regen</a></t>
        </np>
      </pp>
      <t><a>nassen</a> <a>ADJA</a> <a>naß</a></t>
    </ap>
    <t><a>Füße</a> <a>NN</a> <a>Fuß</a></t>
  </np>
  <t><a>ab</a> <a>PTKVZ</a> <a>ab</a></t>
  <t><a>.</a> <a>$.</a> <a>.</a></t>
</s>
```

Figure 4. XML output of YAC (simplified example).

6.2 TIGERSearch

A much more powerful tool for displaying and searching hierarchical annotations is the TIGERSearch software (König and Lezius 2000; Lezius 2002).[2] Wolfgang Lezius kindly provided an import filter for the YAC format, and we have been using the TIGERSearch environment to display the hierarchical annotations of complex phrases. The TIGERSearch query language was designed for the full syntactic analysis of a complex grammar formalism or a manually annotated treebank. Its expressiveness is rarely needed for the relatively simple and shallow structures identified by YAC. So far, we have relied mostly on stylesheets and CQP queries (as described in the following section) for data extraction.

6.3 IMS Corpus Workbench

In order to make the full hierarchical analyses of YAC available to the Corpus Workbench, it was necessary to extend its encoding tool with comprehensive support for XML markup. XML elements are stored in structural attributes with corresponding names (e.g., <np> elements in the "np" attribute). As suggested in Section 4, recursion of elements with the same name, e.g. <np>, is resolved by automatic renaming of the embedded elements to <np1>, <np2>, etc. When the nesting exceeds a pre-determined threshold, the most deeply nested elements are discarded. The attributes of XML elements (attribute-value pairs in start tags) are parsed and assigned to implicitly defined s-attributes, following a simple naming convention. Thus, <np> elements with their f, h, and agr attributes are stored in s-attributes "np", "np_f", "np_h", and "np_agr", where they are easily accessed from CQP queries (cf. Evert 2003).

The feature-set support of CQP is a distinct advantage over XSLT and TIGERSearch. It can be used to test whether a phrase is annotated with a certain feature such as *attr*, and gives access to the partially disambiguated morpho-syntactic annotations in the agr attributes of NPs, PPs, and APs. Most CQP queries will use maximal phrases (of each category) only, but the nesting of different categories can be modelled with balanced start and end tags in a query. A non-trivial example is (17), which matches a PP embedded in a maximal NP with the head lemma *Kaffee* and a genitive NP in post-head position:

```
(17)  <np_h "Kaffee"> []* (<pp>|<pp1>) []+ (</pp1>|</pp>)
            []+ <np_agr1 matches "Gen:.*"> []+ </np_agr1> </np_h> ;
```

[2] See http://www.ims.uni-stuttgart.de/projekte/TIGER/TIGERSearch/ for further information and availability.

(Kermes 2003) uses large collections of similar queries, again with Perl as a scripting and post-processing language, to extract evidence for linguistic and lexicographic phenomena from YAC-parsed text corpora. Although CQP is a useful tool for interactive queries and information extraction, its KWIC-style output, which displays structure boundaries as XML tags, is hardly suitable for hierarchical phrase structure annotations. For the corpus linguist and lexicographer, a concise labelled-bracketing notation might be the best choice. However, different applications have different needs. Sometimes it will be necessary to display the head lemma, or indicate the presence of certain features and morpho-syntactic properties in various ways (e.g. show attributively and predicatively used APs in different colours, or highlight all genitive NPs). Sometimes, a tabular or tree-structured display will be more appropriate. Sometimes, it is sufficient to show only the token sequences matching a query and display a variable amount of context in a separate window when requested. What is needed, thus, is not an all-in-one solution, but a display framework that can be easily and quickly adapted to specific user needs.

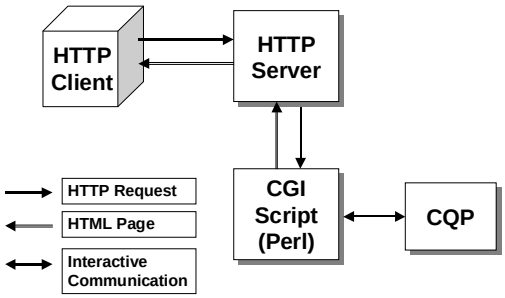

Figure 5. Typical architecture of a web interface to the CWB.

We believe that a web interface to the IMS Corpus Workbench provides such a general framework (Figure 5). CGI scripts written in Perl use the Perl-CQP interface to execute queries, and transform the query results into a suitable HTML display. The powerful string-manipulation facilities of Perl make it easy to implement many different views of the same data. An additional benefit is the client-server architecture of such a web interface, which gives access to CWB corpora even from platforms that are currently not supported (all that is needed is an HTTP server running on a supported platform). Some examples can be found at http://www.ims.uni-stuttgart.de/projekte/CorpusWorkbench/Demos/.

7 References

Abney S 1996a. *Chunk stylebook*. Working draft.

Abney S 1996b. Partial parsing via finite-state cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop*.

Christ O 1994. A modular and flexible architecture for an integrated corpus query system. In *Papers in Computational Lexicography COMPLEX '94.* Budapest, Hungary, pp 22-32.

Eckle-Kohler J 1999. *Linguistisches Wissen zur automatischen Lexikon-Akquisition aus deutschen Textcorpora.* Berlin, Logos Verlag.

Evert S 2003. *The CQP Query Language Tutorial.* Technical Report, University of Stuttgart.

Kermes H 2003. *Off-line (and on-line) text analysis for computational lexicography.* PhD thesis, IMS, University of Stuttgart, to appear.

Kermes H, Evert S 2003. Text analysis meets corpus linguistics. In *Proceedings of Corpus Linguistics 2003*, Lancaster, UK.

König E, Lezius W 2000. A description language for syntactically annotated corpora. In *Proceedings of COLING 2000*. Saarbrücken, Germany, pp 1056-1060.

Lezius W 2002. *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora.* PhD thesis, IMS, University of Stuttgart. *Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS)*, volume 8, number 4.

Mengel A, Lezius W 2000. An XML-based representation format for syntactically annotated corpora. In *Proceedings of the Second International Conference on Language Resources and Engineering (LREC)*, Volume 1, Athens, Greece, pp. 121-126.