The IMS Corpus Workbench:

Corpus Query Processor (CQP)

User's Manual

Oliver Christ, Bruno M. Schulze, Anja Hofmann, and Esther König

Institute for Natural Language Processing
Azenbergstr. 12, 70174 Stuttgart, Germany
`http://www.ims.uni-stuttgart.de/CorpusWorkbench/`

August 16, 1999 (CQP V2.2)

Contents

Chapter 1

Introduction

The IMS Corpus Workbench is a set of tools for the manipulation of large, linguistically annotated text corpora. One of the tools is the IMS Corpus Query Processor `CQP` , a specialized search engine for linguistic research.

This manual explains how information can be extracted from text corpora by the use of the `CQP`-query language and by some additional commands and parameters. `CQP` assumes that corpora have been indexed beforehand. The indexing and general administration of corpora is described in a separate volume, the `CQP`-Administrator's Manual [2]. The philosophy and the overall architecture of the Corpus Workbench has been presented in [1, 3].

1.1 Corpora, Annotations, Queries, and Results

The most basic version of a *text corpus* (or *corpus*) is just a sequence of *words* (or *tokens*). An individual occurrence of a word determines a *corpus position* (which could be represented by a number). In order to make a corpus more useful for linguistic analysis, a corpus can be annotated at least in three dimensions.

Positional annotation

> The individual corpus positions can be annotated with any number of *positional attributes* like part-of-speech (POS) information, morphosyntactic information, base forms (lemmata), ...

Structural annotation

> *Structural attributes* can be inserted into the corpus, e.g. paragraph or sentence boundaries or even more fine-grained phrase boundaries.

Multilevel annotation

> A corpus can be *aligned* with another corpus, e.g. with its translation, or with other representations of the sequence of tokens, e.g. with its intonation patterns or discourse relations. Multilevel annotation of corpora can be seen as a generalization of (hierarchical) structural annotation, since it must be possible to associate overlapping and/or non-adjacent sections of two corpora or annotation levels.

In our philosphy, we use the term *corpus* as a shorthand for *annotated corpus*.

`CQP` allows for *queries* to corpora along the above mentioned dimensions, at least to a certain extent:

Access to positional annotations

- **Single corpus positions**

  The values of positional attributes (where `word` is just one, prominent attribute) can be queried by regular expressions over characters. Constraints on different attributes for the same corpus position can be combined into Boolean expressions.

- **Sequences of corpus positions**

  Queries to single corpus positions can be embedded into Boolean expressions over regular expressions (over corpus positions) in order to describe sequences of corpus positions.

Access to structural annotations

Structural annotations are treated like corpus positions, i.e. they can be part of a query to a sequence of corpus positions. However, neither recursively embedded nor overlapping annotations are supported by `CQP`.

Access to multilevel annotations

`CQP` supports queries to aligned corpora.

In `CQP`, the *result* of a query is the list of *corpus intervals* which match the given query. Such a corpus interval is also called a *match*. In a sense, queries define additional structural boundaries on a corpus, the so-called *ad-hoc structures*. These ad-hoc structures can be referred to in subsequent queries, due to the concept of *query names*. Named queries are the basis for incremental corpus exploration.

Query results can be displayed as plain text, HTML, or Latex - with or without the corresponding corpus annotations. In addition, `CQP` supports the alphabetical and frequency-based sorting of query results.

1.2 Organization of this manual

This manual is organized as follows. The next chapter 2 describes the basic commands of `CQP`. The following chapters introduce the query language of `CQP`: access to single corpus positions (chapter 3), to sequences of corpus positions (chapter 4), to structural attributes (chapter 5), and to aligned corpora (chapter 6). The manual concludes with the description of the sorting routines of `CQP` (chapter 7).

Chapter 2

Basic interaction with CQP

This chapter explains how to call the CQP-system and how to leave it again. It tells how a corpus can be selected, gives a sample query, and explains how to display and save query results.

2.1 Starting and leaving CQP

CQP is started by simply typing the command

```
cqp
```

at the shell prompt. There are a number of command line options and Unix system variables which affect the behavior of cqp. These are described in appendix A.

You leave CQP simply by typing

```
exit;
```

or by using Control-D as a shortcut.

2.2 Selecting a corpus

Before you can run a query, you have to select a corpus. You can find out which system corpora are accessible by typing the CQP command

```
show system;
```

or short:

```
show;
```

In the case that you cannot find the corpus for which you are looking, please check the paths of the *corpus registries*, i.e. the CQP system variable Registry, by typing

```
set Registry
```

You can remedy the situation for the current `CQP` session by changing the value of the `Registry` variable. E.g. if your corpus registries are located in the directories `/corpora/registry` and `~/registry` then you have to issue the command (see also section A.1):

```
set Registry "/corpora/registry:~/registry";
```

If you are lucky, your corpus administrators have provided some *corpus information*. Typically, this information consists of a copyright notice, a description of the corpus annotation and of the composition of the corpus. E.g. for the corpus named `BNC`, `CQP` will display this kind of information (if available) if you use the `info`-command in the following manner

```
info BNC;
```

Note: Each `CQP` command must be completed with a semicolon!

In order to select a corpus, say `UP`, which is the part-of-speech tagged Penn Treebank corpus of the University of Pennsylvania, just enter

```
UP;
```

If the corpus is not available, `CQP` will return an error message. After the successful selection of the corpus `UP`, the command line prompt of `CQP` changes to

```
UP>
```

This means that the next query will be evaluated on that corpus.

2.3 A simple query

If you want to look for a single word, say "`Clinton`", just type "`Clinton`" at the `CQP` prompt.

```
UP> "Clinton";
```

The double quotes are important! Within the double quotes, case and blanks are significant!

After a short time you get the following output:

```
  44963:  Utah , $ 15,000 fine ; <Clinton> P. Hayne , New Orleans , $ 7,500 fine
 653115: n education , Gov. Bill <Clinton> of Arkansas announced that ‘ ‘ this
 653153: essing need . What Gov. <Clinton> is advocating , in effect , is extending the
 894253: nter trading . .TXT .PP <Clinton> Gas Systems Inc. said it received a
 894292: nnsylvania plants . .PP <Clinton> and Timken agreed not to disclose the
 894320: , already buys gas from <Clinton> . .PP Clinton said in Columbus ,
 894323:  gas from Clinton . .PP <Clinton> said in Columbus , Ohio , that its Clinton
 894332: umbus , Ohio , that its <Clinton> Gas Marketing unit wants to line up
 894362: ialist in natural gas , <Clinton> said , and a specialist such as Clinton
 894370: nd a specialist such as <Clinton> can save them substantial amounts of
1681563: ’s , currently owned by <Clinton> Holdings Inc. and affiliates ,
1826420:  Hayes National Bank in <Clinton> . The bank holding company signed an
```

This output is called the *result* of the query. It consists of a list of individual lines. Each line has the following shape: The angle brackets $<\ldots>$ mark the *corpus interval* which matches the query, for short, the *match*. The text on both sides of the match is called the *context* of the match. Historically, this output format of concordances is called *kwic format* which stands for *keyword in context*. In our example, the keyword is "`Clinton`". In `CQP`, kwic lines should rather be called 'match in context' since `CQP` search is not restricted to single corpus positions.

2.4 Displaying a query result

This section tells you how to display query results and other `CQP`-interaction. `CQP` allows for changing the browsing method, the size of the result, the size of context, the display of annotations, and the text filters for the output to a file.

2.4.1 Setting the `AutoShow` system variable

It depends on the value of the `CQP` *system variable* `AutoShow` whether a query result is displayed immediately after the evaluation of the query, or only on demand. One of the numerous applications of the `set`-command is to show the contents of a system variable. The command

```
set;
```

or more precisely

```
set AutoShow;
```

or, since case doesn't matter in this case:

```
set autoshow;
```

will tell you, whether `AutoShow` has been switched on or off. The set of legal values for *flag-like system variables* is usually the following: `yes`, `no`, `on`, `off`. So, in the case that you don't want to see the query results automatically, say

```
set autoshow no;
```

`AutoShow` can be switched on again by

```
set autoshow yes;
```

2.4.2 Basic display method

Unless the system variable `AutoShow` has been switched off (see section 2.4.1), the result of a query is shown immediately after the evaluation of the query. If the variable has been switched off, or if you have got lost in a lengthy query result, the result of the last query can be *displayed* (again) by the `cat`-command:

```
cat;
```

or

```
cat Last;
```

`Last` is the *query name* which is associated automatically with the most recent query.

2.4.3 Changing the browsing method

If the `CQP` variable `Paging` has been switched to `yes`, the output of the `cat` command will be sent to the Unix browsing tool which is defined by the `CQP` variable `Pager`. The output of the `cat` command is flushed to the screen, without user interaction, if `Paging` has been desactivated by the command:

```
set Paging no;
```

You can choose another pager program, e.g. the `more` program, by changing the value of the `Pager` variable (see also section A.1).

```
set Pager more;
```

You can switch off the *highlighting* of matches by setting the value of the `Highlighting` variable, appropriately.

```
set Highlighting no;
```

2.4.4 Restricting the size of the result

Often, the number of matches is enormous, so the user may be satisfied to see only a few matches. For this purpose, the `cut` and the `reduce` operators have been introduced.

Restricted search

The `cut` operator takes a query on its left side and a positive integer n on its right side, i.e.

Query `cut` n;

Only the n initial matches of the query result will be calculated. For example, the result of the following query will be cut after 10 matches.

```
"Bill" cut 10;
```

Random reduction of a result

The `reduce` operator applies after the computation of a query result. It randomly reduces the query result to either a given number of concordance lines or to a certain percentage. For example, the statements

```
reduce to 5;
```

or

```
reduce Last to 5;
```

will discard all but 5 randomly selected lines from the last query result.

```
reduce to 10%;
```

or

```
reduce Last to 10%;
```

will randomly discard all but 10% of the matches from the last query result. In order to see the effect, you have to have displayed the, now reduced query result, once more (see section 2.4.2).

2.4.5 Changing the context size

The *size of the context* on the left and the right side of a match can be adapted by manipulating the `CQP` variables `LeftContext`, `RightContext`, and `Context`.

The following command will set the context on each side of the match to five characters.

```
set Context 5;
```

This is equivalent to

```
set LeftContext 5;
set RightContext 5;
```

If you mean five words, you must mention that explicitly.

```
set Context 5 word;
```

If your text comes with structural attributes, for example with the structural attribute `s`, you may have the context extended to the next `s` tag by stating:

```
set Context 1 s;
```

2.4.6 Displaying corpus annotations

Positional attributes

Say, if the corpus you work on has been annotated with a `pos`-attribute (part-of-speech annotation), you can switch on the display of that attribute by

```
show +pos;
```

Similarly for other attributes like `lemma`, ...

After that, the result for our simple query

```
"Clinton";
```

will be shown with the `pos`-annotation.

```
44963: /$ 15,000/CD fine/NN ;/:  <Clinton/NP> P./NP Hayne/NP ,/, New/N
653115: n/NN ,/, Gov./NP Bill/NP  <Clinton/NP> of/IN Arkansas/NP announ
653153: N ./SENT What/WP Gov./NP  <Clinton/NP> is/VBZ advocating/VBG ,/
...
```

The display of the `pos`-attribute is switched off by the obvious command

```
show -pos;
```

If the display of all attributes has been switched off, or hasn't been set at all, only the word forms of the corpus will be shown.

Structural attributes

The display of structural attributes can be switched on in a similar manner as the positional attributes. If `s` is a structural attribute of the current corpus, its display is switched on by

```
show +s;
```

and switched off by

```
show -s;
```

If the structural attributes `s` and `article` have values, these values will be displayed in the query result after having issued the command:

```
set PrintStructures "s,article";
```

The display of tag values is switched off by

```
set PrintStructures "";
```

Multilevel annotation

For a corpus `HANSARD-E`, which is aligned e.g. on the sentence with its translation `HANSARD-F`, `CQP` shows the aligned contexts if you issue the appropriate `show` command.

```
show +hansard-f;
```

Note that, for some reasons, here, the command is case-sensitive and expects the name of the aligned corpus in lower-case letters!

The display of the aligned contexts is switched off by

```
show -hansard-f;
```

2.4.7 Sending a result to a file or a Unix pipe

The output of the `cat` command can be redirected to a file `/tmp/myResult` by the following commands.

```
cat > "/tmp/myResult";
```

resp.

```
cat Last > "/tmp/myResult";
```

This version of the command will overwrite the file `/tmp/myResult` if it already exists. If you want to *append* the query result to an existing file, you have to use

```
cat >> "/tmp/myResult";
```

The `cat` command is not intended to work for system corpora since it does not make sense to copy a whole corpus. Use the appropriate low-level commands of `CQP` instead which have been described [2]!

Redirection to a Unix pipe is possible, as well. For example

```
cat > "| a2ps -8 | lpr";
```

will pass the output to the ASCII-to-Postscript converter and afterwards to the printer. The append mode (`>>`) must not be used for output into pipes.

`Paging` is disabled during output to files or to pipes.

Print modes

If you want to insert query results into a publication, it is convenient to have `CQP` produce the result in a useful format like LaTeX code. (This option has been added in release 2.3.) This change of output format is caused by a `set PrintMode` command:

```
set PrintMode latex;
```

Instead of 'latex', there are also the values `html`, `sgml`, and `ascii`. The latter is the default output format of `CQP`. So, in order to get rid of the LaTeXor HTML formatting commands again, just say:

```
set PrintMode ascii;
```

Print options

Since `CQP` wants to be nice and informative, the file created by the `cat>` command will start with a *file header*. This header includes the query, size of the query result, etc. You can create headerless files due to the command

```
set PrintOptions noheader;
```

The header is switched on again with a command where the `no`-prefix is left out:

```
set PrintOptions header;
```

With the print option `number`, the result lines will be numbered.

```
set PrintOptions number;
```

For the `PrintMode ascii`, the left and right *match delimiters* can be adapted to the user's personal taste.

```
set LeftKWICDelim "[";
set RightKWICDelim "]";
```

In the case of `html` output (`PrintMode html`), there are also the print options `table` for creating an HTML table, `border` for inserting borders into HTML tables, and `wrap` for the wrapping of cells in HTML tables.

Several options can be set simultaneously if the list of values is inserted into quotation marks, e.g.

```
set PrintMode html;
set PrintOptions "table,border,wrap";
```

2.5 Profiling

2.5.1 Evaluation time

The computation time for a query will be shown if the `Timing` variable has been switched on.

```
set Timing yes;
```

2.5.2 User interaction

`CQP` will protocol automatically your interactive input to `CQP`, if you set the `CQP` variable `HistoryFile` to a file name

```
set HistoryFile "/tmp/myHistory";
```

and switch on the `WriteHistory` flag

```
set WriteHistory yes;
```

The history file will be re-opened each time a user interaction has been completed and will be closed afterwards, so that the history should survive a `CQP` crash.

`CQP` commands which have been read from a file (see appendix A) will not be protocoled in the history file.

2.5.3 Warnings and messages

`CQP` warnings and messages can be switched off by changing the value of the `Silent` variable:

```
set Silent yes;
```

For the interactive use of `CQP`, you'd better make sure that `Silent` is set to `no`:

```
set Silent no;
```

Chapter 3

Access to single corpus positions

This chapter explains the syntax of `CQP`-queries which refer to single corpus positions.

3.1 Representations of characters

For special characters such as the German umlauts, there are two representations. You can use the \ followed by the octal code of that character (in the character set of the corpus!), e.g.

```
"Sp\344tzle";
```

This query will retrieve occurrences of the South-West German word `Spätzle`. Alternatively, the LaTeX format of the special character can be used (see appendix B).

```
"Sp\"atzle";
```

As the above example indicates, a character such as the quotation mark which has a special meaning in `CQP` is interpreted *literally* if it is preceded by the backslash \.

Note that all characters between the double quotes count! E.g. the two queries

```
"Clinton";
```

```
"Clinton ";
```

are **not** identical since the second one requires the blank character to appear as the final character of the string!

3.2 Regular expressions over characters

If you do not know exactly how a word is spelled in the corpus, you can leave the spelling of the word 'underspecified' by stating a *regular expression*. `CQP` has adopted the POSIX egrep notation of regular expressions. This comprises the following operations: parentheses for marking embedded expressions, concatenation, disjunction, lists of alternative characters, unspecified character, optionality, Kleene star, and Kleene plus. Certain types of regular expressions can be abbreviated by the use of a 'flag'.

3.2.1 Embedded regular expressions

Subsequently, it may be necessary to mark embedded regular expressions. For this purpose, parentheses `(`, `)` are used.

3.2.2 Concatenation

Even the simple query like

```
"Clinton";
```

is an instance of a regular expression. It is formed by the *concatenation* of the characters `C`, `l`, `i`, `n`, `t`, `o`, and `n`. Concatenation is expressed by the juxtaposition of regular expressions.

3.2.3 Disjunction

Let's assume, that we want to find the occurrences of the English word "`the`", but we do want both, upper and lower case occurrences. This query can be expressed as

```
"(the)|(The)";
```

Here, the *disjunction operator* `|` lets `CQP` look for occurrences of the word form "`the`" and of the sentence initial form "`The`". The disjunction operator is an infix operator which takes two regular expressions as its arguments. Due to the bracketing conventions for the disjunction operator, the above query is equivalent to

```
"the|The";
```

By inserting parentheses again, the query can be reformulated more shortly as

```
"(t|T)he";
```

3.2.4 Lists of alternative characters

By using a *list of alternative characters*, the last query can be again rewritten.

```
"[tT]he";
```

For example, you can search for all occurrences of single digits in the corpus by the query

```
"[0123456789]";
```

This will retrieve any of the following tokens "`0`", "`1`", "`2`", ..."`9`". A shorter way to formulate the same query is

```
"[0-9]";
```

3.2.5 Unspecified character

Say, if you don't know whether the correct spelling is "`Velazquez`" or maybe "`Velasquez`", you would write

```
"Vela[zs]quez";
```

But you could also use the *unspecified character* `.` in the place of the list of alternative characters. This makes the query a bit more sloppy on the one hand, but on the other hand, this is more handy to write.

```
"Vela.quez";
```

The `.`-operator will match any character.

3.2.6 Optionality

You may want to find simultaneously the two word forms "`walk`" and "`walks`". Both word forms are captured by the regular expression

```
"walk(s)?";
```

The *optionality operator* `?` indicates that the preceding expression is optional. Since, by default, the `?` operator takes the preceding character as its operand, the parentheses can be omitted in the above case.

```
"walks?";
```

However, in the query

```
"walk(ed)?";
```

which retrieves the occurrences of "`walk`" and those of "`walked`", the parentheses cannot be omitted!

3.2.7 Iteration (Kleene star and Kleene plus)

A word like "`walk`" has several morphological variants: "`walks`", "`walked`", and "`walking`". Being sloppy, we query for all word forms which start with the character sequence `walk`. This is expressed by

```
"walk.*";
```

The *Kleene star operator* $*$ means that the preceding regular expression, here the unspecified character, can occur any number of times, or needn't occur at all. Since this is a sloppy way to express our intended query, we get also matches like "`walker`", "`walkie-talkie`" etc.

In the last query, the word form "`walk`" itself was a part of the query result. If you only want to see word forms which are strictly longer as `walk` itself, you have to use the *plus operator* `+` instead of the star $*$.

```
"walk.+";
```

The plus + works like the star $*$, but it requires that its argument expression occurs at least once.

It is a bit hard to think of natural language examples which match a regular expression where the Kleene operator takes a string of length 2 or longer as its argument. In the Penn Treebank corpus, the following query will match only the word ``Honolulu''.

```
".*lu(lu)+.*";
```

3.2.8 Flags

Some common types of regular expressions can be expressed in a much shorter manner with the help of CQP*flags*.

%d "insert diacritics"

It is sufficient to specify in the query the plain character without diacritics, but still all its occurrences with diacritics will be considered. E.g. our query for the word Spätzle will turn into:

```
"Spatzle" \%d;
```

%c "case insensitive"

Retrieve both upper and lower case variants of the query. Example:

```
"the" %c;
```

for searching ``the'' as well as ``The''.

%l "literal use"

With this option, all the CQP operators in the query are interpreted literally.

```
"+" %l;
```

finds all occurrences of + in the corpus. This query is equivalent to

```
"\+";
```

As the %l option turns off both %d and %c, only the combinations %l, %c, %d and %cd are useful.

3.2.9 A nontrivial example

We will conclude this section with quite a nontrivial example of a regular expression. Let's assume, we want to find occurrences of the German verb "`treffen`". Since the German language has a rich inflectional morphology, many word forms are based on this stem:

> "`treffen`", "`treffe`", "`triffst`", "`trifft`", "`trefft`", "`traf`", "`trafst`", "`trafen`", "`traft`", "`getroffen`", "`träfe`", "`träfst`", "`träfen`", "`träft`", "`treffend`", "`treff`", "`triff`"

The easiest way of rendering this list as a regular expression would be to write down a long disjunction of all the individual word forms. However, the query can become shorter (but maybe more opaque) based on the following observations.

- All word forms with the prefix `treff` are matched by the regular expression `treff(e(nd?)?|s?t)?`
- All word forms with the prefix `triff` are matched by the regular expression `triff(s?t)?`
- All word forms with the prefix `traf` are matched by the regular expression `traf(s?t|en)?`
- All word forms with the prefix `träf` are matched by the regular expression `tr\"af(en?|s?t)?`
- The word form `getroffen` is matched by the regular expression `getroffen`
- All word forms of the stem `treffen` may occur at the beginning of a sentence, so the initial character may be capitalized.

In total, we get the following regular expression

```
"[tT]reff(e(nd?)?|s?t)?|[tT]riff(s?t)?|[tT]raf(s?t|en)?|[tT]r\"af(en?|s?t)?|getroffen";
```

3.3 String variables

So far, we have seen only regular expressions as queries for strings. However, in corpus linguistics, people often deal with word lists and don't want to bother packing such a list into a regular expression. Therefore, `CQP` admits the definition and the use of *string variables*. A string variable takes a list of strings (or words) as its value. The name of string variable starts with the -symbol. For example, in order to define the string variable `$colors` as a list of the three strings `red`, `green`, and `blue`, you have to issue the followind `define`-command.

```
define $colors="red green blue";
```

Now, instead of writing a lengthy query like

```
"red|green|blue";
```

it is sufficient to say:

```
$colors;
```

Note that the string variable must not be included in quotation marks! Otherwise, it would be mistaken for a regular expression.

You can add further elements to the value of a string variable by the use of the `+=`-operator.

```
define $colors += "yellow";
```

Elements can be removed with the help of the `-=`-operator.

```
define $colors -= "green";
```

After so many manipulations of this variable, we may want to check its current value:

```
show $colors;

        red blue yellow
```

String variables are particularly useful, if the lists of tokens get longer. For this reason, the value of a string variable can be read from a file. The individual tokens must be listed in a *one-word per line format*. This means that the contents, say, of the file `mydict` could look as follows.

```
green
red
blue
yellow
```

The string variable `$colors` is set to the disjunction of these words by the following variant of the `define`-command.

```
define $verbs < "mydict";
```

Note that string variables can only stand for lists of tokens, not for regular expressions over strings.

3.4 Attribute Expressions

3.4.1 Attributes

The examples in the last section referred to the words of a text. However, CQP supports the work with annotated corpora. The *corpus positions* of a corpus can be annotated with an unlimited number of *attributes*. Usually, each word defines a corpus position, whereas, for a speech corpus, corpus positions could be defined at the phoneme level. For CQP, the word form (or the phoneme) which is associated with a corpus position, is just one, albeit *distinguished* kind of *positional attribute*. This means that the query

```
"Clinton";
```

is just an abbreviation of its more formal equivalent:

```
[word = "Clinton"];
```

The square brackets `[`, `]` mark the beginning and the end of a query for a single corpus position. The =-symbol marks an *attribute-value pair*. It is a two-place operator which takes an *attribute name* (e.g. `word`) on its left side and an *attribute value* on its right side. Attribute values have to be string variables or regular expressions over character strings.

The Penn Treebank corpus is an example of an annotated corpus (or tagged corpus.) Its corpus positions come with part-of-speech *tags* ('`pos` tags'). In this tag set, the part-of-speech tag for determiners is `DT`. Hence, if we want to find all determiners in the `UP` corpus, we have to write the following query.

```
[pos = "DT"];
```

Here is an example of a (nontrivial) regular expression for the `pos`-attribute.

```
[pos = "N.*"];
```

This query matches all corpus positions whose `pos`-value is in the set of 'noun tags' `NN`, `NNS`, `NP`, `NPS`. In order to see which one of the matching words comes with which `pos`-value, you can switch on the display for this attribute (see section 2.4.6) and have the result of the previous query displayed again.

```
show +pos;
cat;

 2: .TXT/.TXT .PP/.PP  <Pierre/NP> Vinken/N ,/, 61/CD year
 3: T/.TXT .PP/.PP Pierre/NP  <Vinken/NP> ,/, 61/CD years/NNS old/
 6: e/NP Vinken/NP ,/, 61/CD  <years/NNS> old/JJ ,/, will/MD join/
12: , will/MD join/VB the/DT  <board/NN> as/IN a/DT nonexecutive/
```

By the way, the *distinguished attribute* can be changed. For example, if your corpus is annotated with a `lemma` attribute and you are interested mainly in that attribute, you can change the value of the `CQP` variable `DefaultNonbrackAttr` in the following manner:

```
set DefaultNonbrackAttr lemma;
```

Then the query

```
"see";
```

will mean

```
[lemma = "see" ];
```

3.4.2 Boolean expressions over attribute-value pairs

Attribute-value pairs can be combined into Boolean expressions. CQP admits the usual logical operators: conjunction, disjunction, and negation.

Embedded Boolean expressions

As usual, *parentheses* `(`, `)` are used to mark an expression which is embedded into another Boolean expression. The parentheses help to specify the range and the precedence of the logical operators.

Conjunction

It is convenient to combine constraints on several attributes into a single query. For example, if we want to see only the uses of the word "rain" as a noun (in the UP-corpus), we need the *conjunction operator* `&`.

```
[word="rain" & pos="NN"];
```

This attribute expression calculates the *intersection* of the results of the two queries

```
[word="rain"];
```

and

```
[pos="NN"];
```

Disjunction

An example for the use of the *disjunction operator* `|` in an attribute expression would be

```
[ word="the" | word="The" ];
```

This query results in the set *union* of the results of the two queries

```
[ word="the" ];
```

and

```
[ word="The" ];
```

However, in the earlier sections of this chapter, we have seen more succinct ways to express the disjunctive query above. For example

```
"[tT]he";
```

Negation

When you want to judge the quality of an automatic part-of-speech tagger, you are looking for words which have been tagged in the wrong way. Astonishingly, although the word ‘‘the’’ is only used as a determiner ('DT') in English, it occurs also with some other part-of-speech tags in the Penn Treebank corpus. So let's search for those occurrences.

```
[word = "[tT]he" & !( pos = "DT" ) ];
```

The -operator takes a Boolean expression on its right side. In the case of the given query, logically, CQP calculates the intersection of the query result for the word ‘‘the’’ (resp. ‘‘The’’) and the result of the query for all corpus positions which are labelled with a part-of-speech tag different from DT. Here are the first few lines of the result.

```
 5909: entennial/NN year/NN ,/,  <The/NP> Wall/NP Street/NP Journa
 26347: , billed/VBD as/IN ‘‘/‘‘  <The/NP> Practical/NP Journal/NP
50124:  6/CD editorial/NN ‘‘/‘‘  <The/NP> Ill/NP Homeless/NP ’’/’’
80051: P Jones/NP publishes/VBZ  <The/NP> Wall/NP Street/NP Journa
259798: P Atwood/NP ’s/POS ‘‘/‘‘  <The/NP> Handmaid/NP ’s/POS Tale/
262883: es/NP ’’/’’ and/CC ‘‘/‘‘  <The/NP> Mary/NP Tyler/NP Moore/N
```

In the case of negated attribute-value pairs such as `!(pos = "DT")`, CQP offers an abbreviation, based on the -operator, the so-called *value negation*. The previous query can be equivalently written as

```
[word = "[tT]he" & pos != "DT"];
```

Negations of complex expressions are evaluated according to the well-known Boolean equivalences. This means that the three following queries are equivalent.

```
[ !(word = "water" | pos = "NN")];

[ !(word = "water") & !( pos = "NN")];

[ word != "water" & pos != "NN" ];
```

Operator precedence

The precedence properties of the (logical) operators are defined by the following list, i.e. if operator x is listed before operator y, operator x has precedence over y.

```
=, !=, !, &, |
```

For example,

```
[ ! word = "water" & ! pos = "NN" ];
```

disambiguates as

```
[ !(word = "water") & !( pos = "NN")];
```

Operators are evaluated from left to right.

x

Chapter 4

Access to sequences of corpus positions

This chapter describes how one can retrieve sequences of corpus positions. In chapter 3, we have learned about the retrieval of single corpus positions. The language for retrieving single corpus positions consists of attribute expressions, i.e. Boolean expressions over regular expressions over characters. Subsequently, *attribute expression* will often be called *descriptions of corpus positions*, or simply, *corpus positions*. Now, we scale up once more, by introducing Boolean expressions over regular expressions over attribute expressions. We will call them *(sequence) patterns*.

4.1 Regular expressions over attribute expressions

The language of regular expressions over attribute expressions includes the following operators: parentheses for marking embedded expressions, concatenation, disjunction, unspecified corpus position, optionality, Kleene star, and Kleene plus.

4.1.1 Embedded regular expressions

As usual, the parentheses `(`, `)` will mark embedded expressions.

4.1.2 Concatenation

Regular expressions over corpus positions are concatenated by simple juxtaposition. So, for example, the search for the string ``Bill'' followed by the string ``Clinton'' is triggered as follows

```
"Bill""Clinton";
```

```
653114: ence on education , Gov. <Bill Clinton> of Arkansas announced th
```

Since the query refers to sequences of length 2, the match is also of length 2, i.e. there are two word forms beetween the angle brackets.

For the concatenation of corpus positions, blanks and even new lines don't count. This means that you can format your query according to your taste, e.g.

```
"Bill"    "Clinton";
```

or

```
UP> "Bill"
"Clinton";
```

Of course, the description of an individual corpus position can be any kind of attribute expression. E.g. if you want to find the proper name (NP) which follows the Christian name "Bill" in the UP corpus, type:

```
"Bill" [pos = "NP"];
```

Concatenated corpus positions can be concatenated with further corpus positions, and so on. E.g. a query for multiword proper names, which consist of three individual proper names is represented by

```
[pos = "NP"] [pos = "NP"] [pos = "NP"];
```

Among others, it returns the following matches.

```
  177: BP in/IN today/NN 's/POS  <New/NP England/NP Journal/NP> of/IN Medicine/NP ,/,
  270: sk/NN ,/, ''/'' said/VBD  <James/NP A./NP Talcott/NP> of/IN Boston/NP 's/POS D
  276: P of/IN Boston/NP 's/POS  <Dana-Farber/NP Cancer/NP Institute/NP> ./SENT Dr./NP
```

4.1.3 Disjunction

Certain multiword entity names have the following pattern: two individual proper names, possibly with the preposition "of" in between. With the help of the *disjunction operator* , this query is simply formulated by enumerating the two different shapes:

```
([pos = "NP"] [pos = "NP"]) | ([pos = "NP"] "of" [pos = "NP"]);
```

Evaluation strategy

Whereas CQP enumerates all possible matches for a disjunctive regular expression over characters, CQP employs a (left-to-right) *first match strategy* for disjunctive regular expressions over corpus positions. This means the following: If the prefix of one of the disjuncts equals another disjunct, the longer disjunct will be ignored. I.e. in the query result, there will be no matches for the longer disjunct. - This applies only to the whole query, not to its embedded disjunctive expressions. For example, in the following query, the disjunct which encodes a sequence of three proper names (NP), is useless, since one of its prefixes is the other disjunct which asks only for two proper names. Hence, practically, albeit not logically, the query

```
([pos = "NP"] [pos = "NP"]) | ([pos = "NP"] [pos = "NP"] [pos = "NP"]);
```

is equivalent to the query

```
[pos = "NP"] [pos = "NP"];
```

This kind of incompleteness which is caused by the first match strategy applies to all the operations which will be introduced subsequently!

4.1.4 Unspecified corpus position

CQP also admits completely *unspecified* descriptions of *corpus positions*.

```
[];
```

This expression matches any position in the corpus.

4.1.5 Optionality

The *optionality operator* `?` makes the preceding expression optional. In this way, the above mentioned pattern of two individual proper names, possibly with the preposition ‘‘of’’ in between, can be formulated in a more succinct manner.

```
[pos = "NP"] "of"? [pos = "NP"];
```

4.1.6 Iteration (Kleene star and Kleene plus)

Whereas `.*` and `.+` are the *wild cards* or *iteration operators* on the level of character strings, on the level of corpus positions, the corresponding wild cards are represented by `[]*` and `[]+`, respectively. A naive way to search for coocurrences of the verb ‘‘give’’ and its particle ‘‘up’’ would be the following query.

```
"give" []* "up";
```

Since `[]*` matches any sequence of any length, we get matches like

```
 51446: <give the Transportation Department up>
62796:  <give up>
101737: <give it up>
132633: <give the department ample power to block undesirable deals .
   .TXT .PP For years , a strict regimen governed the staff meetings
   at Nissan Motor Co. 's technical center in Tokyo 's
   western suburbs . .PP Employees wore identification badges
   listing not only their names but also their dates of hire .
   No one could voice an opinion until everybody with more seniority
   had spoken first , so younger employees -- often the most
   enthusiastic and innovative -- seldom spoke up>
```

4.1.7 Restricted iteration

The third, very long match of the last query does not correspond to our intentions, when we want to find meaningful coocurrences of the verb ‘‘give’’ and the particle ‘‘up’’. If sentence-final punctuation is available generally in the corpus, we could include a negated attribute expression into the query in order to avoid that sentence boundaries are covered by the wildcard:

```
"give" [pos != "SENT"]* "up";
```

However, in the case of typos in the query (e.g. `[pos!="SNET"]*`), an iteration expression may match very large sequences, e.g. the whole corpus - which may cause a disaster. For that reason, `CQP` offers various ways to represent *restricted iterations*: multipliers, the `within` operator, and the `HardBoundary` variable. Remember also that an iteration at the end of a pattern is restricted by the first-match strategy (see section 4.1.3). As an abbreviation for iterations at the beginning or the end of a pattern, `CQP` provides the `expand` operator.

Multipliers

The *multiplier* operator takes a sequence pattern X on its left side as its argument and comes in three shapes.

***X*{*n*}**
: means exactly n repetitions of the expression X

***X*{*n*,}**
: means n or more repetitions of X

***X*{*n*,*m*}**
: means at least n and at most m repetitions of X

Now, we can reformulate the last query in a more reliable manner. E.g. we can state that the corpus interval between the words "`give`" and "`up`" should not be longer than five corpus positions.

```
"give" []{0,5} "up";
```

By the way, the operators `?`, `*`, and `+` are short forms for certain uses of the multiplier operator.

***X*?** is equivalent to X`{0,1}`

X* is equivalent to X`{0,}`

***X*+** is equivalent to X`{1,}`

Local upper bound of match size

The maximal length of matches for a query can be restricted by using the `within` operator. It takes a sequence pattern as its left argument and a natural number (or more exactly: a *distance expression*) on its right side. For example,

```
"give" []* "up" within 7;
```

which is equivalent to our earlier query

```
"give" []{0,5} "up";
```

since the maximal number of consecutive corpus positions which can be covered is 8.

By the way, the following queries are notational variants of each other:

```
"give" []* "up" within 7;
"give" []* "up" within 7 word;
"give" []* "up" within 7 words;
```

Note that the `cut` operator (see section 2.4.4) must not precede the `within` operator. A correct example would be:

```
"give" []* "up" within 7 cut 5;
```

Here are a few more pairs of equivalent queries:

```
[]* "Clinton" within 3;
[]{0,3} "Clinton";
894370: <Clinton>
1681560: <currently owned by Clinton>
1681561: <owned by Clinton>
1681562: <by Clinton>

"Clinton" []{0,2};
"Clinton" []* within 2;
```

The latter pair of queries will only return matches with the single word "`Clinton`" due to the first-match strategy.

When the `within` operator applies to an expression with several iterations, an equivalent, plain sequence pattern is quite hard to formulate. For example, the following query restricts the whole match to a maximum size of 11 corpus positions.

```
"man" []* "is" []* [pos="VBN"] within 10;
```

In a naive, non-optimized formulation, this corresponds to a large disjunction of all the possible partitions of 11 corpus positions.

```
( ( "man"         "is" []{0,8} [pos="VBN"] )
| ( "man" []{0,1} "is" []{0,7} [pos="VBN"] )
| ( "man" []{0,2} "is" []{0,6} [pos="VBN"] )
...
| ( "man" []{0,7} "is" []{0,1} [pos="VBN"] )
| ( "man" []{0,8} "is"         [pos="VBN"] ) )
```

Fortunately, `CQP` does a certain amount of optimization for us, since otherwise the following match would occur several times in the result, since it is a match of each disjunct.

```
1452974: <man/NN ,/, is/VBZ attributed/VBN>
```

Global upper bound of match size

For people who don't want to bother with figuring out the right kind of local restrictions, `CQP` comes with a default maximum size for matches, the so-called `HardBoundary` variable. You can check the value by stating

```
set HardBoundary;
```

or change it, e.g. by

```
set HardBoundary 20;
```

This will restrict the matches of subsequent queries to the maximal length of 21 corpus positions - unless a query makes use of the `within` operator. A `within` restriction makes `CQP` ignore the value of the `HardBoundary`.

Expansion of matches

Sometimes it is convenient to include a certain amount of the context into the match itself. If you want to include three corpus positions to both sides of the matches of the ``Clinton'' query, you would write:

```
[]{5} "Clinton" []{5};
```

But it may be more convenient to use the following, semantically equivalent syntax:

```
"Clinton" expand to 5;
```

Readability can be somewhat increased by using the keywords `word` or `words`:

```
"Clinton" expand to 5 word;
"Clinton" expand to 5 words;
```

If you only want to include left context into the match, the statement reads:

```
"Clinton" expand left to 5;
```

Similarly for 'expansion' on the right side:

```
"Clinton" expand right to 5;
```

Note, that, in the `CQP` syntax, the `expand` operator must neither precede the `cut` operator (see section 2.4.4) nor the `within` operator (see section 4.1.7).

4.2 Sequence patterns

The full language of *sequence patterns* allows for Boolean expressions over regular expressions over corpus positions. However, in the tradition of automata theory, `CQP` does not use the proper syntax of Boolean operators for that purpose. The Boolean operations of conjunction, disjunction, and negation, are rendered by their set-theoretical counterparts: intersection, union, and set difference.

4.2.1 Named queries

Regular expressions over corpus positions cannot be directly embedded into Boolean expressions, since the Boolean operators (or set operators) in sequence patterns take only *query names* as their arguments. Query names are similar to variables in programming languages. A query name is created by an *assignment statement*. `CQP` will assign automatically the (result of the) most recent query to the query name `Last`. If you want to refer to a query later on, you can assign the value of `Last` to some other query name, say `MyQuery`, e.g.

```
"Clinton";
MyQuery = Last;
```

The above two statements can be packed into a single one. The following `CQP` statement will run the query `"Clinton"` and assign its result to the query name `MyQuery` (and to `Last` !).

```
MyQuery = "Clinton";
```

The righthand side of an assignment statement may also include an `expand` operator whose left argument is a query name:

```
"Clinton";
Q1 = Last expand to s;
```

Remarks

- The assignment statement suppresses the output of the query result, although the `AutoShow` variable might have been switched on (see sections 2.4.1 and 2.4.2). Therefore, if you want to see the result of `MyQuery`, you have to say

  ```
  cat MyQuery;
  ```

- Query names must be distinct from attribute names, names of system corpora, `CQP` variables, and `CQP` commands. The names of system corpora are listed by the `show` command (see section 2.2). The reserved `CQP` symbols have been included in the index of this manual.
- Like in programming languages, an assignment statement will overwrite any previous value of the query name on the lefthand side of the `=` sign. If you have lost track of the query names which have been already introduced in the current `CQP` session or have been saved in an earlier session (see appendix C.1), you can use the `show` command (see section 2.2), or more precisely,

```
show sub;
```

to have the existing query names listed.

- Assignment does **not** mean identification. E.g. the `reduce` operation (see section 2.4.4) below will only affect the result of its argument `MyQuery` but not the result of `Last`, from which `MyQuery` inherited its value.

```
"Clinton";
MyQuery = Last;
reduce MyQuery to 5;
cat MyQuery;
cat Last;
```

4.2.2 Conjunction

The conjunction operator `intersect` (or `inter`) is a prefix operator which takes two query names on its right side as its arguments. For example:

```
UP> Q1 = "rain";
UP> Q2 = [pos="NN"];
intersect Q1 Q2;
```

The above sequence of `CQP` statements is equivalent to the query in section 3.4.2:

```
UP> [word = "rain" & pos = "NN"];
```

Here is a more complicated example:

```
UP> Q1 = [pos = "JJ"] [pos = "NN"];
UP> Q2 = "acid"       "rain";
UP> intersect Q1 Q2;
```

which amounts to

```
UP> [word = "acid" & pos = "JJ"] [word = "rain" & pos = "NN"];
```

Note that the use of the intersection operator makes only sense if the queries denote the same type of sequences. E.g. the result of the intersection below is empty, since it asks for corpus positions which are simultaneously labeled as adjectives and as nouns - a condition which obviously never holds.

```
UP> Q1 = [pos = "JJ"];
UP> Q2 = [pos = "NN"];
UP> intersect Q1 Q2;
```

Similarly, the following query results in an empty set, since it looks for corpus intervals which are simultaneously of length 1 and of length 2.

```
UP> Q1 = [pos = "NP"];
UP> Q2 = [pos = "NP"] [pos = "NP"];
UP> intersect Q1 Q2;
```

Even more importantly, both arguments of the `intersect` operator (and the operators `join` and `diff` below) must have been evaluated on the same system corpus, i.e. the following statements will lead to the CQP warning "`Original corpora of Q1 (UP) and Q2 (WSJ) differ.`"

```
UP> Q1 = "rain";
UP> WSJ;
WSJ> Q2 = "acid" "rain";
intersect Q1 Q2;
```

4.2.3 Disjunction

The disjunction operator `join` takes two query names as its arguments and creates the union of the results of both queries, for example:

```
Q1 = "acid" "rain" ;
Q2 = "brown" "coal" ;
join Q1 Q2;
```

Since regular expressions include a disjunction operation, the above query is logically equivalent to

```
( "acid" "rain" ) | ( "brown" "coal" );
```

Practically, there may be differences between the operators `|` and `join`, due to the first match strategy. Compare

```
Q1 = [pos = "JJ"] "rain" ;
Q2 = [pos = "JJ"] "rain" "forest" ;
join Q1 Q2;
```

with

```
( [pos = "JJ"] "rain" ) | ( [pos = "JJ"] "rain" "forest" );
```

The result of the `join` expression will include the match `<muddy rain forest>` whereas the result of the `|` expression will consist only of matches of length two.

4.2.4 Negation

The `difference` (or `diff`) operator takes also two arguments `Q1` and `Q2`, but this time the order of arguments counts! The result of the second argument `Q2` will be subtracted from the result of the first one `Q1`. Logically, this means `(Q1 & not Q2)`. For example, if you want to find out which are the less common prepositions in the `UP` corpus, you could write:

```
Q1 = [pos = "IN"] ;
Q2 = "as|at|by|for|from|of|in|into|on|up|with" ;
diff Q1 Q2;
```

The above query can be rephrased in an equivalent manner by using the (value) negation of attribute expressions:

```
[ pos = "IN"  & word != "as|at|by|for|from|of|in|into|on|up|with" ] ;
```

Here is an example where negation cannot be pushed down to the attribute expressions:

```
Q1 = [pos = "NP"] [pos = "IN"]? [pos = "NP"] ;
Q2 = [pos = "NP"] [pos = "NP"] ;
diff Q1 Q2;
```

The above query is just a complicated way to say:

```
[pos = "NP"] [pos = "IN"] [pos = "NP"] ;
```

Note that if the order of arguments is inversed, the result will be quite different, i.e. in this case it will be empty, since the result of `Q2` is included in the result of `Q1`.

```
Q1 = [pos = "NP"] [pos = "IN"]? [pos = "NP"] ;
Q2 = [pos = "NP"] [pos = "NP"] ;
diff Q2 Q1;
```

4.2.5 Embedded Boolean expressions

Since the operators `intersect`, `join`, and `diff` take only query names as their arguments, the proper embedding of other expressions is ruled out. However, fortunately, a Boolean expression may occur on the right side of an *assignment statement*. So, simply, assign the embedded expression to a (new) query name, and use that name instead of the expression itself. For example, the following statements describe the set of all adjective-noun sequences without the two sequences "`acid rain`" and "`brown coal`".

```
Q0 = [pos = "JJ"] [pos = "NN"] ;
Q1 = "acid" "rain" ;
Q2 = "brown" "coal" ;
Q3 = join Q1 Q2;
diff Q0 Q3;
```

Chapter 5

Access to structural information

CQP supports the access to two kinds of structural information with respect to corpora: *predefined structures* (structural annotations) and *ad hoc structures* (results of earlier queries).

5.1 Predefined structures

As we have seen in section 4.1.6, the unrestricted use of the wild card operator for corpus sequences does not make much sense, since it produces unintuitive results:

```
"give" []* "up";
```

```
...
132633: <give the department ample power to block undesirable deals .
   .TXT .PP For years , a strict regimen governed the staff meetings
   at Nissan Motor Co. 's technical center in Tokyo 's
   western suburbs . .PP Employees wore identification badges
   listing not only their names but also their dates of hire .
   No one could voice an opinion until everybody with more seniority
   had spoken first , so younger employees -- often the most
   enthusiastic and innovative -- seldom spoke up>
```

In section 4.1.7, a number of ways have been shown how the span of a match can be restricted. In the case that a corpus has been structurally annotated, the restrictions can refer to the these annotations. (The kind of annotations which a corpus provides can be checked by the `info` command, see section 2.2 and by the `show` command, see section 2.4.6.)

5.1.1 Use of structural tags in regular expressions

Since the Penn Treebank corpus is annotated with sentence boundaries (structural tag `s`), we can check, for example, which conjunctions (`CC`) occur at the beginning of a sentence:

```
<s> [pos="CC"];
```

Or, we can find out what kind of punctuation is used at the end of sentences:

```
[pos="SENT"] </s>;
```

The reference to structural tags may occur anywhere in a query. For example, we may look for nouns and determiners, separated by a punctuation mark and a sentence boundary:

```
[pos="N.*"] [pos="SENT"] </s> <s> [pos="DT"];
```

or shorter:

```
[pos="N.*"] [pos="SENT"] <s> [pos="DT"];
```

Now, we can try also to restrict the cooccurrences of the words "give" and "up" to single sentences:

```
<s> []* "give" []* "up" []* </s>;
```

However, this delivers still plenty of unwanted matches, where one of the wild cards `[]*` matches another `<s>` boundary:

```
132582:  <<s>.PP ‘‘ This ought to be subtitled the ‘ Don’t let Frank Lorenzo
take over another airline ’ amendment , ’’ said Rep. James Oberstar
( D. , Minn. ) , chairman of the House aviation subcommittee , who
argued that the provision was unnecessary because the bill already
would give the department ample power to block undesirable deals .</s>
<s>.TXT .PP For years , a strict regimen governed the staff meetings
at Nissan Motor Co. ’s technical center in Tokyo ’s western suburbs .</s>
<s>.PP Employees wore identification badges listing not only their
names but also their dates of hire .</s> <s>No one could voice an
opinion until everybody with more seniority had spoken first , so
younger employees -- often the most enthusiastic and innovative --
seldom spoke up at all .</s>>
```

5.1.2 Structural restrictions for matches

Again, the `within` operator (see section 4.1.7) helps us to express exactly what we want, i.e. the restriction of cooccurrences of "give" and "up" to single sentences.

```
"give" []* "up" within s;
```

The argument after the `within` operator can also be a natural number followed by the name of a structural attribute. In this way, more general cooccurrence patterns can be investigated. For example, the following query looks for occurrences of the words "gain" and "profit" in an interval of three sentences:

```
("gain" []* "profit") | ("profit" []* "gain")  within 3 s;
```

In the case that article boundaries are annotated like in the Wall Street Journal (WSJ), the above type of query can be made more meaningful:

```
WSJ> ("gain" []* "profit") | ("profit" []* "gain")  within article;
```

5.1.3 Expansion of matches

Matches cannot only be expanded by a fixed number of corpus positions from their contexts (see section 4.1.7) but they can also be expanded up to structural boundaries. For example, the following query will return all the sentences (tag `s`) where the word "`Clinton`" occurs.

```
"Clinton" expand to s;
```

Alternatively, only left or right context can be incorporated:

```
"Clinton" expand left to s;
"Clinton" expand right to s;
```

Matches can also be expanded to cover several consecutive structures:

```
"Clinton" expand left to 2 s;
```

expands the match to the second sentence boundary on the left side. The query

```
"Clinton" expand to 2 s;
```

expands the match to the current sentence and adds one sentence on each side, so that the whole match then spans three sentences.

Note that the number of matches for the query `"Clinton"` may differ from the number of matches for `"Clinton" expand to s;`. In the case that the word "`Clinton`" occurs more than once in the same sentence, this sentence will occur only once in the 'expanded result', whereas the result of the original query contains all the individual occurrences of the word "`Clinton`". On the other hand, if the match of the initial query `"Clinton"` is not part of a sentence at all (but e.g. a part of a headline), it will still be part of the 'expanded result'.

The match expansion to a structural boundary allows for the simulation of queries in the style of internet search engines. An internet-style query for documents which contain both the words "`gain`" and "`profit`" (`"gain" AND "profit"`) does not refer to the relative order of the individual keywords, whereas the regular expressions in `CQP` do make reference to 'word order'. For that reason, the query `"gain" AND "profit"` corresponds to a rather complicated regular expression in `CQP`:

```
WSJ> ("gain" []* "profit") | ("profit" []* "gain")
     within article expand to article;
```

If we position the `expand` operator in a more 'intelligent' manner, the above query can be rephrased as

```
WSJ> Q1 = "gain" expand to article;
WSJ> Q2 = "profit" expand to article;
WSJ> intersect Q1 Q2;
```

This means that we ask for all articles `Q1` which contain the word "`gain`", all articles `Q2` which contain the word "`profit`", and obtain those articles which contain both words by a simple intersection of `Q1` and `Q2`.

5.2 Ad hoc structures

In addition to the predefined structural annotation of a corpus, temporarily defined structural boundaries (*ad hoc structures*) can be exploited in queries.

5.2.1 Structural restrictions for matches

The class of attributive adjectives is defined by the syntactic context in which the adjective occurs, e.g. the sequence determiner (`DT`), adjective (`JJ`), and noun (`NN`).

```
[pos = "DT"] [pos = "JJ"] [pos = "NN"];
```

If we want to look only at the adjectives in the given context, we can express that in `CQP` as follows:

```
UP> NounPhrase = [pos = "DT"] [pos = "JJ"] [pos = "NN"];
UP> NounPhrase;
UP:NounPhrase[25913]> [pos = "JJ"];
```

This yields:

```
...
      77:  <high>
     204:  <old>
     259:  <useful>
     331:  <different>
     426:  <striking>
     526:  <common>
     627:  <gradual>
     696:  <huge>
     870:  <current>
     948:  <average>
...
```

The command sequence

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

tells `CQP` that the query `[pos = "JJ"]` has to find its matches *within* the boundaries defined by the query `NounPhrase`.

In order to get rid of the structural restriction, you have to select again the system corpus (see section 2.2), e.g.

```
UP:NounPhrase[25913]> UP;
UP>
```

Note that the `expand` operator and the insertion of an ad hoc structural boundary can be combined:

```
UP>          "Clinton";
UP>          Last expand to 5;
UP:Last[12]>
```

This is equivalent to

```
UP>         "Clinton" expand to 5;
UP>         Last;
UP:Last[12]>
```

Chapter 6

Access to multilevel annotation

The only kind of multilevel annotation which is supported by CQP is the *alignment* of two corpora, e.g. a text and its translation or a text and its phonetic transliteration.

6.1 Alignment constraints

CQP allows for queries which put simultaneously a constraint on the matches from a *source corpus* and on the corresponding corpus intervals from the *target corpus*, the corpus which is aligned with the source corpus. The *alignment constraint*, the constraint on the intervals from the target corpus, can only be global to query to the source corpus, i.e. alignment constraints cannot be embedded somewhere into queries.

The following examples are taken from the Hansard corpora (Canadian parliament protocols) which are available in French (HANSARD-F) and in English (HANSARD-F). Remember that in order to see the English counterparts, the alignment display has to be switched on (see section 2.4.6) by `show +hansard-e;`

```
HANSARD-F> "neuf" :HANSARD-E "new";
```

```
...
  1387939:  <neuf>
-->hansard-e: It is hard to renovate something that is already there ,
  but if it is new buildings going up , it might be a good place to start .
  1414074:  <neuf>
-->hansard-e: From what I have heard , that is a slightly new angle .
  1557034:  <neuf>
-->hansard-e: That is a new wrinkle too that we have not heard in
   now nine provinces and two territories .
...
```

The result of the above query is a list of pairs:

1. a match from the source corpus for the query `"neuf"`
2. the corpus interval (e.g. a sentence) from the target corpus

(a) which is aligned to that interval in the source corpus where the match for `"neuf"` occurs

(b) and which contains a match for the alignment constraint

In contrast to usual `CQP` patterns, alignment constraints can be negated by the operator . This operator constrains all the corpus positions in the aligned interval which is selected by the constraint on the source corpus. The following query searches for occurrences of ``neuf'' where there is **no** occurrence of ``new'' in the aligned corpus interval:

```
"neuf" :HANSARD-E ! "new";
```

The negation operator may occur only directly after the name of the aligned corpus.

The source corpus constraints and the alignment constraints may make use of the whole expressiveness of the `CQP` query language.

Remarks

An alignment constraint (whether negated or not) evaluates to `false` if a match from the source corpus has no aligned counterpart in the target corpus, e.g. in the case that there are 'holes' in the alignment.

The `cut` command considers only the matches for the constraint on the source corpus. So, when running queries on aligned corpora, a low bound for the `cut` operator may incidentally cut off all those source corpus matches with satisfiable alignment constraints. I.e. the query result may be empty.

Chapter 7

Inspection of query results

For the inspection by the human user, `CQP` offers alphabetical and frequency-based sorting of query results. In order to make the sorting routines more flexible, `CQP` allows for the marking of interesting corpus positions in a match or related to a match.

7.1 Marked corpus positions

By default, the sorting algorithms of `CQP` consider only the first position of a match or an interval of corpus positions at a fixed set-off from the beginning of a match. However, when a query includes iteration operators (see sections 4.1.6 and 4.1.7) there is no such fixed set-off, in general. In `CQP`, the `set collocate` command and its identical twin, the `set keyword` command, have to be used to mark 'interesting' corpus positions whose distance from the beginning of a match may vary. A `set collocate` command corresponds to a restricted kind of regular expressions **plus** an individual search strategy. For example, the following statements will create a query result for the regular expression `"give" []{0,5} [pos="RP"]` where the particles (`RP`) are marked.

```
"give" []{0,5} [pos="RP"];
set Last collocate rightmost [pos="RP"] within right 6 words from match inclusive;
```

The command `set collocate` takes five arguments:

The *first argument* (`Last`) is a query name.

The *fourth argument* is a way to encode a multiplier expression. '`within right 6 words`' may mean `[]{0,5}` between the first position of the match and the marked corpus position on the `right` side of the match. If the direction, `left` or `right`, is omitted, both sides of the match will be searched. If the number (`6`) is omitted, the default distance is `1`. In the place of `words`, you can use `word` or any available structural annotation.

If the *fifth argument* has the value `from match inclusive`, the fourth argument is evaluated starting from the first position of the match. The value `from match exclusive` means that corpus positions are counted starting right after the match. The other possible values of the fifth argument are `from collocate exclusive`, `from collocate inclusive`, `from keyword exclusive`, and `from keyword inclusive`. The fifth argument is optional. When omitted, the default is `from match exclusive`.

The *third argument* of `set collocate` (`[pos="RP"]`) is an attribute expression which restricts the 'interesting' corpus position. Note that one cannot use a general pattern (for a sequence of corpus positions) at this place!

The *second argument* (`rightmost`) indicates the *search strategy*. Alternative values are `leftmost`, `nearest`, and `farthest`. The search strategy helps to disambiguate when there are several matches around the same corpus position, e.g

```
"to" []{0,5} [pos = "DT"];
...
 2008: tempts them <to return to a> heartland city for
 2010: tempts them to return <to a> heartland city for
...
```

Remarks

- The `set collocate` operator suppresses the output of the query result although the `AutoShow` variable might have been switched on. Use the `cat` operator (see section 2.4.2) to have the result displayed.
- In contrast to the real multiplier operator, the `set collocate` command does **not** exclude those matches **without** a fitting 'collocate', it just marks the 'collocates' in those matches where they occur. If the initial match is too unrestricted, the `set collocate` will **not** make it more restricted! For example, the following statements will produce all matches for the word **"`give`"** where those particles are marked which are maximally five positions right of "`give`".

  ```
  "give";
  set Last collocate leftmost [pos="RP"] within right 5 words;
  ```

- In the following example, the default query name `Last` will still be associated with the value of `NounPhrase` **before** `set collocate` and `set keyword` were carried out. I.e. in `Last` there won't be anything marked even after all the statements below have been processed.

  ```
  NounPhrase = [pos = "DT"] [pos = "JJ"] [pos = "NN"];
  set NounPhrase collocate leftmost [pos = "JJ"] within right 1 words
      from match inclusive;
  ```

- Note the implications of the fact that the keyword '`match`' in '`from match inclusive`' means actually "the first position of the match". This may lead to confusion when searching on the `left` side of the match. E.g. in order to have marked occurrences of the word **"a"** in the expression `"to" []{0,5} [pos = "DT"]` we might be tempted to say:

  ```
  "to" []{0,5} [pos = "DT"];
  set Last collocate rightmost "a" within left 5 words from match inclusive;
  ```

  However, this will mark the rightmost occurrence of **"a"** on the left side of the **second** position of the match.

- When both sides are searched, the `set collocate` command may still be too underspecified to disambiguate a single position to be marked. But never mind, CQP will simply take the first fitting position from the left. E.g.

```
"Clinton";
set Last collocate farthest [] within 2 words from match exclusive;
```

> *44963:* Utah , $ 15,000 *fine* ; **Clinton** P. Hayne , New Orleans ,
> *653115:* on education , *Gov.* Bill **Clinton** of Arkansas announced ...
> *894320:* , already buys *gas* from **Clinton** . .PP Clinton said in
> *1681563:* 's , currently *owned* by **Clinton** Holding

- In the third statement below, the first occurrence of the keyword `collocate` refers to a corpus position **different** from the corpus position designated by the second occurrence of `collocate` in the same line!

```
"Bill" "Clinton";
set Last collocate leftmost [] within right 1 words from match exclusive;
set Last collocate leftmost [] within right 1 words from collocate exclusive;
```

7.1.1 Two marked positions

One can combine the `set collocate` and the `set keyword` command in order to mark two 'interesting' positions simultaneously. The following statements mark the adjectives (JJ) in `NounPhrase` as 'collocates' and the nouns as 'keywords'.

```
NounPhrase = [pos = "DT"] [pos = "JJ"] [pos = "NN"];
set NounPhrase collocate leftmost [pos = "JJ"] within right 1 words
    from match inclusive;
set NounPhrase keyword leftmost [pos = "NN"] within right 2 words
    from match inclusive;
cat NounPhrase;
```

76: <a **high** percentage>
203: <an **old** story>
258: <no **useful** information>
330: <a **different** type>
425: <a **striking** finding>
525: <the **common** kind>
626: <a **gradual** ban>
695: <a **huge** bin>
869: <the **current** yield>
947: <The **average** maturity>

7.2 Alphabetical sorting

(available in release 2.3)

The basic command to have a query result ordered alphabetically, is the statement `sort by 0;`. Try:

```
"Clint.*";
sort by 0;

   371332:    Lockman } and third {  <Clint> Hartung } as Bobby
    44963: , Utah , $ 15,000 fine ;  <Clinton> P. Hayne , New Orleans ,
   653115: on education , Gov. Bill  <Clinton> of Arkansas announced
...
  1826420:   Hayes National Bank in  <Clinton> . The bank holding
   726164:     Telephone Corp. , of  <Clintonville> , Wis. , the
```

The `sort` command takes maximally five arguments. E.g. the statement `sort by 0` reads in its full beauty:

```
sort Last by 0 relative to match on word ascending;
```

The *first argument* (`Last`) is a query name. If the query name is omitted, its default is `Last`.

The *fourth argument* (`on word`) is the positional attribute whose value will serve as the basis for sorting. If omitted, the default value is `on word`. Here is an example where the verbs which start with `be` (and whose base forms are not equal to ``be'') are sorted according to the lemma value of the match.

```
[word = "be.*" & pos = "VB" & lemma != "be"];
sort by 0 on lemma;
```

The *third argument* (`relative to match`) indicates the (marked) corpus position, which serves as the basis to define the corpus interval to be sorted. Other values are `relative to collocate` and `relative to keyword`. The default value is `relative to match`. Again, '`match`' means here "first position of the match".

The *second argument* (`by 0`) designates the position, or more generally the corpus interval, which will be sorted. This corpus interval is defined relative to the corpus position indicated by the third argument. Examples:

The statements

```
"give" []{0,5} [pos="RP"];
sort by 5;
```

will sort on the basis of the fifth position from the beginning of the match, even if that position is not part of the match. Since this is probably not what you want, you should rather include a `set collocate` statement to mark the particle (`RP`)..

```
"give" []{0,5} [pos="RP"];
set Last collocate leftmost [pos="RP"] within right 6 words
    from match inclusive;
sort by 0 relative to collocate;
```

The matches where no particle is found in that range will be usually added to the end of the result.

Longer corpus intervals are represented by comma-separated integers. E.g. the statements below will sort the query result by considering the complete matches of length 2.

```
"Bill" [pos = "NP"];
sort by 0,1;
```

Negative integers can be used as to abbreviate certain kinds of `set collocate` expressions:

```
"Inc\." ;
sort by -1;
```

abbreviates

```
"Inc\." ;
set Last collocate rightmost [] within left 1 word;
sort by 0 relative to collocate;
```

The *fifth argument* lets you choose between `ascending` or `descending` (i.e. inverse) alphabetical sorting. The parameters can be abbreviated as `desc` and `asc`, respectively. The default sorting order is `ascending`.

Remarks

- The `sort` command displays its result immediately unless the `AutoShow` variable has been switched off.
- The `sort` command is destructive on its first argument, the query name whose result is to be sorted. Subsequent `cat` statements will always display the sorted result.

7.3 Frequency-based sorting

The simplest command to have a query result sorted with respect to the frequencies of the different matches reads:

```
"Clint.*";
group Last match word;
```

In total, the `group` command may take four arguments. E.g. the following `group` statement is essentially equivalent to the simpler one above:

```
group Last match word foreach match word cut 0;
```

The *first argument* (`Last`) is a query name.

The *second argument* (`match word`) defines the attribute (`word`) of a corpus position (here the first position of the match) whose values will be counted. Other designators for corpus positions are `collocate` and `keyword` (see section 7.1).

The *third argument* gives the justification for the name of this command. The following statements generate for each particle a list of verbs with the respective cooccurrence frequencies.

```
[pos="VB.*" & pos!="VB(P|Z)"] []{0,5} [pos="RP"];
set Last collocate leftmost [pos="RP"] within right 6 words
    from match inclusive;
group Last match lemma foreach collocate word;
```

The keyword `foreach` is followed by a designator for a corpus position, i.e. `match`, `collocate`, or `keyword`, and a name of a positional attribute.Instead of the symbol `foreach`, one can also use '`by`'.

The *fourth argument*, which is optional, is the `cut`-operator. (The `cut` operator for `group` commands has been added in release 2.3.) It takes a natural number as its argument. This number determines the *cut-off* frequency for matches to be ignored.

Remarks

- The `group` command always outputs its result, regardless whether the `AutoShow` variable has been switched off or not. If the output of the `group` command to the standard output is annoying, it can be directed to a file or to a Unix pipe and be influenced by the `PrintMode` and `PrintOptions` similarly to the output of the `cat` command (see section 2.4.7, some effects are only part of release 2.3).
- The `group` command only shows the values and respective frequencies of the attributes of the corpus position or the two corpus positions under consideration. I.e. other details of the original matches are ignored and cannot recovered from the output of a `group` command.
- The `group` command always sorts by decreasing frequency.
- The `cut` and the `foreach`/`by` operators do not work together correctly.

Appendix A

The cqp and cqpcl commands

CQP can be invoked with the shell commands cqp (for interactive use) resp. cqpcl for non-interactive use. Upon startup, both commands read the values of certain Unix system variables (see section A.1) and evaluate the contents of the file ~/.cqprc. This file may contain any number of valid CQP statements. As an alternative to the CQP commands, certain CQP variables can be set by command line options (see section A.2). In addition, the command cqpcl reads an arbitrary of command line arguments, which will be evaluated as CQP queries unless they are marked as command line options. The result will be written to stdout.

A.1 Environment variables for CQP

CQP uses the values of the following UNIX environment variables.

CORPUS_REGISTRY

When CQP is started, the CQP variable Registry (see section 2.2) is set to the value of CORPUS_REGISTRY. Usually, CORPUS_REGISTRY is set by the system administrator.

CQP_LOCAL_CORP_DIR

This variable serves to initialize the CQP variable LocalCorpusDirectory (see section C.1).

PAGER

This is a generally used UNIX environment variable. Its value initializes the CQP variable Pager (see section 2.4.3).

A.2 Command line options

A.2.1 General options

Help

-h

The CQP help message will be displayed with short information about the usage of the command line options. This message will be also shown if CQP is called with incorrect parameter settings.

Batch mode

-f *File-Name*

CQP will read its input from the file *File-Name* instead of interacting with the user. In order to make your CQP 'programs' more readable you can add *comments* in the usual shell style, i.e. prefixed with the #-symbol.

Binary output

-i

With this option, CQP provides a simple protocoll for the communication with other applications. The query results which are represented as corpus position pairs are printed as binary coded integers.

Protected mode

-x

The user will not be able to redirect the output of the cat and group commands into Unix pipes. Recommended for WWW-applications of CQP.

A.2.2 Basic interaction

Corpus selection

-D *Corpus-Name*

The corpus named *Corpus-Name* will be selected when CQP is started. Instead of using the command line option -D, you can set the CQP variable DefaultCorpus in your ~/.cqpcr file, e.g. by the command

```
set DefaultCorpus "UP";
```

which will select the UP corpus upon start-up of CQP.

Browsing method

-P *BrowserName*

For e.g. *BrowserName*=more, this amounts to (see section 2.4.3)

```
set Pager more;
```

Context size

`-W` *Number*

`-L` *Number*

`-R` *Number*

E.g. for *Number*=`20`, these options equal the following `CQP` statements (see section 2.4.5), respectively:

```
set Context 20;
set LeftContext 20;
set RightContext 20;
```

A.2.3 Sequence patterns

Size of matches

`-b` *Number*

E.g. for *Number*=`50`, the use of this option is equivalent to setting the `HardBoundary` variable (see section 4.1.7) to `50`:

```
set HardBoundary 50;
```

Appendix B

Special characters

Country-specific characters do not belong to the 7-bit ASCII standard. However, it is possible to enter these characters by using the corresponding (country-specific) LaTeXcommands. For example, the word **``façade''** is entered as

```
"fa\,cade";
```

Ä	`\"A`	Á	`\'A`	À	`` \`A ``	Â	`\^A`
Ë	`\"E`	É	`\'E`	È	`` \`E ``	Ê	`\^E`
Ï	`\"I`	Í	`\'I`	Ì	`` \`I ``	Î	`\^I`
Ö	`\"O`	Ó	`\'O`	Ò	`` \`O ``	Ô	`\^O`
Ü	`\"U`	Ú	`\'U`	Ù	`` \`U ``	Û	`\^U`
ä	`\"a`	á	`\'a`	à	`` \`a ``	â	`\^a`
ë	`\"e`	é	`\'e`	è	`` \`e ``	ê	`\^e`
ï	`\"i`	í	`\'i`	ì	`` \`i ``	î	`\^i`
ö	`\"o`	ó	`\'o`	ò	`` \`o ``	ô	`\^o`
ü	`\"u`	ú	`\'u`	ù	`` \`u ``	û	`\^u`
Ç	`\,C`	ç	`\,c`				
ß	`\"s`						

Appendix C

Incremental corpus exploration

E.g. if you are investigating the cooccurrence patterns of adjectives (`JJ`) and nouns (`NN`), it makes sense to ask first a very general query like

```
Q1 = [pos = "JJ"] [pos = "NN"];
```

and then to refine `Q1` in various **alternative** manners. This will save computation time. Query names (like `Q1`) have been introduced in section 4.2.1. Sections 4.2 and 5.2 showed how to make use of query results in subsequent queries. In addition, this chapter tells you how query results can be saved to hard disk and deleted from working memory.

C.1 Saving of intermediate results

If your work extends over several `CQP` sessions, you can save a query result into a file in `CQP`-encoded format (which should not be edited manually). The command

```
UP> save;
```

will save the result of the most recent query into the file `UP:Last` in the 'local corpus directory' (and possibly overwrite an older version of `UP:Last`). Whereas

```
UP> save MyQuery;
```

will save the result of the query named `MyQuery` as `UP:MyQuery`.

In order to carry out the `save` command successfully, the `CQP` variable `LocalCorpusDirectory` must have been set to the path of a directory with write access, e.g.

```
set LocalCorpusDirectory "/corpora/subcorpora:~/subcorpora";
```

The `save` operation is illegal for system corpora.

C.2 Erasing of intermediate results

Query names produce overhead, since `CQP` has to manage the results which are associated with these names. Therefore, at times, it might make sense to remove query names from the `CQP` working memory. E.g. for the query names `Q1` and `Q2`, this is done by the command

```
discard Q1 Q2;
```

Note that this only deletes the information about `Q1` and `Q2` from the `CQP` working memory, but it does **not** delete any saved query results (see section C.1) from your hard disk.

Index

Bibliography

[1] Oliver Christ. A modular and flexible architecture for an integrated corpus query system. In *Proceedings of COMPLEX'94: 3rd Conference on Computational Lexicography and Text Research (Budapest, July 7–10 1994)*, pages 23–32, Budapest, Hungary, 1994. CMP-LG archive id 9408005.

[2] Oliver Christ. *The IMS Corpus Workbench Corpus Administrator's Manual*. Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, 1994. (Revised November 1994).

[3] Bruno M. Schulze. Entwurf und Implementierung eines Anfragesystems für Textcorpora. Diplomarbeit Nr. 1059, Universität Stuttgart, Institut für maschinelle Sprachverarbeitung (IMS) and Institut für Informatik, January 1994. (In German).