Contents

1 Introduction 3
  1.1 The IMS Corpus Workbench (CWB) . . . . . . . . . . . . . . . . . . . . . . . 3
  1.2 The CWB corpus data model . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  1.3 Corpora used in the tutorial . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

2 Basic CQP features 8
  2.1 Getting started . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  2.2 Searching for words . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  2.3 Display options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
  2.4 Useful options . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  2.5 Accessing token-level annotations . . . . . . . . . . . . . . . . . . . . . . . . 11
  2.6 Combinations of attribute constraints: Boolean expressions . . . . . . . . . 12
  2.7 Sequences of words: token-level regular expressions . . . . . . . . . . . . . . 12
  2.8 Example: finding “nearby” words . . . . . . . . . . . . . . . . . . . . . . . 12
  2.9 Sorting and counting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

3 Working with query results 15
  3.1 Named query results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  3.2 Saving data to disk . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
  3.3 Anchor points . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
  3.4 Frequency distributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
  3.5 Set operations with named query results . . . . . . . . . . . . . . . . . . . . 18
  3.6 The set target command . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

4 Labels and structural attributes 21
  4.1 Using labels . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
  4.2 Structural attributes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
  4.3 Structural attributes and XML . . . . . . . . . . . . . . . . . . . . . . . . . . 23
  4.4 XML document structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
A.4 Reserved words in the CQP language

a: asc ascending
b: by
c: cat cd collocate contains cut
d: def define delete desc descending diff difference discard dump
e: exclusive exit expand
f: farthest foreach
g: group
h: host
i: inclusive info inter intersect intersection
j: join
k: keyword
l: left leftmost
m: macro maximal match matchend matches meet MU
n: nearest no not NULL
o: off on
r: randomize reduce RE reverse right rightmost
s: save set show size sleep sort source subset
t: TAB tabulate target target[0-9] to
u: undump union unlock user
w: where with within without
y: yes
1 Introduction

1.1 The IMS Corpus Workbench (CWB)

History and framework

- 1993 – 1996: Project on Text Corpora and Exploration Tools (financed by the Land Baden-Württemberg)
- 1996 – 2004: Continued in-house development (partly financed by various research and industrial projects)
- CWB version 3.0 to be released in early 2005 (pre-release versions have been shipped since October 2001)

Related projects and applications at the IMS

- 1994 – 1996: DECIDE project (EU programme MLAP-93) (extraction of collocation candidates, macro processor mp)
- Since 1996: Various commercial and research applications (terminology extraction, dictionary updates)
- 1999 – 2000: DOT project (Databank Overheidsterminologie) (stand-alone system for extraction of Dutch legal terminology)
- 2001 – 2003: Transferbereich 32 (financed by the DFG) (applications in computational lexicography)

Some external applications of the IMS Corpus Workbench

- AC/DC project at the Linguata centre (SINTEF, Oslo, Norway) (on-line access to a 180 M word corpus of Portuguese newspaper text) http://acdc.linguateca.pt/cetempublico/
- CorpusEye (user-friendly CQP) in the VISL project (SDU, Denmark) (on-line access to annotated corpora in various languages) http://corp.hum.sdu.dk/corpustop.html
- SSSLIT Dev Online services (SSSLIT, University of Bologna, Italy) (on-line access to annotated corpora in various languages) http://sslmitdev-online.sslmit.unibo.it/
- Corpografo environment at the Linguata centre (FLUP, Porto, Portugal) (an easy-to-use Web-based environment for corpus research) http://www.linguateca.pt/corpografo/

\[^1\] Designing and evaluating Extraction Tools for Collocations in Dictionaries and Corpora
Technical aspects

- CWB uses proprietary token-based format for corpus storage:
  - binary encoding ⇒ fast access
  - full index ⇒ fast look-up of word forms and annotations
  - specialised data compression algorithms
  - corpus size: up to 500 million words, depending on annotations
  - text data and annotations cannot be modified after encoding
    (but it is possible to add new annotations or overwrite existing ones)
  - assumes Latin-1 encoding, but compatible with other 8-bit ASCII extensions
    (Unicode text in UTF-8 encoding can be processed with some caveats)

- Typical compression ratios for a 100 million word corpus:
  - uncompressed text: ≈ 1 GByte (without index & annotations)
  - uncompressed CWB attributes: ≈ 790 MBytes (ratio: 1.3)
  - word forms & lexical attributes: ≈ 360 MBytes (ratio: 2.8)
  - categorical attributes (e.g. POS tags): ≈ 120 MBytes (ratio: 8.5)
  - binary attributes (yes/no): ≈ 50 MBytes (ratio: 20.5)

- Supported operating systems:
  - SUN Solaris 2.8 (Sparc processors)
  - Linux 2.4+ (Intel i386 and compatible processors)
  - Corpus data format is platform-independent
  - Source code should compile on most POSIX-compliant 32-bit platforms

Components of the CWB

- tools for encoding, indexing, compression, decoding, and frequency distributions
- global “registry” holds information about corpora (name, attributes, data path)
- corpus query processor (CQP):
  - fast corpus search (regular expression syntax)
  - use in interactive or batch mode
  - results displayed in terminal window
- CWB/Perl interface for post-processing, scripting and web interfaces

A.3 Annotations of the tutorial corpora

English corpus: DICKENS

- Positional attributes (token annotations)
  - word word forms ("plain text")
  - pos part-of-speech tags (Penn Treebank tagset)
  - lemma base forms (lemmata)

- Structural attributes (XML tags)
  - novel individual novels
  - novel_title title of the novel
  - book when text is subdivided into books
  - book_num number of the book
  - chapter chapters
  - chapter_num number of the chapter
  - chapter_title optional title of the chapter
  - title encloses title strings of novels, books, and chapters
  - p paragraphs
  - p_len length of the paragraph (in words)
  - s sentences
  - s_len length of the sentence (in words)
  - np noun phrases
  - np_h head lemma of the noun phrase
  - np_len length of the noun phrase (in words)
  - pp prepositional phrases
  - pp_h functional head of the PP (preposition)
  - pp_len length of the PP (in words)

German corpus: GERMAN-LAW

- Positional attributes (token annotations)
  - word word forms ("plain text")
  - pos part-of-speech tag (STTS tagset)
  - lemma base forms (lemmatised forms)
  - alemma ambiguous lemmatisation (see examples in Section 5.6)
  - agr noun agreement features (see section 5.6)

Each agreement feature has the form <ccc>:<gg>:<nn>:<ddd> with

- ccc = case (Nom, Gen, Dat, Acc)
- g = gender (M, F, N)
- nn = number (Sg, Pl)
- ddd = determination (Def, Ind, Nil)
A.2 Part-of-speech tags and useful regular expressions

The English PENN tagset (DICKENS)

- **NN**: Common noun, singular or mass noun
- **NNS**: Common noun, plural
- **NP, NPS**: Proper noun, singular/plural
- **N**: Matches any common or proper noun
- **PP**: Matches any pronoun (personal or possessive)
- **JJ**: Adjective
- **JJR, JJR**: Adjective, comparative/superlative
- **VB**: Matches any verbal form
- **VBG, VBG**: Present/past participle
- **RBR, RRS**: Adverb, comparative/superlative
- **MD**: Modal
- **DT**: Determiner
- **PDT**: Predeterminer
- **IN**: Preposition, subordinating conjunction
- **CC**: Coordinating conjunction
- **TO**: Any use of “to”
- **RP**: Particle
- **WP**: Wh-pronoun
- **WDT**: Wh-determiner
- **SENT**: Sentence-final punctuation

The German STTS tagset (GERMAN-LAW)

- **NN**: Common noun (singular or plural)
- **NE**: Proper noun (singular or plural)
- **N**: Matches any nominal form
- **PP**: Matches any pronoun (personal or possessive)
- **ADJA**: Attributive adjective
- **ADJD**: Predicative adjective (also when used adverbially)
- **ADP**: Matches any adjectival form
- **VVB**: Matches any full verb
- **VAA**: Matches any auxiliary verb
- **MN**: Matches any modal verb
- **V**: Matches any verbal form
- **ADV**: Adverb
- **ART**: Determiner
- **APP**: Preposition
- **APPR**: Fused preposition and determiner
- **KOD**: Matches any conjunction
- **TRUNC**: Truncated word (e.g. “unter-“)
- `\$`: Sentence-final punctuation
- `\$`: Sentence-internal punctuation

1.2 The CWB corpus data model

The following steps illustrate the transformation of textual data with some XML markup into the CWB data format.

1. **Formatted text** (as displayed on-screen or printed)
   - An easy example. Another very easy example. Only the easiest examples!

2. **Text with XML markup** (at the level of texts, words or characters)
   - `<text id=42 lang="English"> <s>An easy example.</s><s> Another <i>very</i> easy example.</s> <s><b>O</b>nly the <b>ea</b>siest ex<b>a</b>mples!</s>`

3. **Tokenised text** (character-level markup has to be removed)
   - `<text id=42 lang="English"> <s> An easy example . </s> <s> Another very easy example . </s> <s> Only the easiest examples ! </s>`

4. **Text with linguistic annotations** (annotations are added at token level)
   - `<text id=42 lang="English"> <s>An DET/a easy ADJ/easy example SN/exampl . </s> <s>Another DET/another very/ADV/very easy ADJ/easy example SN/exampl ./PUN/. </s> <s>Only/ADV/only the DET/the easiest ADJ/easy example SN/exampl ./PUN/! </s>`

5. **Text encoded as CWB corpus** (tabular format, similar to relational database)
   - A schematic representation of the encoded corpus is shown in Figure 1. Each token (together with its annotations) corresponds to a row in the tabular format. The row numbers, starting from 0, uniquely identify each token and are referred to as *corpus positions*.
   - Each (token-level) annotation layer corresponds to a column in the table, called a *positional attribute* or *p-attribute* (note that the original word forms are also treated as an attribute with the special name *word*). Annotations are always interpreted as character strings, which are collected in a separate lexicon for each positional attribute. The CWB data format uses lexicon IDs for compact storage and fast access.

Matching pairs of XML start and end tags are encoded as token regions, identified by the corpus positions of the first token (immediately following the start tag) and the last token (immediately preceding the end tag) of the region. (Note how the corpus position of an XML tag in Figure 1 is identical to that of the following or preceding token, respectively.) Elements of the same name (e.g. `<s>...</s>` or `<text>...</text>`) are collected and referred to as *structural attribute* or *s-attribute*. The corresponding regions must be non-overlapping and non-recursive. Different s-attributes are completely independent in the CWB: a hierarchical nesting of the XML elements is neither required nor can it be guaranteed.

Key-value pairs in XML start tags can be stored as an annotation of the corresponding s-attribute region. All key-value pairs are treated as a single character string, which has to be “parsed” by a CQP query that needs access to individual values. In the recommended encoding procedure, an additional s-attribute (named `element_key`) is automatically created for each key and is directly annotated with the corresponding value (cf. `<text_id>` and `<text_lang>` in Figure 1).
A.1 Summary of regular expression syntax

At the character level, CQP supports POSIX 1003.2 regular expressions as provided by the system libraries. A full description of the regular expressions is found in the manual page. Various books such as 'Mastering Regular Expressions' (1992) and 'Advanced Computer Programming Languages' (1993) provide a lot of additional information.

In many programming languages, it is customary to embed regular expressions in strings (single or double-quoted) using a slightly different syntax where regular expressions are written as
\[ \text{<regexp>} \rightarrow \text{word, word} \]

Basic syntax of regular expressions

- letter, digit, and word are matched literally (including all non-ASCII characters)
- \( ^ {n} \) matches any single character except the character immediately following the \( ^ {n} \) (\( ^ {n} =a, b, c, \ldots \))
- \( [^n] \) is an equivalent to the less compact expression \( a \ldots z \) (\( a \ldots z \) are the characters listed in the character set \( \{ a, b, c, \ldots \} \), \( a \ldots z \) are the characters listed in the character set \( \{ a, b, c, \ldots \} \), \( a \ldots z \) are the characters listed in the character set \( \{ a, b, c, \ldots \} \), \( a \ldots z \) are the characters listed in the character set \( \{ a, b, c, \ldots \} \), \( a \ldots z \) are the characters listed in the character set \( \{ a, b, c, \ldots \} \), \( a \ldots z \) are the characters listed in the character set \( \{ a, b, c, \ldots \} \), \( a \ldots z \) are the characters listed in the character set \( \{ a, b, c, \ldots \} \).
6.6 Easter eggs

- the pre-release versions of CQP v3.0 include a hidden regular expression optimiser; this optimiser detects simple expressions used for prefix, suffix or infix searches such as
  
  > "under.*";
  > ".*ment";
  > ".*time.*";

  and replaces the regexp engine with a highly efficient Boyer-Moore search algorithm

- the regular expression optimiser is activated with the command
  
  > set Optimize on;

  you can watch the optimiser at work by setting

  > set CLDebug on;

- the optimiser will be activated by default in the official v3.0 release

1.3 Corpora used in the tutorial

Pre-encoded versions of these corpora are distributed free of charge together with the IMS Corpus Workbench. Perl scripts for encoding the British National Corpus (World Edition) can be provided at request.

**English corpus: DICKENS**

- a collection of novels by Charles Dickens
- ca. 3.4 million tokens
- derived from Etext editions (Project Gutenberg)
- document-structure markup added semi-automatically
- part-of-speech tagging and lemmatisation with TreeTagger
- recursive noun and prepositional phrases from Gramotron parser

**German corpus: GERMAN-LAW**

- a collection of freely available German law texts
- ca. 816,000 tokens
- part-of-speech tagging with TreeTagger
- morphosyntactic information and lemmatisation from IMSLex morphology
- partial syntactic analysis with YAC chunker

See Appendix A.3 for a detailed description of the token-level annotations and structural markup of the tutorial corpora (positional and structural attributes).
2 Basic CQP features

2.1 Getting started

- start CQP by typing
  `cqp` in a shell window (the `$` indicates a shell prompt)
- `-e` flag activates command-line editing features
- optional `-C` flag activates color highlighting (experimental)
- every CQP command must be terminated with a semicolon (`;`)
- list available corpora
  `show corpora;`
- get information about corpus (including corpus size in tokens)
  `info DICKENS;`
- displays information file associated with the corpus, whose contents may vary; ideally, this should give a description of the corpus composition, a summary of the positional and structural annotations, and a brief overview of annotation codes such as the part-of-speech tag set used
  - activate corpus for subsequent queries (use `TAB` key for name completion)
  `[no corpus]> DICKENS;`
  DICKENS>
  in the following examples, the CQP command prompt is indicated by a `>` character
  - list attributes of activated corpus ("context descriptor")
    `show cd;`

2.2 Searching for words

- search single word form (single or double quotes are required: `'...` or `"..."`)
  `"interesting";`
  → shows all occurrences of interesting
- the specified word is interpreted as a regular expression
  `"interest(s|(ed|ing)(ly)?)?";`
  → interest, interests, interested, interesting, interestedly, interestingly
- see Appendix A.1 for an introduction to the regular expression syntax
- note that special characters have to be "escaped" with backslash (`\`)
  `"a\" fails;  "\"\" → \;  \".\" → . . a b c ⋯;  "\$\" → $.
  - `critical` characters are: . ? * + | ( ) { } ^ $`

Stefan Evert 8 © 2005 IMS Stuttgart
6.5 Generating frequency tables

- for many applications it is important to compute frequency tables for the matching strings, tokens in the immediate context, attribute values at different anchor points, different attributes for the same anchor, or various combinations thereof
- frequency tables for the matching strings, optionally normalised to lowercase and extended or reduced by an offset, can easily be computed with the count command (cf. Sections 2.9 and 3.3); when pretty-printing is deactivated (cf. Section 6.3), its output has the form
  
  frequency TAB first line TAB string (type)

- advantages of the count command:
  - strings of arbitrary length can be counted
  - frequency counts can be based on normalised strings (%cd flags)
  - the instances (tokens) for a given string type can easily be identified, since the underlying query result is automatically sorted by the count command, so that these instances appear as a block starting at match number first line

- an alternative solution is the group command (cf. Section 3.4), which computes frequency distributions over single tokens (i.e. attribute values at a given anchor position) or pairs of tokens (recall the counter-intuitive command syntax for this case); when pretty-printing is deactivated, its output has the form

  [ attribute value TAB ] attribute value TAB frequency

- advantages of the group command:
  - can compute joint frequencies for non-adjacent tokens
  - faster when there are relatively few different types to be counted
  - supports frequency distributions for the values of s-attributes

- the advantages of these two commands are for the most part complementary (e.g., it is not possible to normalise the values of s-attributes, or to compute joint frequencies of two non-adjacent multi-token strings); in addition, they have some common weaknesses, such as relatively slow execution, no options for filtering and pooling data, and limitations on the types of frequency distributions that can be computed (only simple joint frequencies, no nested groupings)

- therefore, it is often necessary (and usually more efficient) to generate frequency tables with external programs such as dedicated software for statistical computing or a relational database; these tools need a data table as input, which lists the relevant feature values (at specified anchor positions) and/or multi-token strings for each match in the query result; such tables can often be created from the output of cat (using suitable PrintOptions, Context and show settings)

- this procedure involves a considerable amount of re-formatting (e.g. with Unix command-line tools or Perl scripts) and can easily break when there are unusual attribute values in the data; both cat output and the re-formatting operations are expensive, making this solution inefficient when there is a large number of matches.

- \begin{itemize}
  \item \LaTeX-angle escaping sequences \texttt{\string}, \texttt{\char}, \texttt{\char} and \texttt{\char}, followed by an appropriate ASCII letter, are used to represent characters with diacritics when they cannot be entered directly
  \item NB: this feature works only for the Latin-1 encoding and cannot be deactivated
  \item additional special escape sequences:
    \texttt{\char{a} \char{b} \char{c} \char{d} \char{e} \char{f} \char{g} \char{h} \char{i} \char{j} \char{k} \char{l} \char{m} \char{n} \char{o} \char{p} \char{q} \char{r} \char{s} \char{t} \char{u} \char{v} \char{w} \char{x} \char{y} \char{z} \char{A} \char{B} \char{C} \char{D} \char{E} \char{F} \char{G} \char{H} \char{I} \char{J} \char{K} \char{L} \char{M} \char{N} \char{O} \char{P} \char{Q} \char{R} \char{S} \char{T} \char{U} \char{V} \char{W} \char{X} \char{Y} \char{Z} \char{0} \char{1} \char{2} \char{3} \char{4} \char{5} \char{6} \char{7} \char{8} \char{9}}
    \item use flags \texttt{\char{c}} and \texttt{\char{d}} to ignore case / diacritics
\end{itemize}

2.3 Display options

- KWIC display ("key word in context")

- if query results do not fit on screen, they will be displayed one page at a time
- press SPC (space bar) to see next page, RET (return) for next line, and q to return to CQP
- some pagers support b or the backspace key to go to the previous page, as well as the use of the cursor keys, PgUp, and PgDn
- at the command prompt, use cursor keys to edit input (\texttt{\char{a} \char{b} \char{c} \char{d} \char{e} \char{f} \char{g} \char{h} \char{i} \char{j} \char{k} \char{l} \char{m} \char{n} \char{o} \char{p} \char{q} \char{r} \char{s} \char{t} \char{u} \char{v} \char{w} \char{x} \char{y} \char{z} \char{A} \char{B} \char{C} \char{D} \char{E} \char{F} \char{G} \char{H} \char{I} \char{J} \char{K} \char{L} \char{M} \char{N} \char{O} \char{P} \char{Q} \char{R} \char{S} \char{T} \char{U} \char{V} \char{W} \char{X} \char{Y} \char{Z} \char{0} \char{1} \char{2} \char{3} \char{4} \char{5} \char{6} \char{7} \char{8} \char{9}})
- change context size
  \begin{itemize}
    \item \texttt{\char{set Context 20;}} \hspace{1cm} (20 characters)
    \item \texttt{\char{set Context 5 words;}} \hspace{1cm} (5 tokens)
    \item \texttt{\char{set Context s;}} \hspace{1cm} (entire sentence)
    \item \texttt{\char{set Context 3 s;}} \hspace{1cm} (same, plus 2 sentences each on left and right)
  \end{itemize}
- \texttt{\char{type \char{cat};}} to redisplay matches
- display current context settings
  \begin{itemize}
    \item \texttt{\char{set Context;}}
    \item \texttt{\char{left and right context can be set independently}}
  \end{itemize}
in an interactive CQP session, the input file can be omitted and the undump table can then be entered directly on the command line; this feature works only when command-line editing support is enabled with the \(-e\) switch, so CWB/Perl and similar interfaces have to create a temporary file for the undump table; try out the example above with

\> undump B;

further columns for the \texttt{target} and \texttt{keyword} anchors (in this order) can be selected by adding \texttt{with target} or \texttt{with target keyword} to the command:

\> undump B with target keyword < "mydump.tbl";

when the rows of the undump table are not sorted in the natural order (i.e. by corpus position), they have to be re-ordered internally so that CQP can work with them; however, the original sort order is automatically remembered and will be used by the \texttt{cat} and \texttt{dump} commands (until it is modified by a new \texttt{sort} command); if you sort a query result \texttt{A}, save it with \texttt{dump} to a text file, and then read this file back in as named query \texttt{B} (after adjusting the file format), \texttt{A} and \texttt{B} will be sorted identically.

\begin{itemize}
  \item in many cases, overlapping or unsorted matches are not intentional but rather errors in an automatically generated undump table; in order to catch such errors, the additional keyword \texttt{ascending (or asc)} can be specified before the \(<\) character:
  \begin{verbatim}
  \> undump B with target ascending < "mydump.tbl";
  \end{verbatim}
  will abort with an error message (indicating the row number where the error occurred) unlessthematchranges in \texttt{mydump.tbl} are non-overlapping and sorted in corpus order.
\end{itemize}

A typical example for the use of \texttt{dump} and \texttt{undump} is linking CQP queries to corpus metadata stored in an external SQL database; assume that a corpus consists of a large collection of transcribed dialogues, which are represented by \texttt{<dialogue>} regions; a rich amount of metadata (about the speakers, setting, topic, etc.) is available in a SQL database; the database entries can be linked directly to the \texttt{<dialogue>} regions by recording their start and end corpus positions in the database.\footnote{Of course, it is also possible to establish an indirect link through document IDs, which are annotated as \texttt{<dialogue id=XXXX> \ldots</dialogue>}. Especially when there are a very large number of dialogues in the corpus, the direct link can be much more efficient.} the following commands generate a dump table with the required information, which can easily be loaded into the database (ignoring the third and fourth column)

\begin{verbatim}
  \> A = <dialogue> [] expand to dialogue;
  \> dump A > "dialogues.tbl";
\end{verbatim}
corpus queries will often be restricted to a subcorpus by specifying constraints on the metadata; when the metadata constraints have been resolved in the SQL database, they can be translated to the corresponding regions in the corpus (again represented by start and end corpus position); after sorting these regions in ascending order and saving them to a suitable undump file, they are loaded into CQP with an \texttt{undump} command; the resulting query result can then be activated as a subcorpus for the ensuing query

\begin{verbatim}
  \> undump SubCorpus < "subcorpus.tbl";
  \> SubCorpus;
  \end{verbatim}

\section*{2.4 Useful options}

\begin{itemize}
  \item enter \texttt{set}; to display list of options (abbreviations shown in brackets)
  \item \texttt{set <option>} ; shows current value
  \item \texttt{set ProgressBar (on|off)};
    to show progress of query execution
  \item \texttt{set Timing (on|off)};
    to show execution times of queries and some other commands
  \item \texttt{set PrintNode (ascii|sgml|html|latex)};
    to set output format for KWIC display and frequency distributions
  \item \texttt{set PrintOptions (hdr|num|...)};
    to turn various formatting options on (\texttt{hdr}, \texttt{num}, \ldots) or off (\texttt{nohdr}, \texttt{nonum}, \ldots)
  \item type \texttt{set PrintOptions}; to display the current option settings useful options: \texttt{hdr} (display header), \texttt{num} (show line numbers), \texttt{tbl} (format as table in HTML and \LaTeX{} modes), \texttt{bdr} (table with border lines)
  \item \texttt{set (LD|RD) <string>};
    change left/right delimiter in KWIC display from the default \(<\) and \(>\) markers
  \item \texttt{set ShowTagAttributes (on|off)};
    to display key-value pairs in XML start tags (if annotated in the corpus)
\end{itemize}
6.4 Exchanging corpus positions with external programs

- an important aspect of interfacing CQP with external tools is access to the corpus positions of query matches (as well as target and keyword anchors); this is a prerequisite for the extraction of further information about the matches by direct corpus access, and it is the most efficient way to relate query matches to external data structures (e.g. in a SQL database)

- the dump command (Section 3.3) prints the required information in a tabular ASCII format that can easily be parsed by other tools or read into a SQL database; each row of the resulting table corresponds to one match of the query, and the four columns give the corpus positions of the match, matchend, target and keyword anchors, respectively; the example below is reproduced from Section 3.3

| 1019887 | 1019888 | -1 | -1 |
| 1924977 | 1924979 | 1924978 | -1 |
| 1986623 | 1986624 | -1 | -1 |
| 2086708 | 2086710 | 2086709 | -1 |
| 2087618 | 2087619 | -1 | -1 |
| 2125665 | 2125666 | -1 | -1 |

undefined target anchors are represented by -1 in the third column; even though no keywords were set for the query, the fourth column is included in the dump table with all values set to -1

- the table created by the dump command is printed on stdout by default (where it is parsed by the CWB/Perl interface); it can also be redirected to a file or pipe; use the following command to create a compressed dump file:

```
> dump A > "gzip > dump.tbl.gz";
```

- sometimes it is desirable to reload a dump file into CQP after it has been modified by an external program; this functionality is provided by the undump command, which reads its input from a file or pipe

```
> undump B < "gzip -cd mydump.tbl.gz |"
```

this command creates a new named query result $B$ for the currently activated corpus, or overwrites an existing one

- the format for undump tables is almost identical to the dump format, with two exceptions: the first line must contain a single number specifying the total number of rows in the table (i.e. the number of matches), and the following table has only two columns (for the match and matchend anchors) by default; the example below is a valid undump file for the DICKENS corpus, creating a query result with 5 matches

```
5
20681 20687
379735 379741
1915978 1915983
```

- since this command respects the current sort order of the query results, the natural order should first be restored by calling sort without a by clause; One exception is a formatter for a CGI interface to CQP, which usually displays matches in their sorted order.

- create .cqprc file in your home directory with your favourite settings (contains arbitrary CQP commands that will be read and executed during startup)
- for a persistent command history, add the lines

```
set HistoryFile "<home>/.cqphistory";
set WriteHistory yes;
```

to your .cqprc file (if CQP is run with -e option)

NB: the size of the history file is not limited automatically by CQP
- set AutoShow off;

- no automatic KWIC display of query results
- set Optimize on;

- enable experimental optimisations (sometimes included in beta versions)

2.5 Accessing token-level annotations

- specify p-attribute/value pairs (brackets are required)

```
> [pos = "JJ"];
```

( find adjectives)

```
> [lemma = "go"];
```

- "interesting" is an abbreviation for [word = "interesting"]

- the implicit attribute in the abbreviated form can be changed with the DefaultNonbrackAttr option; for instance, enter

```
> set DefaultNonbrackAttr lemma;
```

to search for lemmatised words instead of surface forms

- &c and %d flags can be used with any attribute/value pair

```
> [lemma = "pole" %c];
```

- values are interpreted as regular expressions, which the annotation string must match; add %l flag to match literally:

```
> [word = "?" %l];
```

- != operator: annotation must not match regular expression

```
> [pos != "N.*"]
```

- everything except nouns

- [] matches any token (⇒ matchall pattern)

- see Appendix A.2 for a list of useful part-of-speech tags and regular expressions

- or find out with the /codist[] macro (more on macros in Sections 5.4 and 5.5):

```
> /codist[lemma, "go", word];
```

- finds all tokens whose lemma attribute has the value go and computes frequency distribution of the corresponding word forms
6.3 Running CQP as a backend

- CQP is a useful tool for interactive work, but many tasks become tedious when they have to be carried out by hand; macros can be used as templates, providing some relief, however, full scripting is still desirable (and in some cases essential)
- similarly, the output of CQP requires post-processing at times: better formatting of KWIC lines (especially for HTML output), different sort options for frequency tables, frequency counts on normalised word forms (or other transformations of the values)
- for both purposes, an external scripting tool or programming language is required, which has to interact dynamically with CQP (which acts as a query engine)
- CQP provides some support for such interfaces: when invoked with the -c flag, it switches to child mode (which could also be called “slave” mode):
  - the init file ~/.cqprc is not automatically read at startup
  - CQP prints its version number after initialisation
  - all interactive features are deactivated (paged display and highlighting)
  - query results are not automatically displayed (setAutoShowoff;)
  - after the execution of a command, CQP prints an empty line &
    - in case of a syntax error, the string PARSEERROR is printed on stderr
  - the special command .EOL.; inserts the line
    - as a marker into CQP’s output
  - running CQP as a backend can be a security risk, e.g. when queries submitted to a Web server are passed through to the CQP process unaltered; this may allow malicious users to execute arbitrary shell commands on the Web server; as a safeguard against such attacks, CQP provides a query lock mode, which allows only queries to be executed, while all other commands (including cat, sort, group, etc.) are blocked
  - the query lock mode is activated with the command
    - the lock mode is activated with the command
       > set QueryLock n;
      where n is a randomly chosen integer number; it can only be de-activated with
       > unlock n;
      using the same number n
6.2 Labels and scope

- returning to the np_agr macro from Section 6.1, we note a problem with this query:
  > A = /np_agr[] [pos = "VVFIN"] /np_agr[];
when the second NP does not contain any adjectives but the first does, the b label will still point to an adjective in the first NP; consequently, the agreement check may fail even if both NPs are really valid.

- in order to solve this problem, the two NPs should use different labels; for his purpose, every macro has an implicit $SS$ argument, which is set to a unique value for each interpolation of the macro; in this way, we can construct unique labels for each NP:

```
DEFINE MACRO np_agr(0)
  $$a:[pos="ART"]
  $$b:[pos="ADJA"]
  $$c:[pos="NN"]
  [: ambiguity(/unify[agr, $$a, $$b, $$c]) >= 1 :
  ];
```

a comparison with the previous results shows that this version of the /np_agr[] macro finds additional matches that were incorrectly rejected by the first implementation.

```plaintext
> B = /np_agr[] [pos = "VVFIN"] /np_agr[];
> diff B A;
```

- however, the problem still persists in queries where the macro is interpolated only once, but may be matched multiple times:
  > A = ( /np_agr[] ) (3);
here, a solution is only possible when the scope of labels can be limited to the body of the macro in which they are defined; i.e., the labels must be reset to undefined values at the end of the macro block; this can be achieved with the built-in /undef[] macro, which resets the labels passed as arguments and returns a true value:

```
DEFINE MACRO np_agr(0)
  a:[pos="ART"]
  b:[pos="ADJA"]
  c:[pos="NN"]
  [: ambiguity(/unify[agr, a, b, c]) >= 1 :
  [: /undef[a, b, c] :
  ];
```

```plaintext
> B = ( /np_agr[] ) (3);
> diff B A;
```

- note that it may still be wise to construct unique label names (either in the form np_agr_a etc., or with the implicit $SS$ argument) in order to avoid conflicts with labels defined in other macros or in the top-level query:

```
DEFINE MACRO np_agr(0)
  a:[pos="ART"]
  b:[pos="ADJA"]
  c:[pos="NN"]
  [: ambiguity(/unify[agr, a, b, c]) >= 1 :
  [: /undef[a, b, c] :
  ];
```

```plaintext
> B = ( /np_agr[] ) (3);
> diff B A;
```
6 Undocumented CQP

6.1 Zero-width assertions

- constraints involving labels have to be tested either in the global constraint or in one of the token patterns; this means that macros cannot easily specify constraints on the labels they define: such a macro would have to be interpolated in two separate places (in the sequence of token patterns as well as in the global constraint)
- zero-width assertions allow constraints to be tested during query evaluation, i.e. at a specific point in the sequence of token patterns; an assertion uses the same Boolean expression syntax as a pattern, but is delimited by \[:...:\] rather than simple square brackets ([...]); unlike an ordinary pattern, an assertion does not “consume” a token when it is matched; it can be thought of as a part of the global constraint that is tested in between two tokens
- with the help of assertions, NPs with agreement checks can be encapsulated in a macro

```
DEFINE MACRO np_agr(0)
a:[pos="ART"] b:[pos="ADJA"]
c:[pos="NN"]
[:ambiguity(/unify[agr,a,b,c])>=1:

```

(in this simple case, the constraint could also have been added to the last pattern)

- when the *this* label (\_) is used in an assertion, it refers to the corpus position of the following token; the same holds for direct references to attributes
- in this way, assertions can be used as look-ahead constraints, e.g. to match maximal sequences of tokens without activating longest match strategy

```
> [pos="NNS?"]{2,}[:pos!="NNS?":]
```

- assertions also allow the independent combination of multiple constraints that are applied to a single token; for instance, the `region(5)` macro from Section 5.5 could also have been defined as

```
MACRO region($0=Att$1=Key1$2=Val1 $3=Key2 $4=Val12)
<$0> [: ...$0,$1="$2":] [: ...$0,$3="$4":] []* /$0>
```

- like the matchall pattern [], the matchall assertion [::] is always satisfied; since it does not “consume” a token either, it is a no-op that can freely be inserted at any point in a query expression; in this way, a label or target marker can be added to positions which are otherwise not accessible, e.g. an XML tag or the start/end position of a disjunction

```
> ... a[::] /region[np] ... ;
```

starting a query with a matchall assertion is extremely inefficient: use the match anchor or the implicit match label instead
5 ADVANCED CQP FEATURES

- match set members against regular expression
  > [ (pos \= "NN") & (agr matches \".*:Pl:.*\") ];
  \(\rightarrow\) nouns which are uniquely identified as plurals
- both \texttt{contains} and \texttt{matches} use regular expressions and accept the \%c and \%d flags
- unification of agreement features \(\iff\) intersection of feature sets
- use built-in /\texttt{unify[\]} macro:
  /\texttt{unify[agr, \{label1\}, \{label2\}, \ldots]}\n- undefined labels will automatically be ignored
  > a:\{pos\=\"ART\"\} b:\{pos\=\"ADJA\"\}? c:\{pos\=\"NN\"\}
  \(\rightarrow\) (simple) NPs uniquely identified as genitive
- in the GERMAN-LAW corpus, NPs and other phrases are annotated with partially disambiguated agreement information; these features sets can also be tested with the \texttt{contains} and \texttt{matches} operators, either indirectly through label references or directly in XML start tags
  > /\texttt{region[np, a]} :: a.np\_agr matches \"Dat::Pl::\";
  > /\texttt{np\_agr matches \"Dat::Pl::\"} \[\\]* </\texttt{np\_agr}>

3 Working with query results

3.1 Named query results

- store query result in memory under specified name (should begin with capital letter)
  > Go = \[lemma = \"go\"\] \"and\" \[\];
  \(\rightarrow\) note that query results are \textit{not} automatically displayed in this case
- list named query results
  > show named;
- result of last query is implicitly named \texttt{Last}; commands such as \texttt{cat}, \texttt{sort}, and \texttt{count} operate on \texttt{Last} by default; note that \texttt{Last} is always temporary and will be overwritten when a new query is executed (or a \texttt{subset} command, cf. Section 3.5)
- display number of results
  > size Go;
- (full or partial) KWIC display
  > cat Go;
  > cat Go 5 9; \(\text{(6}^{th} - 10^{th} \text{ match})\)
- sorting a named query result automatically re-displays the matches
  > sort Go by lemma \%cd;
- the \texttt{count} command also sorts the named query on which it operates:
  > count Go by lemma cut 5;
  \(\rightarrow\) implicitly executes the command \texttt{sort Go by lemma};
- this has the advantage that identical word sequences now appear on adjacent lines in the KWIC display and can easily be printed with a single \texttt{cat} command; the respective line numbers are shown in square brackets at the end of each line in the frequency listing

\begin{verbatim}
13  go and see \[\#128-\#140\]
10  go and sit \[\#144-\#153\]
  go and do \[\#29-\#37\]
  go and fetch \[\#42-\#48\]
  go and look \[\#87-\#93\]
  go and play \[\#107-\#113\]
\end{verbatim}

\(\rightarrow\) to display occurrences of \texttt{go and see}, enter
  > cat Go 128 140;

3.2 Saving data to disk

- named query results can be stored on disk in the \texttt{DataDirectory}
  > set DataDirectory \".\";
  > DICKENS;
  \(\text{NB: you need to re-activate your working corpus after setting the DataDirectory option}\)
3.3 Anchor points

- the result of a (complex) query is a list of token sequences of variable length (**matches**)

- each match is represented by two **anchor points**: 
  - match (corpus position of first token) and matchend (corpus position of last token)

- set additional **target** anchor with @ marker in query (prepended to a pattern)

- **only a single token can be marked as target**: if multiple @ markers are used (or if the marker is in the scope of a repetition operator such as +), only the rightmost matching token will be marked
MACRO np_start()
    (<np1>|<np2>)
    ;

MACRO np_end()
    (</np2>|</np1>|</np>)
    ;

MACRO np()
    (/np_start[][]) /np_end[])
    ;

Figure 6: Macro definition file for accessing embedded noun phrases.

\begin{itemize}
\item extend \texttt{/region[]} macro to embedded regions:
\begin{verbatim}
MACRO anyregion($0=Tag)
    (<$0>|<$01>|<$02>)[]*</$02>|</$01>|</$0>)
    ;
\end{verbatim}
\end{itemize}

\begin{itemize}
\item extend \texttt{/codist[]} macro to two constraints:
\begin{verbatim}
MACRO codist($0=Att1 $1=V1 $2=Att2 $3=V2 $4=Att3)
    _Results = [($0 = "$1") & ($2 = "$3")];
group _Results match $4;
discard _Results;
    ;
\end{verbatim}
\end{itemize}

\begin{itemize}
\item usage examples:
\begin{verbatim}
> "man" /anyregion[pp];
> /codist[lemma, "go", pos, "V.*", word];
\end{verbatim}
\end{itemize}

\begin{itemize}
\item the simple string interpolation of macros allows some neat tricks, e.g. a \texttt{region} macro with constraints on key-value pairs in the start tag
\begin{verbatim}
MACRO region($0=Att $1=Key $2=Val)
    <$0.$1 = "$2"> []* </$0.$1>
    ;

MACRO region($0=Att $1=Key1 $2=Val1 $3=Key2 $4=Val2)
    <$0.$1 = "$2"> <$0.$3 = "$4"> []* </$0.$3></$0.$1>
    ;
\end{verbatim}
\end{itemize}

5.6 Feature set attributes (GERMAN-LAW)

\begin{itemize}
\item feature set attributes use special notation, separating set members by | characters
\item e.g. for the \texttt{alemma} (ambiguous lemma) attribute
\item when \texttt{targeted} pattern is optional, check how many matches have target anchor set
\begin{verbatim}
> A = [pos="DT"] 0[pos="JJ"]? [pos="NNS?"];
> size A;
> size A target;
\end{verbatim}
\item anchor points allow a flexible specification of sort keys with the general form
\begin{verbatim}
> sort by attribute on start point .. end point;
\end{verbatim}
both \texttt{start point} and \texttt{end point} are specified as an anchor, plus an optional offset in square brackets; for instance, \texttt{match[-1]} refers to the token before the start of the match, \texttt{matchend} to the last token of the match, \texttt{matchend[1]} to the first token after the match, and \texttt{target[-2]} to a position two tokens left from the \texttt{target} anchor.
\item NB: the \texttt{target} anchor should only be used in the sort key when it is always defined
\item example: sort noun phrases by adjectives between determiner and noun
\begin{verbatim}
> [pos="DT"] [pos="JJ"]{2,} [pos="NNS?"];
> sort by word \%cd on match[1] .. matchend[-1];
\end{verbatim}
\item if \texttt{end point} refers to a corpus position before \texttt{start point}, the tokens in the sort keys are compared from right to left; e.g. sort on the left context of the match (by \texttt{token})
\begin{verbatim}
> sort by word \%cd on match[-1] .. match[-42];
\end{verbatim}
whereas the \texttt{reverse} option sorts on the left context by \texttt{character}
\begin{verbatim}
> sort by word \%cd on match[-42] .. match[-1] reverse;
\end{verbatim}
\item complex sort operations can sometimes be speeded up by using an external helper program (the standard Unix \texttt{sort} tool)
\begin{verbatim}
> sort by word \%cd;
> set ExternalSort on;
> sort by word \%cd;
> set ExternalSort off;
\end{verbatim}
\item the \texttt{count} command accepts the same specification for the strings to be counted
\begin{verbatim}
> count by lemma on match[1] .. matchend[-1];
\end{verbatim}
\item display corpus positions of all anchor points in tabular format
\begin{verbatim}
> A = "behind" 0[pos="JJ"]? [pos="NNS?"];
> dump A;
> dump A 9 14;
\end{verbatim}
the four columns correspond to the \texttt{match}, \texttt{matchend}, \texttt{target} and \texttt{keyword} (see Section 3.6) anchors; a value of \texttt{-1} means that the anchor has not been set
\begin{verbatim}
1019887 1019888 -1 -1
1924977 1924979 1924978 -1
1986623 1986624 -1 -1
2086708 2086710 2086709 -1
\end{verbatim}
\item \texttt{External sorting} may also allow language-specific sort order (\texttt{collation}) if supported by the system's \texttt{sort} command. To achieve this, set the \texttt{LC_COLLATE} or \texttt{LC_ALL} environment variable to an appropriate locale before running CQP. You should not use the \texttt{\%c} and \texttt{\%d} flags in this case.
\end{itemize}
note that a previous sort or count command affects the ordering of the rows (so that
the n-th row corresponds to the n-th line in a KWIC display obtained with cat)

- the output of a dump command can be written (>) or appended (>>) to a file, if the first
character of the filename is #, the output is sent to the pipe consisting of the following
command(s): use the following trick to display the distribution of match lengths in the
query result A:

  > A = [pos="DT"] [pos="JJ.*"]* [pos="NNS?"];
  > dump A | gawk '{print$2-$1+1}' | sort -nr | uniq -c | less;

- see Section 6.4 for an opposite to the dump command, which may be useful for certain
tasks such as locating a specific corpus position

3.4 Frequency distributions

- frequency distribution of tokens (or their annotations) at anchor points

  > group Go matchend pos;
  > set cutoff threshold with cut option to reduce size of frequency table

  > NP = [pos="DT"] @[pos="JJ"]? [pos="NNS?"];
  > group NP target lemma cut 50;

- add optional offset to anchor point, e.g. distribution of words preceding matches

  > group NP match[-1] lemma cut 100;

- frequencies of token/annotation pairs (using different attributes or anchor points)

  > group NP matchend word by target lemma;
  > group Go matchend lemma by matchend pos;

NB: despite what the command syntax and output format suggest, results are sorted by
pair frequencies (not grouped by the second item); also note that the order of the
two items in the output is opposite to the order in the group command

- you can write the output of the group command to a text file (or pipe)

  > group NP target lemma cut 10 > "adjectives.go";

3.5 Set operations with named query results

- named queries can be copied, especially before destructive modification (see below)

  > B = A;
  > C = Last;

- compute subset of named query result by constraint on one of the anchor points

  > PP = [pos="IN"] [pos="JJ"]+ [pos="NNS?"];
  > group PP matchend lemma by match word;
  > PP1 = subset PP where match: "in";
  > PP2 = subset PP1 where matchend: [lemma = "time"];
  > PP2 contains instances of in ... time(s)

4.5 CQP macro examples

- use macros for easier access to embedded noun phrases (NP)

  write and load the macro definition file shown in Figure 6

- then use /np_start[] and /np_end[] instead of <np> and </np> tags in CQP queries,
as well as /np[] instead of /region[np]

  > /np_start[] /np[] "and" /np[] /np_end[];

- CQP ensures that the "generalised" start and end tags nest properly
  (if the StrictRegions option is enabled, cf. Sections 4.2 and 4.3)

- extending built-in macros: view definitions

  > show macro region(1);
  > show macro codist(3);
Adv. CQP features

- Macros accept up to 10 arguments; in the macro definition, the number of arguments must be specified in parentheses after the macro name.
- In the macro body, each occurrence of $0$, $1$, ... is replaced by the corresponding argument value (escapes such as \$1 will not be recognised).
- e.g. a simple PP macro with 2 arguments: the initial preposition and the number of adjectives in the embedded noun phrase:

```cqp
MACRO pp(2)
    [pos = "IN"] & (word="$0")
    [pos = "DT"]
    [pos = "JJ.*"]($1)
    [pos = "NNS?"]

```
- Invoking macros with arguments:

```cqp
> /pp["under", 2];
> /pp["in", 3];
```
- Macro arguments are character strings and must be enclosed in (single or double) quotes; they may be omitted around numbers and simple identifiers.
- The quotes are not part of the argument value and hence will not be interpolated into the macro body; nested macro invocations will have to specify additional quotes.
- Define macro with prototype ⇒ named arguments

```cqp
MACRO pp ($0=Prep $1=N_Adj)
    ...
    
```
- Argument names serve as reminders; they are used by the `show` command and the macro name completion function (TAB key).
- Argument names are not used during macro definition and evaluation.
- In interactive definitions, prototypes must be quoted:

```cqp
> define macro pp('"$0=Prep $1=N_Adj") ... ;
```
- CQP macros can be overloaded by the number of arguments (i.e. there can be several macros with the same name, but with different numbers of arguments).
- This feature is often used for unspecified or "default" values, e.g.

```cqp
MACRO pp($0=Prep, $1=N_Adj)
    ...
MACRO pp($0=Prep)  (any number of adjectives)
MACRO pp()  (any preposition, any number of adjs)
    ...
```
- Macro calls can be nested (non-recursively) ⇒ macro file defines a context-free grammar (CFG) without recursion (see Figure 5).

CQP Query Language Tutorial

- Set operations on named query results:

```cqp
> A = intersection B C;  A = B \ C
> A = union B C;  A = B \cup C
> A = difference B C;  A = B \ C
```
- Intersection (or inter) yields matches common to B and C; union (or join) matches from either B or C; difference (or diff) matches from B that are not in C.
- When there are a lot of matches, look at a random selection to get a quick overview:

```cqp
> A = "time";
> size A;
```
- The `reduce` command randomly selects a given number or proportion of matches, deleting all other matches from the named query; since this operation is destructive, it may be necessary to make a copy of the original query results first (see above):

```cqp
> reduce A to 10%;
> size A;
> sort A by word %cd on match .. matchend[42];
> reduce A to 100;
> size A;
```
- Set random number generator seed before `reduce` for reproducible selection:

```cqp
> randomize 42;
```
- The modifier `cut` can be appended to a query and performs a similar function: it returns approximately the first n results:

```cqp
> "time" cut 50;
```
- Since this is not a representative subset of the matches, the `cut` option should not be used for reducing large query results; its main purpose is to limit the number of query matches (and thus memory consumption) in Web interfaces and similar applications.

3.6 The `set target` command

- Additional keyword anchor can be set after query execution by searching for a token that matches a given `search pattern` (see Figure 3):

```cqp
> A = [(pos="JJ") & (lemma="modern")];
> set A keyword nearest [pos="NNS?"] within right 5 words from match;
```
- Keyword should be underlined in KWIC display (may not work on some terminals).
- Search starts from the given anchor point (excluding the anchored token itself), or from the left and right boundaries of the match if `match` is specified.
- With `inclusive`, search includes the anchored token, or the entire match, respectively.
- From `match` is the default and can be omitted.
5.4 The CQP macro language

- complex queries (or parts of queries) can be stored as macros and re-used
- define macros in text file (e.g. macros.txt):
  ```
  # this is a comment and will be ignored
  MACRO np(0)
  [pos = "DT"] # another comment
  ([pos = "RB.*"]? [pos = "JJ.*"])*
  [pos = "NNS?"]
  ;
  ```
  (defines macro "np" with no arguments)
- load macro definitions from file
  ```
  > define macro < "macros.txt";
  ```
- macro invocation as part of a CQP command (use TAB key for macro name completion)
  ```
  > <s> /np[ ]@[pos="VB.*"]/np[ ];
  ```
- list all defined macros or those with given prefix
  ```
  > show macro;
  > show macro region;
  ```
- show macro definition (you must specify the number of arguments)
  ```
  > show macro np(0);
  ```
- re-define macro interactively (must be written as a single line)
  ```
  > define macro np(0) '[pos="DT"] [pos="JJ.*"]+ [pos="NNS?"]';
  ```
  or re-load macro definition file
  ```
  > define macro < "macros.txt";
  ```
- macros are interpolated as plain strings (not as elements of a query expression) and may have to be enclosed in parentheses for proper scoping
  ```
  > <s> <\np[]>+ [pos="VB.*"];
  ```
- it is safest to put parentheses around macro definitions:
  ```
  MACRO np(0)
  {
  [pos = "DT"]
  ([pos = "RB.*"]? [pos = "JJ.*"])*
  [pos = "NNS?"]
  }
  ;
  ```

NB: The start (**MACRO**) and end (**;**) markers must be on separate lines in a macro definition file.
5 ADVANCED CQP FEATURES

---

4.1 Using labels

- patterns can be labelled (similar to the target marker @)
  
  > adj:[pos = "JJ.*"] ... ;  
  
  the label adj then refers to the corresponding token (i.e. its corpus position)
- label references are usually evaluated within the global constraint introduced by ::
  
  > adj:[pos = "ADJ.*"] :: adj < 500;  
  
  -> adjectives among the first 500 tokens
- annotations of the referenced token can be accessed as adj.word, adj.lemma, etc.
- labels set to optional patterns may be undefined
  
  > [pos="DT"] a:[pos="JJ"]? [pos="NNS?" ] :: a;  
  
  ! a is true i match contains an adjective
- labels are not part of the query result and must be used within the query expression (otherwise, CQP will abort with an error message)
- labels to optional patterns may be undefined
  
  > [pos="DT"] a:[pos="JJ"]? [pos="NNS?" ] :: a;  
  
  ! a is true i match contains an adjective
- labels used to specify additional constraints that are beyond the scope of ordinary regular expressions
  
  > a:[ ]) "and" b:[ )] :: a.word = b.word;  
  
  ! labels allow modelling of long-distance dependencies
- labels can be used within patterns as well
  
  > a:[ ] pos = "NPS";  
  
  ! labels can be used within patterns as well
- labels allow modelling of long-distance dependencies
  
  > a:[pos="PP"] ?(0,5) b:[pos="VB.*"]  
  
  ! this query ensures that the pronoun preceding a 3rd-person singular verb form is he, she or it; an additional constraint could exclude these pronouns for other verb forms
- labels can be used within patterns as well
  
  > a:[ ] pos = "NPS";  
  
  ! sequences of four identical part-of-speech tags
- however, a label cannot be used within the pattern it refers to; use the special this label represented by a single underscore (_) instead to refer to the current corpus position
  
  > [.pos = "NPS"] => [.pos = "NPS"]  
  
  ! the built-in functions distance() and distabs() compute the (absolute) distance between 2 tokens (referenced by labels)
- the built-in functions distance() and distabs() compute the (absolute) distance between 2 tokens (referenced by labels)
  
  > a:[pos="DT"] [pos="JJ"]* b:[pos="NNS?" ] :: distabs(a,b) >= 5;  
  
  ! simple NPs containing 6 or more tokens
- the standard anchor points (match, matchend, and target) are also available as labels (with the same names)
  
  > a:[pos="DT"] [pos="JJ"]* b:[pos="NNS?" ] :: distabs(match, matchend) >= 5;

---

Stefan Evert 28 © 2005 IMS Stuttgart
4.2 Structural attributes

- XML tags match start/end of s-attribute region (shown as XML tags in Figure 1)
  > <$ [pos = "VBG"]
  > [pos = "VBG"] [pos = "SENT"]? </$>
  → present participle at start or end of sentence

- pairs of start/end tags enclose single region (if strictRegions option is enabled)
  > <np> [], [(pos="JJ.*") [], ]{3,} </np>
  → NP containing at least 3 adverbs

(when strictRegions are switched off, XML tags match any region boundaries and may skip intervening boundaries as well as material outside the corresponding regions)

- /region/ macro matches entire region
  /region[np]; ⇐< np> ]* </np>

- different tags can be mixed
  > <s><np> [[]* ]</np> [[]* </np></s>
  → sentence that starts and ends with a noun phrase (NP)

- the name of a structural attribute (e.g. np) used within a pattern evaluates to true if
  the corresponding token is contained in a region of this attribute (here, a <np> region)
  > [(pos = "NNS?") & !np]
  → noun that is not contained in a noun phrase (NP)

- built-in functions lbound() and rbound() test for start/end of a region
  > [(pos = "VBG") & lbound(s)]
  → present participle at start of sentence

- use within to restrict matches of a query to a single region
  > [pos="NN"] [], [pos="NN"] within np;
  → sequence of two singular nouns within the same NP

- most linguistic queries should include the restriction within a to avoid crossing sen-
  tence boundaries; note that only a single within clause may be specified

- query matches can be expanded to containing regions of s-attributes
  > A = [pos="JJ.*"] [[]* [pos="JJ.*"]]{2} within np;
  > B = A expand to np;
  one-sided expansion is selected with the optional left or right keyword
  > C = B expand left to s;

- the expansion can be combined with a query, following all other modifiers
  > [pos="JJ.*"] [[]* [pos="JJ.*"]]{2} within np cut 20 expand to np;

Figure 4: CQP matching strategies.
5 Advanced CQP features

5.1 The matching strategy

- set MatchingStrategy (shortest | standard | longest);
- in shortest mode, ?, * and + operators match smallest number of tokens possible (refers to regular expressions at token level)
  - finds shortest sequence matching query,
  - optional elements at the start or end of the query will never be included
- in longest mode, ?, * and + operators match as many tokens as possible
- in the default standard mode, CQP uses an “early match” strategy: optional elements at the start of the query are included, while those at the end are not
- the somewhat inconsistent matching strategy of earlier CQP versions is currently still available in the traditional mode, and can sometimes be useful (e.g. to extract cooccurrences between multiple adjectives in a noun phrase and the head noun)
  - [pos=“JJ”]+ [pos=“NNS?”];
  - group Last matchend lemma by match lemma;
  - only gives the intended frequency counts in traditional mode
- Figure 4 shows examples for all four matching strategies

5.2 Word lists

- word lists can be stored in variables
  - define $week = “Monday Tuesday Wednesday Thursday Friday”;
  - and used instead of regular expressions in the attribute/value pairs
  - $week = “Saturday Sunday”;
- show list of words stored in variable
  - show $week;
  - use show var; to see all variables
- read word list from file (one-word-per-line format)
  - define $week < “/home/weekdays.txt”;
  - use TAB key to complete word list names (e.g. type “show $ue” + TAB)
- word lists can be used to simulate type hierarchies, e.g. for part-of-speech tags
  - define $common_noun = “NN NNS”;
  - define $proper_noun = “NP NPS”;
  - define $noun = $common_noun;
  - define $noun += $proper_noun;

4.3 Structural attributes and XML

- XML markup of NPs and PPs in the DICKENS corpus (cf. Appendix A.3)

<s len=9>
  <np h=“it” len=1> It </np>
</s>

- constraints on key-value pairs can also directly be tested in start tags, using the appropriate auto-generated s-attribute (make sure to use a matching end tag)
  - <np_h=“bank”/></np_h>;
- comparison operators = and != are supported, together with the %c and %d tags;
- constraints on multiple key-value pairs require multiple start tags
  - <np_h=“bank”><np_len=“[1-6]”></np_len></np_h>;
  - (or access the value of np_len through a label reference)
- <np> and <pp> tags are usually shown without XML attribute values; they can be displayed explicitly as <np_h>, <np_len>... tags:
  - use this label for direct access to s-attribute values within pattern
    - [(pos=“NNS?”) & (lemma = _.np_h)];
    - (recall that np_h would merely return an integer value indicating whether the current token is contained in a <np> region, not the desired annotation string)
• typecast numbers to \texttt{int()} for numerical comparison
  \begin{verbatim}
  > /region[np,a] :: int(a.np_len) \geq 30;
  \end{verbatim}
• NB: s-attribute annotations can only be accessed with label references:
  \begin{verbatim}
  > [np_h="bank"] ;
  \end{verbatim}
  \textit{does not work!}
• regions of structural attributes are non-recursive
  \begin{verbatim}
  => embedded XML regions are renamed to <np1>, <np2>, ..., <pp1>, <pp2>, ...
  \end{verbatim}
  \textit{embedding level must be explicitly specified in the query:}
  \begin{verbatim}
  > [pos="NP"] <np1> \* </np1> ;
  \end{verbatim}
  will only find NPs contained in \textit{exactly one} larger NP
  \textit{(use show +np +np1 +np2; to experiment)}
• regions representing the attributes in XML start tags are renamed as well:
  \begin{verbatim}
  => <np_h1>, <np_h2>, ..., <pp_len1>, <pp_len2>, ...
  \end{verbatim}
  \begin{verbatim}
  > /region[np1, a] :: a.np_h1 = a.np_h within np;
  \end{verbatim}
• CQP queries typically use maximal NP and PP regions (e.g. to model clauses)
  \begin{verbatim}
  find any NP (regardless of embedding level):
  > (<np>|<np1>|<np2>) \* (<np2>|<np1>|</np>) ;
  \texttt{CQP ensures that a matching pair of start and end tag is picked from the alternatives}
\end{verbatim}
• observe how results depend on matching strategy (see Section 5.1 for details)
  \begin{verbatim}
  > set MatchingStrategy shortest;
  > set MatchingStrategy longest;
  > set MatchingStrategy standard;
  \texttt{(re-run the previous query after each set and watch out for “duplicate” matches)}
\end{verbatim}
• when the query expression shown above is embedded in a longer query, the matching strategy usually has no influence
• annotations of a region at an arbitrary embedding level can only be accessed through constraints on key-value pairs in the start tags:
  \begin{verbatim}
  > (<np_h "bank">|<np_h1 "bank">|<np_h2 "bank">) \* (<np_h2>|<np_h1>|</np_h>);
  \end{verbatim}

4.4 XML document structure

• XML document structure of DICKENS:
  \begin{verbatim}
  <novel title="A Tale of Two Cities">...
  </titlepage> ...
  <book num=1>
    <chapter num=1 title="The Period">
      ...
    </chapter>
  ...
  </book>
  \end{verbatim}