CorpusSearch 2: a tool for linguistic research

CorpusSearch 2 is a Java program that supports research in corpus linguistics. It is useful both for the construction of syntactically annotated (parsed) corpora and for searching them. Running CorpusSearch on an appropriately annotated corpus a user can automatically:

- find and count lexical and syntactic configurations of any complexity
- correct systematic errors
- code the linguistic features of corpus sentences for later statistical analysis

Both the input and output files of CorpusSearch are ordinary text files, with syntactic annotations in the Penn-Treebank format.

CorpusSearch 2 runs under any Java-supported operating system, including Linux, Macintosh, Unix and Windows. It requires Java 2, version 1.3 or later. In addition to being downloadable from this site, CorpusSearch is distributed with the Penn-Helsinki Parsed Corpora of Historical English.

Download Page

Features

Compatible Corpora

Users Guide

Credits

Report Bugs

Developers

Last modified: Fri Nov 20 13:37:00 EST 2009

CorpusSearch 2: a tool for linguistic research

Features of CorpusSearch 2:

- Tree search configurations in CS are defined in a Boolean query language over tree predicates.
- The output of a CS search is itself searchable.
- CS runs on any Java-supported platform.
- The CS query language contains many features to make searching easier and more intuitive for linguistic research.
- CS has extensive user configuration options.

For more details, read the CorpusSearch on-line Users Guide.

Download Page

CS Home

Report Bugs

Developers

Last modified: Wed Feb 23 19:43:04 EST 2005

SOURCEFORGE.NET

A list of corpora compatible with CorpusSearch

Download Page

CS Home

Report Bugs

Developers

The following historical corpora have been constructed to be compatible with CorpusSearch. Other parsed corpora can easily be made compatible with the program, as described in the Users Guide.

Parsed corpora of historical English:

- The Penn-Helsinki Parsed Corpus of Middle English, 2nd edition (PPCME2), currently available.
- The Penn-Helsinki Parsed Corpus of Early Modern English (PPCEME), currently available.
- The York-Helsinki Parsed Corpus of Old English Poetry, currently available.
- The York-Toronto-Helsinki Parsed Corpus of Old English Prose, currently available.
- The Brooklyn-Geneva-Amsterdam-Helsinki Parsed Corpus of Old English, currently available.
- The York-Helsinki Parsed Corpus of Early English Correspondence (PCEEC), currently available.

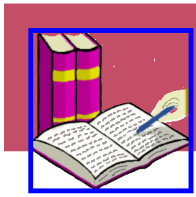
- The Penn Parsed Corpus of Modern British English (1700-1914), under construction at the University of Pennsylvania by Anthony Kroch and Beatrice Santorini.

Historical parsed corpora of other languages:

- **The Tycho Brahe Corpus**, a parsed corpus of historical Portuguese
  Charlotte Galves (U. of Campinas, Brazil) and collaborators.

- **Modéliser le changement: les voies du français**, a parsed corpus of historical French
  France Martineau (University of Ottawa) and collaborators.
  (Click here for the English language page.)

Last modified: Fri Nov 20 19:43:04 EST 2009



CorpusSearch 2 Users Guide

This guide evolves with the program it describes.
Suggestions for improvement are always welcome.
Please send them to:
CorpusSearch comments.

Click here for a free-standing pdf file of the entire Users Guide.

Updates
CorpusSearch Home

CorpusSearch 2: a tool for linguistic research

CorpusSearch was written by Beth Randall as part of a project at the University of Pennsylvania, directed by Anthony Kroch, to create large parsed corpora of historical English. The program design is due to Beth Randall, Ann Taylor and Anthony Kroch.

We thank the following individuals and institutions for their help and financial support in the development of CorpusSearch:

- Ann Taylor (University of Pennsylvania and University of York, Linguistics Departments), for extensive help in testing the program over the course of several years.
- Beatrice Santorini, Tom McFadden and others at the University of Pennsylvania and elsewhere for collaboration in using and testing the program.
- English Arts and Humanities Research Board Grant B/RG/AN5907/APN9528, Anthony Warner and Susan Pintzuk (University of York), Principal Investigators, for financial support.
- DARPA Grant N66001-00-1-8915, Martha Palmer (University of Pennsylvania, Computer Science Department), Principal Investigator, for financial support.
- NSF Grant BCS-0508731, Anthony Kroch (University of Pennsylvania, Linguistics Department), Principal Investigator, for financial support.
- Purchasers of licenses to earlier versions of CorpusSearch, for their encouragement and financial support.

Download Page

CS Home

Report Bugs

Developers

CorpusSearch 2: a tool for linguistic research

Program Documentation:

- For documentation of the CorpusSearch 2 source code, read the **JavaDoc documentation** (available soon).
- For details on program usage, read the CorpusSearch on-line Users Guide.

Download Page

CS Home

Report Bugs

Developers

Last modified: Wed Feb 23 19:43:04 EST 2005

CorpusSearch 2: a tool for linguistic research

CorpusSearch has been used to search Middle English, Old, Middle, and Modern English corpora, as well as corpora of Chinese, Korean and Yiddish. In order for CS to search a corpus it must meet the following formatting requirements:

1. Every sentence in the corpus must be completely parsed; that is, every word must be labeled and must be included within the outside brackets of some sentence.
2. Phrasal and part-of-speech labels may not contain a space or other white space character, nor may they begin with digits.
3. Constituents must be bracketed with parentheses -- "(" and ")" -- not with square brackets or other delimiters.
4. Every sentence must have a "wrapper", that is, an unlabeled pair of parentheses surrounding the sentence.

Below is an example of a sentence bracketed in accordance with these guidelines, using the labels of the PPCME2 and PPCEME. Note that CorpusSearch is indifferent to the choice of phrasal and part-of-speech labels.

> ((IP-MAT (ADVP-TMP (ADV Then)) (NP-SBJ (D the) (N child)) (VBD became) (ADJP (ADJR happier) (CONJ and) (ADJR happier)) (E_S .)))

For more information on corpus formatting for CorpusSearch see the CorpusSearch Users Guide.

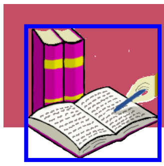
Download Page

CS Home

Report Bugs

Developers

Last modified: Wed Feb 23 20:11:20 EST 2005



Basic Concepts

Contents of this chapter:

Table of Contents
CorpusSearch Home

What is CorpusSearch?

CorpusSearch finds linguistic structures in a corpus of parsed, labelled sentences. It also has other features, including support for the automatic creation of coding strings for statistical analysis and the automatic creation of a lexicon for a corpus.

A new feature of CorpusSearch 2 is support for corpus creation, in the form of automated modification of corpus tree structures. This feature is useful for correction systematic errors and for applying global changes in annotation guidelines to an entire corpus.

input to CorpusSearch

CorpusSearch needs two pieces of information:

- a corpus of sentences to search (source file(s)).
- a specification of what structures to search for (command file).

source file(s)

A source file is any file that contains parsed, labelled sentences. This could be a file from the Penn Parsed Corpora of Historical English or from another parsed corpus. It could also be an output file from a previous search, or perhaps a file of sentences that the user has cut and pasted together. Any number of source files can be searched in a single one run of CorpusSearch.

command file

The command file contains a query, which describes the structures being searched for, and possibly additional control and output specifications. This additional material may specify the node boundaries within which to search, and may choose various options for specifying the form of the output.

output of CorpusSearch

CorpusSearch always builds a text output file, containing the sentences with the specified structure, and basic statistics.

search output

The output file contains the sentences that were found to contain the searched-for structure, along with comments describing where the structures were found. Statistics are kept detailing the number of "hits," that is, distinct constituents containing the structure, the number of matrix sentences ("tokens") containing hits, and the total number of tokens in the file. Notice that the number of hits may change depending on the definition of the boundary node.

coding output

CorpusSearch can be asked to create an output file in which a coding string is added to each boundary node in the corpus that matches a given query. The content of the columns in the coding string can be specified automatically by subqueries.

frames output

CorpusSearch can be asked to generate the set of all local syntactic environments within which a given word of the corpus occurs. Local environments are defined as syntactic sisters of the part-of-speech label of the word and are called local frames.

lexicon output

CorpusSearch can be asked to generate a lexicon for a corpus. The lexicon is a list of every word in the corpus along with the number of times it occurs under each part-of-speech label that it can have.

Top of page
Table of Contents

What's New in CorpusSearch 2

Contents of this chapter:

Table of Contents

CorpusSearch Home

changes to the query language

In previous versions of CorpusSearch, subqueries had to be appended one at a time, like this:

```
query: ((((A function B)
       AND (C function D))
       AND (E function F))
       AND (G function H))
```

In CorpusSearch 2, subqueries can be appended with any logical combination of parentheses. The above query is more easily written:

```
query: (A function B)
       AND (C function D)
       AND (E function F)
       AND (G function H)
```

Old-style queries still work.

additions to the query language

In previous versions of CS, the only query conjunction was "AND". CorpusSearch 2 has added "OR" and "NOT".

new node boundary

There is a new node-boundary option, $ROOT, which is a **variable** that stands for the root node of a matrix sentence (token), whatever its label may be. Even in cases where the root node has no label, it can be referred to with $ROOT.

new search functions

new search functions include:

- Dominates
- iDomsMod
- hasSister
- isRoot
- sameIndex

These search functions have new algorithms:

- iPrecedes
- Precedes

Also, search functions that take an integer argument no longer have the argument jammed onto the end of the function name, but are written with a space before the argument. For example, instead of:

```
(WRONG!) query: (CODING column2 q)
```

you should now write this:

```
query: (CODING column 2 q)
```

These functions are affected:

- column
- domsWords
- domsWords<
- domsWords>
- iDomsNumber
- iDomsTotal
- iDomsTotal<
- iDomsTotal>

new printing commands

- reformat_corpus
- print_only

changes to coding

Coding is now performed once per boundary node, instead of once per sentence (token).

Also, the command "coding_query:" must now appear before the coding query.

new capabilities

CorpusSearch 2 includes lexicon building, automated corpus revision, and local-frames production.

discontinued functions

iDoms_conj_switch: see iDomsMod
anyPrecedes: see precedes
print_complement: see NOT

Top of page
Table of Contents

Getting started with CorpusSearch

Contents of this chapter:

Table of Contents
CorpusSearch Home

paths and shells

In the description below, we assume that CorpusSearch is installed in the top-level directory and that other files also have locations that are simply specified. To put the program and other files into convenient locations and to define aliases that make running the program easier, it is necessary to learn something of how **paths** work on whatever system you are running. This is especially important if you are running CorpusSearch on a multiuser machine. The documentation for your operating system will contain a complete discussion of how paths work. The syntax differs somewhat across operating systems, though the concepts are the same.

If you are using a unix-derived system, including linux and Mac OS X, you should also learn something about what **shells** are and how they work. We assume here that you are using the c-shell, but you may find yourself in a Bourne shell (bash) environment, where the syntax of path specfication and other matters is a bit different. Again, documentation on shells is widely available.

invoking CorpusSearch

CorpusSearch now comes in a single jar file called "CS.jar," which must be installed in an appropriate directory (folder) of your computer. We will assume that you have installed CS.jar into the directory "FOO," which is at the root level of your hard disk. CorpusSearch can then be invoked by typing the following line at a command line prompt (here "%>") in a terminal window:

```
%>java -classpath /FOO/CS.jar csearch/CorpusSearch
```

A terminal window can be obtained under any flavor of unix/linux by launching an xterm under X11. On a Macintosh running OS X, it can also be obtained by launching the Terminal program. Under Windows, depending on the version of the operating system you are running, use Start>Run>cmd or Start>Run>command to launch the appropriate window. The -classpath switch can be left out if your shell initialization file or equivalent specifies the classpath.

Because Windows path syntax differs slightly from unix path syntax, you must invoke CorpusSearch under Windows with the following line, assuming that you have installed it in a directory "FOO" at the top of the C:\ drive:

```
%>java -classpath "C:\FOO\CS.jar" csearch/CorpusSearch
```

Note that, under Windows, the direction of the slashes changes between the class path and the command invocation.

To save typing, the following alias can be entered into your .cshrc file if you are running any variant of the c-shell on any unix system, including Mac OS X. An equivalent form exists for the bash shell.

```
alias CS 'java -classpath /FOO/CS.jar csearch/CorpusSearch'
```

If you put CorpusSearch anywhere but in a top-level directory, or if you install it on a multiuser machine, you must include the entire path to the CS.jar file in any command that invokes it.

your query/output directory

Let us assume that you have a corpus in the directory "corpus" at the root of your hard disk. Make a new sister directory of "corpus"; you might call it "corpus_stuff". This directory will hold your query files (ending with ".q"), and your output files (ending with ".out").

Here's a CorpusSearch command using the query file "inversion.q," run from a directory called "corpus_stuff" on a unix machine:

```
%>CS inversion.q ../corpus/*
```

This command will search the entire corpus (because of the "/*" after "corpus"). The output will appear in a file called "inversion.out" in the corpus_stuff directory.

Be patient; a search of a million word corpus takes a few minutes, depending on the complexity of the query. To run a search in the background under unix, write "&" at the end of your command:

```
%>CS inversion.q ../corpus/* &
```

Top of page
Table of Contents

The CorpusSearch Query Language

Contents of this chapter:

Table of Contents

CorpusSearch Home

about the query language

The CorpusSearch query language has these basic components:

- search-function calls. Each search function looks for one basic relationship, like "dominates" or "precedes".
- arguments to search-function calls. These describe the nodes being searched for. Search function arguments may take the form of an or-list, may include wild cards, and may be negated.
- AND, OR and NOT. AND, OR, and NOT are used as in basic formal logic.
- open parenthesis, "(", and close parenthesis, ")". Parentheses are used as in basic formal logic.

search function calls

The most basic query is a single search-function call. For instance, here is a query that searches for nodes labelled QP ("quantifier phrase") that immediately dominate nodes labelled CONJ ("co-ordinating conjunction"):

```
(QP iDominates CONJ)
```

and here is a sentence found by the query:

```
/~*
and so he is bo+te more and lasse to his seruaunt.
(CMWYCSER,351.2223)
*~/

/*
    1 IP-MAT: 9 QP, 10 CONJ bo+te
    1 IP-MAT: 9 QP, 12 CONJ and
*/

(0
   (1 IP-MAT (2 CONJ and)
             (3 ADVP (4 ADV so))
             (5 NP-SBJ (6 PRO he))
             (7 BEP is)
             (8 ADJP
                     (9 QP (10 CONJ bo+te) (11 QR more) (12 CONJ and) (13 QR lasse))
                     (14 PP (15 P to)
                            (16 NP (17 PRO$ his) (18 N seruaunt))))
             (19 E_S .))
```

```
      (ID CMWYCSER,351.2223))
```

Any number of search-function calls may be combined into more complex queries using AND, OR, and NOT.

wild cards and escaping wild cards

CorpusSearch supports two wild cards, namely * and #.

*

* works as in regular expressions, that is, it stands for any string of symbols. For instance, "CP*" means any label beginning with the letters CP (e.g. CP, CP-ADV, CP-QUE-SPE). "*-SPE" means any label ending with "-SPE", and *hersum* means any string containing the substring "hersum" (e.g., "hersumnesse", "unhersumnesse"). * by itself will match any string. * may be used anywhere in the function argument; beginning, middle or end.

escaping the asterisk (*)

Some labels, for example "*con*" ("subject elided under conjunction"), contain the character '*'. If you're looking for such a label, use \ (escape character) to show that you're searching for * and not using it as a wild card. For instance, to search for *con* dominated by a noun phrase, you could use this query:

```
(NP* dominates \*con\*)
```

to find (among others) this sentence:

```
/~*
ne did euyll.
(CMMANDEV,1.14)
*~/

/*
    1 IP-MAT: 3 NP-SBJ *con*
*/

(0
   (1 IP-MAT (2 CONJ ne)
             (3 NP-SBJ *con*)
             (4 DOD did)
             (5 NP-OB1 (6 N euyll))
             (7 E_S .))
      (ID CMMANDEV,1.14))
```

#

# is the wild card for digits. For instance, to find prepositions divided into parts, you could use this query:

```
(PP iDominates P#)
```

to find sentences like this:

```
/~*
Anone there $with all arose sir Gawtere
(CMMALORY,199.3135)
*~/

/*
```

```
    1 IP-MAT: 4 PP, 7 P21 $with
    1 IP-MAT: 4 PP, 8 P22 all
*/

(0
   (1 IP-MAT
             (2 ADVP-TMP (3 ADV Anone))
             (4 PP
                   (5 ADVP (6 ADV there))
                   (7 P21 $with)
                   (8 P22 all))
             (9 VBD arose)
             (10 NP-SBJ (11 NPR sir) (12 NPR Gawtere)))
      (ID CMMALORY,199.3135))
```

escaping integers

Integer arguments are expected for some search functions and not allowed for others. But suppose you want to search for a piece of text that is an integer, for instance a year. You can't do this:

```
(WRONG!) query: (1929 exists)
```

because "exists" won't take an integer argument. To cause the query parser to accept an integer as text, use a "\" as follows:

```
query: (\1929 exists)
```

logical operators

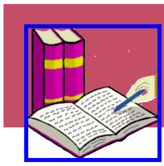
Search-function calls may be combined using the logical operators AND, OR, and NOT.

There are also logical operators that act on arguments to search functions. These are |, which means "or" for a list of arguments (e.g. "MD*|HV*" means "MD* or HV*"), and "!", which negates an argument (or list of arguments) (e.g. "NP-SBJ dominates !N" returns cases where NP-SBJ does not dominate N.)

regular expressions

CorpusSearch allows the use of regular expression syntax in the arguments to functions. For example, the expression "[xyz]" stands for a single character that is either an "x", a "y" or a "z". Note that the period character "." stands for any letter or digit and the sequence ".*" stands for any sequence of such characters. If the argument in the query contains a literal period, it must be escaped with a "\", as in the case of asterisk.

Top of page
Table of Contents



The CorpusSearch Search Functions

◀ ▶

Table of Contents
CorpusSearch Home

Contents of this chapter:

General considerations

We commonly refer to the first argument to a search function as "x", and the second argument as "y".

To save typing and to improve readability, CorpusSearch allows shorthands and lower-case/upper-case variations for the names of search functions. Acceptable variants are listed below with each function.

When a function has an integer argument, there is always a space between the function and argument. This syntax is a change from earlier versions of CorpusSearch.

Search functions

CCommands (variants: cCommands, ccommands)

A node x ccommands a node y if and only if:

1. neither x nor y dominates the other AND
2. the first branching node dominating x does dominate y.

In the following tree,

```
             A
            / \
           B   C
          / \   \
         D   E   F
```

B ccommands **C** and **F** and both **C** and **F** ccommand **B**, **D** and **E**. **D** and **E**, on the other hand, ccommand only each other. **A** ccommands no node because, being the root of the tree, it dominates all of the other nodes. The following query:

```
query: (NP-SBJ* idoms PRO$) AND (PRO$ ccommands NP*)
```

finds examples like:

```
(NP-SBJ (PRO$ his)
        (ADVR+Q ouermoch)
        (N fearinge)
        (PP (P of)
            (NP (PRO you))))
```

in which a possessive pronoun ccommands a noun phrase, here the object of a prepositional complement to the head noun.

Column (variants: column, Col, col)

"Column" is used to search columns of the CODING node, or any other leaf whose text is written in columns separated by ":".

If, for instance, you want to find sentences whose CODING node contains an "m" or "n" in the 7th column, use this query:

```
query:  (CODING column 7 m|n)
```

If you want to find sentences whose CODING node does not contain a "p" or "q" in the 4th column, use this query:

```
query:  (CODING column 4 !p|q)
```

Dominates (variants: dominates, Doms, doms)

dominates means "dominates to any generation." That is, y is contained in the sub-tree dominated by x. Dominates will accept text as y, but text as x will always return an empty set (text never dominates a subtree.) Notice that the following query uses the escape character, "\", to search for *arb*:

```
(IP-INF dominates \*arb*)
```

returns this sentence:

```
/~*
And soo by the counceil of Merlyn the kyng lete calle his barons to counceil,
(CMMALORY,14.419)
*~/

/*
    18 IP-INF: 19 NP-SBJ *arb*
*/
```

```
(
      (18 IP-INF (19 NP-SBJ *arb*)
                 (20 VB calle)
                 (21 NP-OB1 (22 PRO$ his) (23 NS barons))
                 (24 PP (25 P to)
                        (26 NP (27 N counceil))))
      (ID CMMALORY,14.419))
```

DomsWords (variants: domsWords, domswords)

domsWords counts the number of words dominated by the search-function argument. So "domsWords 4" means "dominates 4 words", domsWords 2 mean "dominates 2 words", and so on. A word in this case is defined as a leaf node that is not on the word_ignore_list. Here's the default word_ignore_list:

RMV:*|COMMENT|CODE|ID|LB|'|\"|,|E_S|0|**

Thus, traces, 0 complementizers, punctuation, and comments are not counted as words.

So this query:

```
node: NP*

(NP-OB* domsWords 3)
```

will return this structure (ignoring the trace *ICH*-1):

```
/~*
and by kynge Ban and Bors his counceile they lette brenne and destroy all the
contrey before them there they sholde ryde.
(CMMALORY,20.613)
*~/

/*
    24 NP-OB1: 27 N contrey
*/

(
      (24 NP-OB1 (25 Q all)
                 (26 D the)
                 (27 N contrey)
                 (28 CP-REL *ICH*-1))
      (ID CMMALORY,20.613))
```

(only the NP-OB1 node was printed in this output because the query file included the line "node: NP*").

DomsWords< (variants: domsWords<, domswords<)

domsWords< is just like domsWords except that it returns structures that dominate strictly less than the given number of words. For instance, this query:

```
(NP-OB* domsWords< 3)
```

will return this structure:

```
/~*
for it was I myself that cam in the lykenesse.
(CMMALORY,5.131)
```

```
*~/

/*
    6 NP-OB1: 9 PRO$+N myself
*/

(
      (6 NP-OB1 (7 PRO I)
                (8 NP-PRN (9 PRO$+N myself)))
      (ID CMMALORY,5.131))
```

(only the NP-OB1 node was printed in this output because the query file included the line "node: NP*").

DomsWords> (variants: domsWords>, domswords>)

domsWords> is just like domsWords except that it returns structures that dominate strictly more than the given number of words. For instance, this query:

```
(NP-OB* domsWords> 3)
```

will return this structure:

```
/~*
for she was called a fair lady and a passynge wyse,
(CMMALORY,2.9)
*~/

/*
    9 NP-OB1: 20 ADJ wyse
*/

(
      (9 NP-OB1
                (10 NP (11 D a) (12 ADJ fair) (13 N lady))
                (14 CONJP (15 CONJ and)
                          (16 NP (17 D a)
                                 (18 ADJP (19 ADV passynge) (20 ADJ wyse)))))
      (ID CMMALORY,2.9))
```

(only the NP-OB1 node was printed in this output because the query file included the line "node: NP*").

Exists (variants: exists)

exists searches for label or text anywhere in the sentence. For instance, this query:

```
(MD0 exists)
```

will find this sentence:

```
/~*
but I fere me that I shal not conne wel goo thyder /
(CMREYNAR,14.261)
*~/

/*
    10 IP-SUB: 15 MD0 conne
*/
```

```
(
      (10 IP-SUB
                 (11 NP-SBJ (12 PRO I))
                 (13 MD shal)
                 (14 NEG not)
                 (15 MD0 conne)
                 (16 ADVP (17 ADV wel))
                 (18 VB goo)
                 (19 ADVP-DIR (20 ADV thyder)))
      (ID CMREYNAR,14.261))
```

A common mistake is to use "exists" unneccessarily, as in this example:

```
(MD exists) AND (MD iPrecedes VB)
```

If a sentence contains the structure (MD iPrecedes VB), MD necessarily exists in the sentence. So this query would get the same result:

```
(MD iPrecedes VB)
```

HasSister (variants: hasSister, hassister)

x hasSister y if x and y have the same mother. It doesn't matter whether x precedes y or y precedes x. So this query:

```
node: IP*
query: (NP-SBJ hasSister BE*)
```

finds both of these sentences:

```
/~*
indeede I must be gone:
(DELONEY,69.13)
*~/
/*
1 IP-MAT-SPE:  5 NP-SBJ, 10 BE
*/


( (IP-MAT-SPE (PP (P+N indeede))
              (NP-SBJ (PRO I))
              (MD must)
              (BE be)
              (VBN gone)
              (. :))
  (ID DELONEY,69.13))

/~*
I pray you is it true?
(DELONEY,70.47)
*~/
/*
13 IP-SUB-SPE:  16 NP-SBJ, 14 BEP
*/


( (CP-QUE-SPE (IP-MAT-PRN-SPE (NP-SBJ (PRO I))
                              (CODE {TEMP:prn_ok})
                              (VBP pray)
                              (NP-ACC (PRO you)))
              (IP-SUB-SPE (BEP is)
                              (NP-SBJ (PRO it))
                              (ADJP (ADJ true)))
              (. ?))
  (ID DELONEY,70.47))
```

iDominates (variants: idominates, iDoms, idoms)

iDominates means "immediately dominates". That is, x dominates y if y is a child (exactly one generation apart) of x. So this query:

```
((NP* iDominates FP) AND (FP iDominates ane))
```

finds this sentence:

```
/~*
Sythen he ledes +tam by +tar ane,
(CMROLLEP,118.978)
*~/

/*
    1 IP-MAT: 11 NP, 13 FP ane
*/

(0
   (1 IP-MAT
             (2 ADVP-TMP (3 ADV Sythen))
             (4 NP-SBJ (5 PRO he))
             (6 VBP ledes)
             (7 NP-OB1 (8 PRO +tam))
             (9 PP (10 P by)
                   (11 NP (12 PRO$ +tar) (13 FP ane)))
             (14 E_S ,))
      (ID CMROLLEP,118.978))

/*
```

Notice that "iDominates" describes the relationship between a label and its associated text (e.g., "FP" and "ane").

iDomsFirst (variants: idomsfirst)

"iDomsFirst" means "immediately dominates as a first child."

For instance, this query:

```
node: IP*
query: (NP* iDomsFirst PRO$)
```

results in this output:

```
/~*
My Lady yor mother, I thanke God, is very well and cheerly,
(KNYVETT-1630,86.12)
*~/
/*
1 IP-MAT:  2 NP-SBJ, 3 PRO$
1 IP-MAT:  7 NP-PRN, 8 PRO$
*/

( (IP-MAT (NP-SBJ (PRO$ My)
                  (N Lady)
                  (NP-PRN (PRO$ yor) (N mother)))
          (, ,)
          (IP-MAT-PRN (NP-SBJ (PRO I))
                      (VBP thanke)
                      (NP-ACC (NPR God)))
          (, ,)
          (BEP is)
          (ADJP (ADJP (ADV very) (ADJ well))
                (CONJP (CONJ and)
                       (ADJX (ADJ cheerly))))
          (. ,))
  (ID KNYVETT-1630,86.12))
```

iDomsLast (variants: idomslast)

"iDomsLast" means "immediately dominates as a last child."

So this query:

```
node: IP*
query: (IP* iDomsLast BEN)
```

results in this output:

```
/~*
but keepes her chamber because of the Bitter weather that hath been.
(KNYVETT-1630,86.13)
*~/
/*
31 IP-SUB:  31 IP-SUB, 36 BEN
*/

( (IP-MAT (CONJ but)
          (NP-SBJ *con*)
          (VBP keepes)
          (NP-ACC (PRO$ her) (N chamber))
          (PP (P+N because)
              (PP (P of)
                  (NP (D the)
                      (ADJ Bitter)
                      (N weather)
                      (CP-REL (WNP-1 0)
                              (C that)
                              (IP-SUB (NP-SBJ *T*-1)
                                      (HVP hath)
                                      (BEN been))))))
          (. .))
  (ID KNYVETT-1630,86.13))
```

iDomsMod (variants: idomsmod)

x immediately dominates (mod z) y if x dominates y, and the only nodes intervening on the path from x to y (if any) are members of z. Note that if no intervening nodes at all occur on the path from x to y, the query function is still true. The most obvious use of this function is to search within conjuncts. Thus, to search for pronominal subjects within conjoined NPs, you can use the following query:

```
node: IP*
query: (NP-SBJ iDomsMod NP*|CONJ* PRO)
```

finds this sentence:

```
/~*
So by the entrete at the last the kyng and she met togyder.
(CMMALORY,4.104)
*~/
/*
1 IP-MAT:  21 NP-SBJ, 31 PRO, 27 CONJP
*/


(0  (1 IP-MAT (2 ADVP (3 ADV So))
              (5 PP (6 P by)
                    (8 NP (9 D the) (11 N entrete)))
              (13 PP (14 P at)
                     (16 NP (17 D the) (19 ADJ last)))
              (21 NP-SBJ (22 NP (23 D the) (25 N kyng))
                         (27 CONJP (28 CONJ and)
                                   (30 NP (31 PRO she))))
              (33 VBD met)
              (35 ADVP (36 ADV togyder))
              (38 E_S .))
    (40 ID CMMALORY,4.104))
```

The query

```
node: IP*
query: (NP-SBJ iDomsMod NP*|CONJ* !PRO)
```

would also find the above sentence because "NP-SBJ iDomsMod NP" is true of the full NP "the king."

iDomsNumber (variants: idomsnumber, iDomsNum, idomsnum)

"iDomsNumber" means "immediately dominates as the #th child". That is, x immediately dominates y as the #th child if x immediately dominates y and y is the #th child of x. Note that "iDomsNumber 1" is identical to "iDomsFirst." This query:

```
(CP-DEG iDomsNumber 1 C)
```

produces this output:

```
/~*
And Merlion was so disgysed that kynge Arthure knewe hym nat,
(CMMALORY,30.939)
*~/

/*
    1 IP-MAT: 9 CP-DEG, 10 C that
*/

(0
   (1 IP-MAT (2 CONJ And)
             (3 NP-SBJ (4 NPR Merlion))
             (5 BED was)
             (6 ADJP (7 ADVR so)
                     (8 VAN disgysed)
                     (9 CP-DEG (10 C that)
                               (11 IP-SUB
                                          (12 NP-SBJ (13 NPR kynge) (14 NPR Arthure))
                                          (15 VBD knewe)
                                          (16 NP-OB1 (17 PRO hym))
                                          (18 NEG nat))))
             (19 E_S ,))
      (ID CMMALORY,30.939))
```

iDomsOnly (variants: idomsonly)

iDomsOnly means "immediately dominates as an only child." That is, x immediately dominates y as an only child if x immediately dominates y and y is the only legitimate child of x. So this query:

```
(ADJP iDomsOnly Q*)
```

results in this output:

```
/~*
But after my lytyll wytt it semeth me, sauynge here reuerence, +tat is more.
(CMMANDEV,123.2992)
*~/

/*
    23 IP-SUB: 27 ADJP, 28 QR more
*/

(
      (23 IP-SUB
                 (24 NP-SBJ (25 D +tat))
                 (26 BEP is)
                 (27 ADJP (28 QR more)))
      (ID CMMANDEV,123.2992))
```

iDomsTotal (variants: idomstotal)

iDomsTotal counts the number of nodes immediately dominated by the search- function argument. Traces count as daughters unless they are added to the ignore list. The following query:

```
(NP-OB* iDomsTotal 3)
```

yields this output:

```
/~*
And +tere it lykede him to suffre many repreuynges and scornes for vs
(CMMANDEV,1.4)
*~/

/*
    10 IP-INF-1: 13 NP-OB1, 16 CONJP
*/

(
      (10 IP-INF-1 (11 TO to)
                   (12 VB suffre)
                   (13 NP-OB1 (14 Q many)
                              (15 NS repreuynges)
                              (16 CONJP (17 CONJ and)
                                        (18 NX (19 NS scornes))))
                   (20 PP (21 P for)
                          (22 NP (23 PRO vs))))
      (ID CMMANDEV,1.4))
```

Here, the 3 nodes immediately dominated by NP-OB1 are labelled Q, NS, and CONJP.

iDomsTotal< (variants: idomstotal<)

iDomsTotal< is like iDomsTotal except that it returns structures that immediately dominate strictly less than the given number of nodes. So this query:

```
(NP-OB* iDomsTotal< 3)
```

yields this output:

```
/~*
& take of euereche iliche myche
(CMHORSES,125.397)
*~/

/*
    1 IP-IMP: 8 NP-OB1, 9 QP
*/

(0
   (1 IP-IMP (2 CONJ &)
             (3 VBI take)
             (4 PP (5 P of)
                   (6 NP (7 Q euereche)))
             (8 NP-OB1
                       (9 QP (10 ADV iliche) (11 Q myche))))
      (ID CMHORSES,125.397))
```

iDomsTotal> (variants: idomstotal>)

iDomsTotal> is like iDomsTotal except that it returns structures that immediately dominate strictly more than the given number of nodes. So this query:

```
(NP-OB* iDomsTotal> 3)
```

will yield this output:

```
/~*
& aftur tak an hot yre +tat is smal bi-fore
(CMHORSES,95.119)
*~/

/*
    1 IP-IMP: 6 NP-OB1, 10 CP-REL
*/

(0
   (1 IP-IMP (2 CONJ &)
             (3 ADVP-TMP (4 ADV aftur))
             (5 VBI tak)
             (6 NP-OB1 (7 D an)
                       (8 ADJ hot)
                       (9 N yre)
                       (10 CP-REL (11 WNP-1 0)
                                  (12 C +tat)
                                  (13 IP-SUB (14 NP-SBJ *T*-1)
                                             (15 BEP is)
                                             (16 ADJP (17 ADJ smal))
                                             (18 ADVP-LOC (19 ADV bi-fore))))))
      (ID CMHORSES,95.119))
```

InID (variants: inID)

"inID" is true of substrings of the ID node. This functin is introduced because the ID node, being outside of the parsed sentence, cannot serve as an argument of a search function. In particular, (ID iDominates *) will return no hits.

Here's a typical ID node from the Malory parsed file in the Middle English corpus:

```
(ID CMMALORY,3.41)
```

To isolate Malory sentences from an output file, you could use this query:

```
query:  (*MALORY* inID)
```

iPrecedes (variants: iprecedes, iPres, ipres)

This function is true if and only if its first argument immediately precedes its second argument in the text string spanned by the parse tree.

The algorithm for "x iPrecedes y" runs as follows:

1.) Find x.

2.) If x has an immediately following sister, then that sister and all its leftmost descendants (that is, the first child of the sister, the first child of the first child, and on as far as the tree goes) are candidates for y.

3.) If x has no immediately following sister, recurse from 2.) with the mother of x in place of x.

The following query:

```
query: ([1]as iPrecedes sone) AND (sone iPrecedes [2]as)
```

produces this output:

```
/~*
and as sone as he myght he toke his horse
(CMMALORY,206.3401)
*~/
/*
1 IP-MAT:  6 as, 8 sone, 11 as
*/


( (IP-MAT (CONJ and)
          (ADVP-TMP (ADVR as)
                    (ADV sone)
                    (PP (P as)
                        (CP-CMP (WADVP-1 0)
                                (C 0)
                                (IP-SUB (ADVP-TMP *T*-1)
                                        (NP-SBJ (PRO he))
                                        (MD myght)
                                        (VB *)))))
          (NP-SBJ (PRO he))
          (VBD toke)
          (NP-OB1 (PRO$ his) (N horse)))
  (ID CMMALORY,206.3401))
```

IsRoot (variants: isRoot, isroot)

isRoot searches for the argument label at the root of the tree of the parsed token. For instance, this query:

```
query: (CP* isRoot)
```

will return all tokens in the corpus whose root is a CP, for instance, the following sentence:

```
/~*
why thou whoreson when wilt thou be maried?
(DELONEY,79.296)
*~/
/*
1 CP-QUE-SPE:  1 CP-QUE-SPE
*/


( (CP-QUE-SPE (INTJP (WADV why))
              (NP-VOC (PRO thou) (N$+N whoreson))
              (WADVP-1 (WADV when))
              (IP-SUB-SPE (ADVP *T*-1)
                          (MD wilt)
                          (NP-SBJ (PRO thou))
                          (BE be)
                          (VAN maried))
              (. ?))
  (ID DELONEY,79.296))
```

IsRoot ignores the node boundary set by the query and returns results based only on the label of the root of the parse tree of each token in the input file.

Precedes (variants: precedes, Pres, pres)

"x precedes y" means "x comes before y in the tree but x does not dominate y". So this query:

```
(VB precedes NP-OB*)
```

produces this output:

```
/~*
thenne have ye cause to make myghty werre upon hym. '
(CMMALORY,2.25)
*~/

/*
    9 IP-INF-PRP: 11 VB make, 12 NP-OB1
*/

(
      (9 IP-INF-PRP (10 TO to)
                    (11 VB make)
                    (12 NP-OB1 (13 ADJ myghty)
                               (14 N werre)
                               (15 PP (16 P upon)
                                      (17 NP (18 PRO hym)))))
      (ID CMMALORY,2.25))
```

SameIndex (variants: sameIndex, sameindex)

x sameIndex y finds structures where x ends with the same index as y. This is useful in searching for antecedents with the same index as a trace. For instance, this query:

```
node: IP*
query: (NP* iDoms \*exp\*) AND (NP* sameIndex CP*)
```

finds this sentence:

```
/~*
hym thought there was com into hys londe gryffens and serpentes,
(CMMALORY,33.1031)
*~/
/*
1 IP-MAT:  2 NP-SBJ-1, 3 *exp*, 9 CP-THT-1
*/

( (IP-MAT (NP-SBJ-1 *exp*)
          (NP-OB2 (PRO hym))
          (VBD thought)
          (CP-THT-1 (C 0)
                    (IP-SUB (NP-SBJ-2 (EX there))
                            (BED was)
                            (VBN com)
                            (PP (P into)
                                (NP (PRO$ hys) (N londe)))
                            (NP-2 (NS gryffens) (CONJ and) (NS serpentes))))
          (E_S ,))
  (ID CMMALORY,33.1031))
```

Top of page
Table of Contents

CorpusSearch Logical Operators

Table of Contents
CorpusSearch Home

Contents of this chapter:

AND

AND, in its simplest form, returns trees in which both conjuncts hold within a single boundary node. For instance, this query:

```
node: IP*

query: (NP-TMP* iDominates ADV*)
       AND (TO iPrecedes VB)
```

yields this output:

```
/*
4 IP-INF-SBJ:  5 NP-TMP, 6 ADV+NS, 8 TO, 10 VB
*/

( (IP-MAT (CONJ but)
          (IP-INF-SBJ (NP-TMP (ADV+NS oftymes))
                      (TO to)
                      (VB rede)
                      (NP-OB1 (PRO it)))
          (MD shal)
          (VB cause)
          (NP-OB1 (PRO it))
          (IP-INF (ADVP (ADV wel))
                  (TO to)
                  (BE be)
                  (VAN vnderstande))
          (E_S /))
  (ID CMREYNAR,6.10))
```

AND; same-instance

AND has been implemented with a default feature that we call "same-instance." If the same label occurs twice in search functions conjoined by AND, CorpusSearch assumes that the two occurrences should refer to the same node in the tree. Thus, the following query

```
(IP* iDomsNumber 1 VBP|VBD) AND (IP* iDomsNumber 2 ADVP|PP*)
```

returns only trees where the same IP node has the described number 1 and 2 children. Trees containing one IP with number 1 child VBP and some other IP with number 2 child ADVP are not returned.

The same-instance assumption is triggered by matching argument label strings, so that

```
(ADVP precedes MD|HV*|VB*) AND (MD|HV*|VB* precedes NP-SBJ)
```

returns only sentences with the same instance of MD|HV*|VB*, but

```
(ADVP precedes MD|VB*|HV*) AND (MD|HV*|VB* precedes NP-SBJ)
```

returns sentences with the same instance or different instances (because the argument lists do not match as strings due to the difference in order of elements.)

Same-instance does not apply within single clauses of a query. Thus the query (`ADVP precedes ADVP`) is not vacuous.

AND; same-instance with prefix indices

If you need to specify which arguments coincide (that is, refer to the same instance) and which don't, you can use prefix indices. Matching arguments with the same prefix index must coincide, matching arguments with different prefix indices must not coincide. Prefix indices must be enclosed by the square brackets "[" and "]".

For example, suppose you are looking for two sister noun-phrases that each immediately dominate a pronoun. Use prefix indices as follows:

```
([1]NP* hasSister [2]NP*) AND ([1]NP* iDominates [3]PRO) AND ([2]NP* iDominates
[4]PRO)
```

to find sentences like this one:

```
/~*
And +tere it lykede him to suffre many repreuynges and scornes for vs
(CMMANDEV,1.4)
*~/

/*
    1 IP-MAT: 5 NP-SBJ-1, 8 NP-OB2, 6 PRO it, 9 PRO him
*/

(0
   (1 IP-MAT (2 CONJ And)
             (3 ADVP-LOC (4 ADV +tere))
             (5 NP-SBJ-1 (6 PRO it))
             (7 VBD lykede)
             (8 NP-OB2 (9 PRO him))
             (10 IP-INF-1 (11 TO to)
                          (12 VB suffre)
                          (13 NP-OB1 (14 Q many)
                                     (15 NS repreuynges)
                                     (16 CONJP (17 CONJ and)
                                               (18 NX (19 NS scornes))))
                          (20 PP (21 P for)
                                 (22 NP (23 PRO vs)))))
```

```
        (ID CMMANDEV,1.4))
```

Here's another example:

```
query: (IP-SMC iDoms [1]NP*)
   AND ([1]NP* iDoms [3]\**)
   AND (IP-SMC iDoms [2]NP*)
   AND ([2]NP* iDoms [4]\**)
```

This query searches for a node labelled IP-SMC which immediately dominates two different NP* nodes, each immediately dominating a trace. In this example, the two mentions of IP-SMC must refer to the same node in the tree (same-instance); [1]NP* and [2]NP* must refer to different nodes (because of the different prefix indices); similarly, [3]** and [4]** must not coincide. If the substrings following the indices were not identical, then the arguments would not be forced to pick out distinct nodes.

Here's a sentence found by the above query:

```
/~*
+After +t+am L+acedemonie gecuron him to ladteowe, Ircclidis w+as haten,
(OR4,1.53.30.12)
*~/

/*
    23 IP-SMC: 24 NP-NOM *-2, 25 NP-NOM-PRD *ICH*-1
    23 IP-SMC: 25 NP-NOM-PRD *ICH*-1, 24 NP-NOM *-2
*/


(0  (1 CODE )
  (2 IP-MAT
            (3 PP (4 P +After)
                  (5 NP-DAT (6 D^D +t+am)))
            (7 NP-NOM (8 NPR^N L+acedemonie))
            (9 VBDI gecuron)
            (10 NP-DAT-RFL-ADT (11 PRO|D him))
            (12 PP (13 P to)
                   (14 NP-DAT (15 N|D ladteowe)))
            (16 , ,)
            (17 IP-MAT-PRN (18 NP-NOM-2 *pro*)
                           (19 NP-NOM-1 (20 NPR^N Ircclidis))
                           (21 BEDI w+as)
                           (22 VBN haten)
                           (23 IP-SMC (24 NP-NOM *-2)
                                      (25 NP-NOM-PRD *ICH*-1)))
            (26 . ,))
  (27 ID OR4,1.53.30.12))
```

OR

OR is logical disjunction. "(FOO) OR (BAR)" returns all subtrees rooted in an instance of the query's selected node boundary in which either the property "FOO" or the property "BAR" or **both** hold. "FOO" and "BAR" may consist of single search functions or be built up out of conjunctions, disjunctions and negations of simple search functions.

NOT

WARNING: NOT is currently under active development. It does not yet work correctly in any but the simplest cases. Avoid it except for testing purposes.

NOT returns trees rooted in the node boundary that do not contain the described structure. It differs from ! because none of the arguments need to appear in the node boundary-defined domain.

For instance,

```
NOT(NP* precedes VB*)
```

returns trees that do not contain the structure (NP* precedes VB*), including those that contain neither NP* nor VB*.

On the other hand,

```
(NP* iPrecedes !VB*)
```

returns trees that contain an NP* which does not iPrecede VB*.

! (not)

! is used to negate one argument to a search function.

For instance, suppose you're looking for sentences in which the nodes immediately dominated by the subject do not include a pronoun. You could use this query:

```
(NP-SBJ* iDominates !PRO*)
```

to obtain sentences like this:

```
/~*
a runde fot & +ticke bi-come+t an hors wel.
(CMHORSES,87.17)
*~/

/*
    1 IP-MAT: 2 NP-SBJ, 10 ADJ +ticke
*/

(0
   (1 IP-MAT
             (2 NP-SBJ (3 D a)
                       (4 ADJP (5 ADJ runde)
                               (6 CONJP *ICH*-1))
                       (7 N fot)
                       (8 CONJP-1 (9 CONJ &))
                       (10 ADJ +ticke))
             (11 VBP bi-come+t)
             (12 NP-OB1 (13 D an) (14 N hors))
             (15 ADVP (16 ADV wel))
             (17 E_S .))
      (ID CMHORSES,87.17))
```

! one argument at a time

CorpusSearch does not allow you to negate both arguments to a single search function. So this is *not* a legitimate command, and its appearance will abort a search:

```
(!NP-SBJ iPrecedes !VBD)
```

ordering ! and prefix indices

If you need to use both ! and prefix indices, put the ! before the indices.

For instance, suppose you're looking for sentences that contain a subject that precedes the object, and neither the subject nor the object contains a pronoun. You could use this query:

```
    (NP-SBJ* precedes NP-OB1*)
AND (NP-SBJ* iDominates ![1]PRO*))
AND (NP-OB1* iDominates ![2]PRO*))
```

to obtain sentences like these:

```
/~*
& +tat schal be a good hors.
(CMHORSES,85.9)
*~/

/*
    1 IP-MAT: 3 NP-SBJ, 7 NP-OB1, 4 D +tat, 10 N hors
*/

(0
   (1 IP-MAT (2 CONJ &)
             (3 NP-SBJ (4 D +tat))
             (5 MD schal)
             (6 BE be)
             (7 NP-OB1 (8 D a) (9 ADJ good) (10 N hors))
             (11 E_S .))
      (ID CMHORSES,85.9))
```

Notice that it is necessary to use prefix indices before the PRO* labels. Otherwise, CorpusSearch would try to find an NP-SBJ* and an NP-OB1* both dominating the *same* not-PRO* object, and would come up empty.

| (or argument)

Any number of arguments to a search function may be linked together into an argument list using |, which means "or". For instance,

```
(*VB*|*HV*|*BE*|*DO*|*MD* iPrecedes NP-SBJ*)
```

means "*VB* or *HV* or *BE* or *DO* or *MD* immediately precedes NP-SBJ*," and will find sentences like this:

```
/~*
+Tan was pompe & pryde cast down & leyd on syde.
(CMKEMPE,2.12)
*~/

/*
    2 IP-MAT-1: 5 BED was, 6 NP-SBJ
*/

(
      (2 IP-MAT-1
                 (3 ADVP-TMP (4 ADV +Tan))
                 (5 BED was)
                 (6 NP-SBJ (7 N pompe) (8 CONJ &) (9 N pryde))
                 (10 VAN cast)
                 (11 RP down))
      (ID CMKEMPE,2.12))
```

negating a list

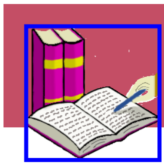
If a list is preceded by !, the entire list is negated. So,

```
(!*VB*|*HV*|*BE*|*DO*|*MD* iPrecedes NP-SBJ*)
```

means, "none of these (*VB* or *HV* or *BE* or *DO* or *MD*) iPrecedes NP-SBJ*", and finds sentences like this:

```
/~*
& sche wold not consentyn in no wey,
(CMKEMPE,3.34)
*~/

/*
    1 IP-MAT: 2 CONJ &, 3 NP-SBJ
*/

(0
   (1 IP-MAT (2 CONJ &)
             (3 NP-SBJ (4 PRO sche))
             (5 MD wold)
             (6 NEG not)
             (7 VB consentyn)
             (8 PP (9 P in)
                   (10 NP (11 Q no) (12 N wey)))
             (13 E_S ,))
      (ID CMKEMPE,3.34))
```

Top of page
Table of Contents



The CorpusSearch Command File

Table of Contents

CorpusSearch Home

Contents of this chapter:

Introduction

Every command file must contain a search specification command and ordinary query files must contain a value for the search control command **node:**. The extension of a command file is determined by the search specification command that it contains See below.

placement of commands

The preamble to a command file consists of the search control commands and the output format commands. These may appear in any order with respect to one another but they must all appear before the query specfication. Comments may appear anywhere. Many commands have default values which are used if no value is found in the command file. The **query:** command itself is obligatory, as is the **node:** command.

boolean shorthand

For commands that take a boolean argument, CorpusSearch will accept any of these strings: "true", "TRUE", "T", "t", or "false", "FALSE", "F", "f".

node label strings

Many commands, including query language clauses, can accept strings of alternative label values as well as single node labels. These alternatives are separated by the vertical bar character "|" without any spaces.

nodes to ignore

There are some nodes in the corpus that we usually don't want to consider as part of the structure of the sentence, for instance, punctuation, line breaks, page numbers, and comments. These and other nodes should usually also be ignored when a query function counts the number of words in a constituent. In deciding whether a function is matched by a given structure in the corpus, CorpusSearch will ignore nodes whose labels are contained in the "ignore-list". If the function is a word counting function, CorpusSearch ignores the nodes on the "word-ignore-list". Below are the default versions of the two ignore-lists. Note that traces and empty complementizers (** and 0) are on the default word-ignore-list but not on the default ignore-list.

```
ignore_nodes: COMMENT|CODE|ID|LB|'|\"|,|E_S|.|/|RMV:*
ignore_words:  COMMENT|CODE|ID|LB|'|\"|,|E_S|.|/|RMV:*|0|\**
```

For instance, if you run this query:

```
(NP* iPrecedes PP*)
```

This sentence will be returned:

```
/*
 1 IP-MAT-SPE: 5 NP-1, 9 PP
*/
/~*
There ar two bretheren beyond the see,
(CMMALORY,15.439)
*~/

(0
(1 IP-MAT-SPE
              (2 NP-SBJ-1 (3 EX There))
              (4 BEP ar)
              (5 NP-1 (6 NUM two) (7 NS bretheren))
              (8 CODE <P_15>)
              (9 PP (10 P beyond)
                    (11 NP (12 D the) (13 N see)))
              (14 E_S ,))
(15 ID CMMALORY,15.439))
```

Notice that NP-1 immediately precedes PP in spite of the intervening node (8 CODE <P_15>). This is because CODE is on the default ignore-list.

We will sometimes refer to nodes that are not to be ignored as "legitimate" nodes.

Search control commands

node: <node_boundary string>

Required element in every command file of the query type. A query file without a node specification produces an ERROR. The node specification is a node label or a disjunction of labels.

The value of **node:** gives CorpusSearch a node boundary within which to search. The list of labels gives boundaries that any structure you search for will fall within; for example, IP* would yield all the basic clauses in the corpus, and $ROOT is the topmost level of every syntactic tree, whatever its label. In the case of searches on the output of a previous search in which nodes_only is set to "true", $ROOT refers to the root of the tree, which will have the label of the node boundary.

Whenever you want to consider the entire tree as the domain within which to search use

```
node: $ROOT
```

The choice of node boundary determines the following:

- the counting of hits, defined as "number of distinct node boundaries containing the structure";
- what nodes are removed if remove_nodes is true;
- what nodes are printed if nodes_only is true.

To illustrate this, we ran the same query with different node boundaries on a simple file containing one sentence. First we ran the query with the node boundary, IP*|$ROOT:

```
node:   IP*|$ROOT
query:  (NP* iDominates PRO*)
```

Here's the output; notice that 1 hit is counted because there was one IP* node `(1 IP-MAT` containing both NP* nodes:

```
/~*
and he made them grete chere out of mesure
(CMMALORY,2.13)
*~/

/*
    1 IP-MAT: 3 NP-SBJ, 4 PRO he
    1 IP-MAT: 6 NP-OB2, 7 PRO them
*/

(0
   (1 IP-MAT (2 CONJ and)
             (3 NP-SBJ (4 PRO he))
             (5 VBD made)
             (6 NP-OB2 (7 PRO them))
             (8 NP-OB1 (9 ADJ grete) (10 N chere))
             (11 ADVP (12 ADV out)
                      (13 PP (14 P of)
                             (15 NP (16 N mesure)))))
      (ID CMMALORY,2.13))

/*
    FOOTER
    source file:  CMMALORY
    hits found:  1
    sentences containing the hits:  1
    total sentences searched:  1
*/
```

Next we ran the query with node boundary NP*:

```
node:   NP*
nodes_only: t
query:  (NP* iDominates PRO*)
```

Here's the output; this time 2 hits are counted, because there are two distinct NP* nodes (`(3 NP-SBJ` and `(6 NP-OB2`. Because nodes_only is set to true in this query, only the NP* nodes are printed:

```
/~*
and he made them grete chere out of mesure
(CMMALORY,2.13)
*~/

/*
    3 NP-SBJ: 4 PRO he
    6 NP-OB2: 7 PRO them
*/

(
      (3 NP-SBJ (4 PRO he))
      (ID CMMALORY,2.13))

(
      (6 NP-OB2 (7 PRO them))
      (ID CMMALORY,2.13))


/*
    FOOTER
    source file:  CMMALORY
    hits found:  2
    sentences containing the hits:  1
    total sentences searched:  1
*/
```

add_to_ignore: (label_list string)

default "" (empty string)

adds given labels to the ignore_list. For instance,

```
add_to_ignore:  \**
```

will tell CorpusSearch to ignore traces for this search. When nodes are ignored, they are not considered as possible arguments for search functions. For example, ignoring traces means that IPs with subject traces due to movement of the subject to a position outside the IP will behave in searches as though they had no subject. Thus, whether a given node type should appear on the ignore list depends on the purpose of the search.

ignore_nodes: (ignore_list string)

default COMMENT|CODE|ID|LB|'|\"|,|E_S|.|/|RMV:*

tells CorpusSearch what nodes to ignore.

To replace the default ignore-list with your own ignore-list, include this command in your command file:

```
ignore_nodes:  <your_ignore_list>
```

To tell CorpusSearch not to ignore any nodes, include this command in your command file:

```
ignore_nodes:   null
```

If you try to search for an item that is on the ignore_list, you'll get an error message. For instance, this query:

```
(NP-SBJ* iPrecedes CODE)
```

generates this message:

```
WARNING!  CODE in y_argument to iPrecedes is on the ignore_list.

    To make the ignore_list empty, add this line to your command file:

        ignore_nodes: null

    To write your own ignore_list, add this line to your command file:

        ignore_nodes:
```

The program goes ahead and runs as usual, but if you don't get the results you were looking for, you should probably change the ignore_list.

ignore_words: (word_ignore_list string)

default COMMENT|CODE|ID|LB|'|\"|,|E_S|.|/|RMV:*|0|**

tells CorpusSearch what nodes to ignore in counting words

To replace the default word-ignore-list with your own word-ignore-list, include this command in your command file:

```
ignore_words:  <your_word_ignore_list>
```

To tell CorpusSearch not to ignore any nodes in counting words, include this command in your command file:

```
ignore_words:   null
```

To add nodes to the word-ignore-list, use

```
add_to_ignore_words:
```

The following search functions are governed by the word-ignore-list: DomsWords, DomsWords<, DomsWords>. All other functions use the main ignore-list.

Output format commands

These commands do not in any way influence the current search. They only give instructions about how the results of the current search should be printed to the output file. However, because these commands can cause the output of the current search to take different forms, they may influence future searches which will take as their input the output of the current search.

begin_remark: (remark string) end_remark

default "" (empty string)

tells CorpusSearch to print user's remark in the output Preface. This is a way for the user to record a note, for instance to remember the goal of the search.

For instance, the command file "pro-obj.q" contains this command:

```
begin_remark:
        pronoun objects
```

```
end_remark
```

which is printed in the output preface like this:

```
/*
    PREFACE:  regular output file.
    CorpusSearch copyright Beth Randall 1999.
    Date:  Wed Nov 03 19:12:03 EST 1999

    command file:       pro-obj.q
    input file:         ipmat-2vb.out
    output file:        pro-obj.out

    remark:
        pronoun objects

    node:   IP*
    query:  (NP-OB* iDominates PRO)
*/
```

nodes_only: (boolean true or false)

default false

If true, CorpusSearch prints out only the nodes that contain the structure described in "query".

If false, CorpusSearch prints out the entire sentence that contains the structure described in "query".

For instance, suppose you have this query:

```
node:  ADVP*

nodes_only: t
query:  (ADVP* iDominates ADVP*)
```

Here's what a piece of the output looks like with **nodes_only** true.

```
/~*
certayn and wit-owte doute, Ihon is is name.
(CMAELR3,45.574)
*~/

/*
 2 ADVP: 3 ADVP
*/

(     (ADVP
            (ADVP (ADV certayn))
            (CONJP (CONJ and)
                   (PP (P wit-owte)
                       (NP (N doute))))
            (, ,))(ID CMAELR3,45.574))
```

And here's the same piece of output with **nodes_only** false:

```
/~*
certayn and wit-owte doute, Ihon is is name.
(CMAELR3,45.589)
*~/
```

```
/*
 2 ADVP: 3 ADVP
*/

(
(IP-MAT
        (ADVP
              (ADVP (ADV certayn))
              (CONJP (CONJ and)
                     (PP (P wit-owte)
                         (NP (N doute)))))
        (, ,)
        (NP-OB1 (NPR Ihon))
        (BEP is)
        (NP-SBJ (PRO$ is) (N name))
        (E_S .))
(ID CMAELR3,45.589))
```

print_complement: (boolean true or false)

default false

In the normal case CorpusSearch prints as output only nodes or tokens that match the query. Setting ***print_complement*** to true causes CorpusSearch to print not only the matching tokens (in the regular output file, extension .out), but also all the tokens that don't match, in a separate file called the complement file (extension .cmp). Thus, ***print_complement*** is a form of NOT applied to queries. Generally ***print_complement*** should be used on the output of a previous search that has narrowed down the possibilities to some set that can be meaningfully divided; using it on corpus files will generally result in a completely meaningless set of tokens.

Examples: the following query could be used on an output file containing all IPs with objects to divide the IPs into two sets: those with two objects (in the .out file) and those with one (in the .cmp file). The first example is from the regular output file and matches the query, that is, it has two objects. The second example is from the complement file and does not match the query; it has only one object.

```
     print_complement: t
     node: IP*
     query: ((IP* iDoms [1]NP-OB*)
     AND (IP* iDoms [2]NP-OB*))
```

from the regular output file:

```
/~*
And there is no knyght now lyvynge that ought to yelde God so grete thanke os
ye,
(CMMALORY,655.4474)
*~/
/*
1 IP-SUB-SPE: 6 NP-OB2, 8 NP-OB1
1 IP-SUB-SPE: 8 NP-OB1, 6 NP-OB2
*/

(0 (1 IP-SUB-SPE (2 NP-SBJ *T*-2)
                 (3 MD ought)
                 (4 TO to)
                 (5 VB yelde)
                 (6 NP-OB2 (7 NPR God))
                 (8 NP-OB1 (9 ADJP (10 ADVR so) (11 ADJ grete))
                            (12 N thanke)
                            (13 PP (14 P os)
                                   (15 NP (16 PRO ye)))))
      (ID CMMALORY,655.4474))
```

from the complement file:

```
/~*
The kynge lyked and loved this lady wel,
(CMMALORY,2.12)
*~/

(0  (1 IP-MAT (2 NP-SBJ (3 D The) (4 N kynge))
              (5 VBD (6 VBD lyked) (7 CONJ and) (8 VBD loved))
              (9 NP-OB1 (10 D this) (11 N lady))
              (12 ADVP (13 ADV wel))
              (14 E_S ,))
    (15 ID CMMALORY,2.12))
```

print_indices: (boolean true or false)

default false

tells CorpusSearch whether or not to print indices in the output.

Indices start at 0 and are used to label every node in the tree. CorpusSearch uses indices to distinguish, for instance, between several different NP nodes in the same output structure.

Here's a piece of output structure with indices:

```
             (10 NP-OB1 (11 NPR Morgan)
                        (12 NPR le)
                        (13 NPR Fay)
```

Here's how it looks without indices:

```
                  (NP-PRN (NPR Morgan)
                          (NPR le)
                          (NPR Fey)))
```

remove_nodes: (boolean true or false)

default false

removes subtrees whose root is of the same syntactic category as the node boundary embedded within a instance of that node boundary. "Remove_nodes" thus removes recursive structure. If the removed subtree matches the query, it will appear as a separate output token later in the output file. If the removed subtree does not contain the searched-for structure, it is discarded and replaced with a label indicating what has been removed.

The purpose of this feature is to make it easier to search output. For instance, if you were looking for IP nodes containing a certain structure, **remove_nodes** will ensure that your output contains only IP nodes with that structure, and no other IP nodes.

CorpusSearch uses the following algorithm to find the syntactic category of a node: Start with the node boundary label. If that label contains any hyphens, the node's syntactic category is the substring of the label up to the leftmost hyphen, with a '*' tacked on. If the node boundary label does not contain a hyphen, the syntactic category is simply the label with a '*' tacked on, unless the label already has one.

Thus, if the node boundary label is IP-PRN*, the node category is IP*.

Consider the following command file, in which **remove_nodes** is set to true, and its effect on the output below:

```
remove_nodes: true
query: (NP-OB* iDoms PRO)
```

Output:

```
/~*
'And I shall defende the,' seyde the knyght.
(CMMALORY,39.1264)
*~/

/*
 1 IP-MAT-SPE: 8 NP-OB1, 9 PRO the
*/

 (0 (1 IP-MAT-SPE (2 ' ')
                  (3 CONJ And)
                  (4 NP-SBJ (5 PRO I))
                  (6 MD shall)
                  (7 VB defende)
                  (8 NP-OB1 (9 PRO the))
                  (10 , ,)
                  (11 ' ')
                  (12 IP-MAT-PRN RMV:seyde_the_knyght...)
                  (13 E_S .))
     (ID CMMALORY,39.1264))
```

The structure of sub-sentence "seyde the knyght" has been removed from the parsed sentence and replaced with the symbol **RMV:<rmv_string>**, where ***rmv_string*** stands for a string of (up to) the first three words (leaf nodes) of the removed material and serves as a reminder of what has been removed. A further search on this output will be a search ***only*** on IP* nodes that contain a pronoun object, and on no other nodes.

Search specification

Every command file must contain a search specification, which instructs CorpusSearch as to what action to carry out. The search specification string must follow the preamble of search control commands and output format commands. The most common search specification, by far, is **query:**, used for searching a corpus.

ur_text_only: (boolean true or false)

default false

prints only the text of the tokens that match the query, suppressing printing of the labeled bracketing and associated information.

query: <query specification>

Queries must follow the syntax of the CorpusSearch query language. Every command files containing queries must bear the extension **.q**.

coding_query: <coding specification>

Coding query syntax is described in the chapter on coding. Every command file containing coding queries must bear the extension **.c**.

local_frames: <frame specification>

See the chapter on local frames for a description of this option.

make_lexicon: true

See the chapter on building a lexicon for a description of this option.

print_only: <pos_label string>

This option was designed for use on a file that contains coding strings produced by coding queries. To create an output file with only the coding strings use the following search specification:

```
print_only: CODING
```

The resultant output file will bear the extension **.ooo**. Please note that his feature does not work on output files of prior queries in which the nodes of the parse were indexed.

In theory, you could substitute a part of speech label for CODING, although if you wanted a list of, for instance, all the nouns in your file, you would probably be better off using the make_lexicon feature.

reformat_corpus: true

This takes as input a corpus file, and outputs the same file in the same format as CS would output from a search. This is useful, for instance, if you need to follow up with unix "diff" to compare search output with an original corpus file. The output file of "reformat_corpus" bears the extension **.fmt**.

copy_corpus: true

See the chapter on automated corpus revision for a description of this option.

Comments

Comments may be added anywhere to the command file or to files of parsed sentences that serve as input to CS. The program uses the following delimiters for such comments. Comment lines begin with "//" and block comments appear between "/*" and "*/". Unlike remarks, comments are not printed to the search output file.

For input files, but not for command files, you can also define custom comment delimiters. Add the following commands to the command file preamble or to the preferences file, followed by the desired delimiter strings.

For a line comment delimiter, add **corpus_line_comment:**

For block comment delimiters, add **corpus_comment_begin:** and **corpus_comment_end:**

Top of page
Table of Contents

Understanding CorpusSearch Output

Contents of this chapter:

Table of Contents

CorpusSearch Home

general form of the output

The extension of an output file will be **.out**. Output files have this general form:

1 per output file	1 per input file	1 per output sentence
Preface	Header	ur_text sentence
Summary	Footer	result block
		parsed sentence

Since the output file can become input to a subsequent search, everything except parsed sentences is surrounded by comment markers /* and */ (the ur_text block has slightly different markers).

a typical output file

As an example, we will walk through a typical output file. The query was designed to search for inverted pronoun subjects, that is, pronoun subjects that appear after the tensed verb.

To make this example easier to follow, the search was done with the default value (false) for "nodes_only."

We will discuss "nodes_only" and "remove_nodes" below.

preface

```
/*
    PREFACE:
    CorpusSearch copyright Beth Randall 2004.
    Date:  Sun Apr 30 07:05:51 EDT 2004

    command file:       under.q
    output file:        under.out

    remark:   this query searches for inverted pronoun subjects.

    node:   IP*
    print_indices: true
    query:  ((((NP*|ADJP*|ADVP*|PP* iPrecedes *MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|
*VBP|*VBD)
            AND (NP*|ADJP*|ADVP*|PP* iDominates !\*T*))
            AND (*MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD iPrecedes NP-SBJ*))
            AND (NP-SBJ* iDominates PRO|MAN))
*/
```

The preface begins with a copyright declaration and the date and time of the search.

The names of the command file and output file are listed. If this search had been performed using an output file as input (instead of a corpus file), the name of the output-as-input file would also have been listed in this block. But because the input file is a corpus file, the header and summary blocks contain all the necessary information (for more on searching output files, see below).

The remark was found in the command file. It serves as a reminder of the purpose of the query.

The beginning of the query,

```
          ((NP*|ADJP*|ADVP*|PP* iPrecedes  *MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|
*VBP|*VBD)
          AND (NP*|ADJP*|ADVP*|PP* iDominates !\*T*))
```

requires a constituent (NP*|ADJP*|ADVP*|PP*) which immediately precedes the tensed verb (*MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD). The constituent is required not to have a trace (*T*) (a placeholder for a word which would appear in that place under some circumstances, but in fact appears elsewhere in this particular sentence.) This requirement was put in to preclude questions (such as, "Kepte he his fadir scheep full mekly?"), where there is no constituent before the inverted pronoun subject other than the tensed verb. In Middle English, there must be one constituent before the tensed verb in statements, as the first two lines of the query describe.

The last two lines of the query,

```
          AND (*MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD iPrecedes NP-SBJ*))
          AND (NP-SBJ* iDominates PRO|MAN))
```

describe the tensed verb (*MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD) which precedes the subject noun phrase (NP-SBJ*), which itself immediately dominates a pronoun ("PRO|MAN") (that is, the subject is a pronoun.)

header

```
/*
    HEADER:
    source file:  cmcapchr.m4.psd
*/
```

Here, the source file is listed as its name appears in the corpus directory. If this had been an output file, the source file would have been listed as its name appears in the ID node of each sentence, that is, CMCAPCHR. (for more on searching output files, see below).

result block with output sentence

Here's an example of an output sentence, first presented as the original text, followed by the result block, which lists the nodes relevant to the query, followed by the sentence in its parsed form:

```
/~*
His fadir scheep kepte he ful mekly;
(CMCAPCHR,32.13)
*~/
```

```
/*
    1 IP-MAT: 2 NP-OB1, 7 VBD kepte, 6 N scheep, 8 NP-SBJ, 9 PRO he
*/

(0
   (1 IP-MAT
             (2 NP-OB1 (3 NP-POS (4 PRO$ His) (5 N$ fadir))
                       (6 N scheep))
             (7 VBD kepte)
             (8 NP-SBJ (9 PRO he))
             (10 ADVP (11 ADVR ful) (12 ADV mekly))
             (13 E_S ;))
   (ID CMCAPCHR,32.13))
```

The indices on the nodes of the labeled bracketing are intended to facilitate seeing how the token comes to match the query. They are present whenever the query preamble contains the line "print_indices: true."

Notice that the original text is surrounded by special markers, "/~*" and "*~/". When a search is run on the output file, CorpusSearch will find and record this block of data as the original text of the output sentence. In this way the entire original text is conserved, even when only bits and pieces of the original parse tree for the sentence appear in the output.

The first item in the result block is the boundary node (in this case, 1 IP-MAT), which match the value of the "node: " line of the command file. The boundary node is followed by a colon to separate it from the rest of the list, which gives the structures that correspond to the "query: " line of the command file. The list of indices and structures is structured so that no node is reported more than once.

For some queries, there may be many nodes that fit one search-function argument. In these cases CorpusSearch always reports the last legitimate fitting node. For instance, look at this part of the query:

```
(NP*|ADJP*|ADVP*|PP* iDominates !\*T*)
```

In the sentence above, (2 NP-OB1 iDominates the following nodes, where neither (3 NP-POS nor (6 N scheep is *T*:

```
                       (3 NP-POS (4 PRO$ His) (5 N$ fadir))
                       (6 N scheep)
```

so it is the last node, (6 N scheep), that is reported in the result block.

The parsed version of the output sentence is indented to show the structure of the tree. Sisters have the same indentation (for instance, 2 NP-OB1 and 7 VBD kepte.) Daughters are indented further than their mothers. If a node dominates only leaves, they are printed on the same line to save space.

footer

```
/*
FOOTER
  source file, hits/tokens/total
  cmcapchr.m4.psd       220/220/4175
*/
```

The footer gives the statistics for hits, tokens, and total as found in that input file. The same information appears again as one line of the summary block.

hits/tokens/total

CorpusSearch reports these statistics:

hits
number of distinct boundary nodes contaning the searched-for structure.
tokens
number of independent parsed objects in which hits occurred.
total
total number of independent parsed objects searched.

When you're searching a corpus file, most "tokens" are "matrix sentences", though some corpora have fragments and other material as independent tokens. In searches of corpus files it is very common to have "hits" greater than "tokens", since one matrix sentence may contain many distinct boundary nodes.

But suppose you follow these steps:

1. Run a search on the corpus with "nodes_only" and "remove_nodes" set to "true." Call the output of this search "1.out".
2. Now, run a second search on "1.out" with the same boundary node as used in the first search. Call the output of this second search "2.out".

In "2.out", "hits" and "tokens" will be the same number, because each token in "1.out" contained exactly one boundary node and thus can contain at most one hit in the second search.

summary block

```
/*
SUMMARY:
source files, hits/tokens/total:
  cmaelr4.m4.psd    46/46/766
  cmcapchr.m4.psd   220/220/4175
  cmcapser.m4.psd   12/12/91
  cmedmund.m4.psd   2/2/300
  cmfitzja.m4.psd   14/14/228
  cmgregor.m4.psd   14/14/2631
  cminnoce.m4.psd   6/6/208
  cmkempe.m4.psd    203/202/3851
  cmmalory.m4.psd   214/213/4995
  cmreynar.m4.psd   36/36/547
  cmreynes.m4.psd   0/0/245
  cmsiege.m4.psd    6/6/731
whole search, hits/tokens/total
                773/771/18772
*/
```

The summary block gives the same information as the footer blocks for each input file, but brought together in one place. This summary block was produced by a search on all files in the Middle English corpus (PPCME2) whose titles contain "m4", meaning they are from the fourth chronological period (1420 - 1500).

using nodes_only and remove_nodes

Consider this query file, called ipmat-2vb.q:

```
begin_remark:
    This query searches for matrix clauses which contain a
    subject and at least two verbs.  The subject precedes
    both verbs.
end_remark
```

```
nodes_only: t
remove_nodes: t
node:  IP-MAT*
query: (((((IP-MAT* iDoms NP-SBJ*)
AND (NP-SBJ* precedes *MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD))
AND (NP-SBJ* precedes VB|VAN|VBN|HV|HAN|HVN|DO|DAN|DON|BE|BEN))
AND (*MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD iDoms ![1]\**))
AND (VB|VAN|VBN|HV|HAN|HVN|DO|DAN|DON|BE|BEN iDoms ![2]\**))
```

Because remove_nodes and nodes_only are set to "true," the output will print only the boundary nodes containing the structure, and irrelevant boundary nodes will be removed. The purpose of these settings would be to ensure that subsequent searches are conducted only on the matrix clauses that contain a subject preceding two verbs. Here's a sample output sentence: in Modern English, this sentence would be: "He would have told you more if you had allowed him to."

```
/~*
and more he wolde a tolde you and $ye wolde a suffirde hym.
(CMMALORY,35.1106)
*~/
/*
 1 IP-MAT-SPE: 5 NP-SBJ, 7 MD wolde, 8 HV a
 1 IP-MAT-SPE: 5 NP-SBJ, 7 MD wolde, 9 VBN tolde
*/

(0 (1 IP-MAT-SPE (2 CONJ and)
                 (3 NP-OB1 (4 QR more))
                 (5 NP-SBJ (6 PRO he))
                 (7 MD wolde)
                 (8 HV a)
                 (9 VBN tolde)
                 (10 NP-OB2 (11 PRO you))
                 (12 PP (13 P and)
                        (14 CP-ADV (15 C 0)
                                   (IP-SUB RMV:$ye_wolde_a...)))
                 (24 E_S .))(ID CMMALORY,35.1106))
```

Notice that the IP-SUB clause, "$ye wold a suffirde hym", has been removed.

Suppose we run this output through a search for pronoun objects, using this query file, called "pro-obj.q":

```
begin_remark:
pronoun objects
end_remark

add_to_ignore: \**
query: (NP-OB* iDoms PRO)
```

The 35.1106 sentence shows up again, because it has a pronoun object "you":

```
/~*
and more he wolde a tolde you and $ye wolde a suffirde hym.
(CMMALORY,35.1106)
*~/
/*
 1 IP-MAT-SPE: 10 NP-OB2, 11 PRO you
*/

 (0 (1 IP-MAT-SPE (2 CONJ and)
                 (3 NP-OB1 (4 QR more))
                 (5 NP-SBJ (6 PRO he))
                 (7 MD wolde)
                 (8 HV a)
                 (9 VBN tolde)
                 (10 NP-OB2 (11 PRO you))
                 (12 PP (13 P and)
                        (14 CP-ADV (15 C 0)
                                   (16 IP-SUB RMV:$ye_wolde_a...)))
                 (17 E_S .))(ID CMMALORY,35.1106))
```

Notice that the results block describes one structure,

```
1 IP-MAT-SPE: 10 NP-OB2, 11 PRO you
```

This structure will be counted as one hit in the final summary block.

Now suppose we run the same series of searches, but this time we change "nodes_only" to "false."

```
nodes_only: f
```

When "nodes_only" is false, "remove_nodes" is automatically false.

Here's how the 35.1106 sentence looks after running ipmat-2vb.q with nodes_only and remove_nodes false:

```
/~*
and more he wolde a tolde you and $ye wolde a suffirde hym.
(CMMALORY,35.1106)
*~/
/*
 1 IP-MAT-SPE: 5 NP-SBJ, 7 MD wolde, 8 HV a
 1 IP-MAT-SPE: 5 NP-SBJ, 7 MD wolde, 9 VBN tolde
*/

(0
(1 IP-MAT-SPE (2 CONJ and)
              (3 NP-OB1 (4 QR more))
              (5 NP-SBJ (6 PRO he))
              (7 MD wolde)
              (8 HV a)
              (9 VBN tolde)
              (10 NP-OB2 (11 PRO you))
              (12 PP (13 P and)
                     (14 CP-ADV (15 C 0)
                                (16 IP-SUB
                                           (17 NP-SBJ (18 PRO $ye))
                                           (19 MD wolde)
                                           (20 HV a)
                                           (21 VBN suffirde)
                                           (22 NP-OB1 (23 PRO hym)))))
              (24 E_S .))
(25 ID CMMALORY,35.1106))
```

Notice that the clause "$ye wolde a suffirde hym" is printed out in full.

Now we run pro-obj.q on this output. Here's the 35.1106 sentence in the output of this search:

```
/~*
and more he wolde a tolde you and $ye wolde a suffirde hym.
(CMMALORY,35.1106)
*~/
```

```
/*
 1 IP-MAT-SPE: 10 NP-OB2, 11 PRO you
 16 IP-SUB: 22 NP-OB1, 23 PRO hym
*/

(0
(1 IP-MAT-SPE (2 CONJ and)
              (3 NP-OB1 (4 QR more))
              (5 NP-SBJ (6 PRO he))
              (7 MD wolde)
              (8 HV a)
              (9 VBN tolde)
              (10 NP-OB2 (11 PRO you))
              (12 PP (13 P and)
                     (14 CP-ADV (15 C 0)
                                (16 IP-SUB
                                           (17 NP-SBJ (18 PRO $ye))
                                           (19 MD wolde)
                                           (20 HV a)
                                           (21 VBN suffirde)
                                           (22 NP-OB1 (23 PRO hym)))))
              (24 E_S .))
(25 ID CMMALORY,35.1106))
```

Notice that here the results block contains two different structures,

```
1 IP-MAT-SPE: 10 NP-OB2, 11 PRO you
16 IP-SUB: 22 NP-OB1, 23 PRO hym
```

The structure

```
16 IP-SUB: 22 NP-OB1, 23 PRO hym
```

is reported in this case because remove_nodes was false in the previous search. The pronoun object "hym" was found in a subordinate clause, not the matrix clause that was of interest to the last search.

Because the structures occur in two distinct boundary nodes (1 IP-MAT-SPE and 16 IP-SUB), this will count as two hits in the summary block, in contrast to the one hit that was found when remove_nodes was true. This explains why, after using "remove_nodes: true" in the initial search, a second search for pronoun objects finds fewer hits than when the initial search was conducted with "remove_nodes: false."

Here's the summary block from the "remove_nodes: true" version:

```
/*
SUMMARY:
source files, hits/tokens/total:
  CMMALORY      177/176/875
whole search, hits/tokens/total
                177/176/875
*/
```

And here's the summary block from the "remove_nodes: false" version:

```
/*
SUMMARY:
source files, hits/tokens/total:
  CMMALORY      290/249/875
whole search, hits/tokens/total
                290/249/875
```

```
*/
```

Top of page
Table of Contents

Searching for Words and Labels

Contents of this chapter:

Table of Contents

CorpusSearch Home

words and labels

"Labels" are the tags inserted by the annotators who prepared the corpus (e.g., "IP", "CONJ", "N".) "Words" are the original words of the text that was parsed. Every node in the sentence-tree has a label, and in the leaf nodes the label is paired with a word. CorpusSearch can conduct searches on labels or words or combinations of the two.

string variations

CorpusSearch uses case-sensitive character-by-character string matching to match search-function arguments to strings found in the input. Therefore, spelling and upper-case/lower-case variations must be described explicitly (usually with an argument list.) For instance, this query searches for a complementizer whose associated text is "that" or "That":

```
(C iDominates that|That)
```

and finds sentences such as this:

```
/~*
and he shalle do yow remedy, that youre herte shal be pleasyd. '
(CMMALORY,3.47)
*~/

/*
    12 CP-ADV: 13 C that
*/

(
      (12 CP-ADV (13 C that)
                 (14 IP-SUB
                            (15 NP-SBJ (16 PRO$ youre) (17 N herte))
                            (18 MD shal)
                            (19 BE be)
                            (20 VAN pleasyd)))
      (ID CMMALORY,3.47))
```

Search Tips

Contents of this chapter:

Table of Contents

CorpusSearch Home

about this chapter

This chapter gives tips on a number of common problems and errors that arise when using CorpusSearch. The reader is assumed to have a general familiarity with the rest of the CorpusSearch manual. Many of the example queries assume a standard definition file containing definitions for at least finite_verb and non_finite_verb.

using definition files

The following are useful definitions to include in a definition file for the Middle and Early Modern English corpora:

```
finite_verb:  *MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD
non_finite_verb:  *VB|V*N|*HV|H*N|*DO|D*N|*BE|BEN
non-pronominal_NP: *N*|D*|Q*|ADJ*|CONJ*|*ONE*|*OTHER*|CP*
```

A common error is to forget to use the `define` command to specify the definition file when using definitions. No error message will be generated, but the search will result in no output.

using *

Be liberal in using *. Using `NP-SBJ` as a search term will only find a subset of subjects. Some subjects are resumptive (`NP-SBJ-RSP`), some are coindexed to a clause, or to trace in a lower clause (`NP-SBJ-1`), some may have other additional labels. Using `NP-SBJ*` will find all the subjects labelled in this way, no matter what might be added on to the end of the label. In general, only leave off the * if you are sure you **don't** want it.

When you want to refer to all the labels referred to by, for instance, `ADVP*`, except one, you have to use a list and list all the options you are interested in, as for instance `ADVP|ADVP-LOC|ADVP-TMP` (this omits `ADVP-DIR` which would be included in `ADVP*`). This is what definition files are for; you only have to write the complex disjunction once.

Note that if you want to refer to an actual * in a search (all traces start with *), escape it with a backslash \ . The following query finds subjects which dominate traces. The first * in `\**` is escaped and thus refers to an actual *, while the second is not and thus matches anything that follows the *; this will match, for instance, `*con*`, `*exp*`, `*T-1*` and others.

```
query: (NP-SBJ* iDoms \**)
```

the "exists" function

A common error is to overuse the `exists` function. Using a search term forces that term to exist in any hit; it is not necessary to specify this separately. Thus the following is an inefficient query, although it is not ill-formed.

```
query: ((NP-SBJ* exists)
AND (IP* iDoms NP-SBJ*))
```

The second clause of the query alone will accomplish the same thing more quickly and efficiently.

same instance

Same instance works by literal string match of arguments to functions. Thus `NP-SBJ` does **not** match `NP-SBJ*`, and `MD|VBD` does not match `VBD|MD`; that is, in neither case would same instance be invoked between the two terms.

When two search arguments do match, they are forced to apply to the same node. Thus two uses of `NP-OB*` will require that, if for instance, `NP-OB2` is found as an instance of the first `NP-OB*`, then the next use of NP-OB* will also apply to the **same** `NP-OB2` (not, for instance, an `NP-OB1` which may also be in the vicinity).

When two search terms do not match but might refer to the same node, as for instance, `NP-SBJ` and `NP-SBJ*`, or `MD|VBD` and `VBD|MD`, same instance is not forced, but neither is it ruled out; that is, the two label strings in the query may or may not wind up referring to the same node in the corpus.

In order to force non-same instance, use index numbers. `[1]NP-SBJ*` and `[2]NP-SBJ*` **cannot** apply to the same `NP-SBJ*` node.

A common error is to forget that impossible (to the linguist) cases of same instance will nonetheless be interpreted this way by CorpusSearch. Thus, for instance, a query such as the following will produce no results:

```
query: ((NP-SBJ* iDoms PRO)
AND (NP-OB1* iDoms PRO))
```

Although it is impossible for these `PRO`s to refer to the same node, since they are dominated by different nodes, CorpusSearch will assume they do, and consequently will find no matches. Traces and zeros also need to be differentiated, as in the following:

```
query: ((MD iDoms [1]!\**)
AND (VB iDoms [2]!\**))
```

or

```
query: ((WNP iDoms [1]0)
AND (C iDoms [2]0))
```

An easier way to accomplish the former is to add traces to the ignore list.

ignoring certain nodes

A default "ignore list" is supplied with CorpusSearch. It contains such things as punctuation and various meta labels that are not part of the text. If you want to search for punctuation, for instance, or line breaks, then you must provide your own ignore list which does not include the items you want to be able to access.

Although the ignore list is primarily a way to avoid non-text annotations, linguistic labels can also be added to the ignore list, in which case CorpusSearch will simply act as if they are not there. Thus for instance, if you add `NEG` to the ignore list, you can find cases in which nothing but possibly negation intervenes between the subject and the finite verb.

```
add_to_ignore: NEG
query: (NP-SBJ* iPrecedes finite_verb)
```

This will find the following two sentences:

Arthur loves Guinevere
Arthur ne loves Guinevere

but not:

Arthur madly loves Guinevere

Using the ignore list is also helpful in looking for V2. In many cases, the verb is not technically the second node in the IP because of initial conjunction. Adding `CONJ` (and possibly some other things, such as `INTJ*`, and `NP-VOC`) to the ignore list will solve this problem (or at least reduce it). The query below will find all the following:

The sword desired Lancelot
And the sword desired Lancelot
Gramercy, Arthur, the sword desired Lancelot

```
add_to_ignore: INTJ*|NP-VOC|CONJ
query: ((IP* iDomsNumber 1 NP-OB*)
AND (IP* iDomsNumber 2 finite_verb))
```

searching for traces

Traces (which all start with * in the PPCME2) are treated as text by CorpusSearch, and thus can be searched for. In order to differentiate the * which means "match anything" from the * that is part of the text of a trace, use `\*` to refer to the latter. The string `\**` will match any trace.

In the more common case, in which you want to simply ignore traces, add them to the ignore list as follows:

```
add_to_ignore: \**
```

This means that any node that contains a trace will not be found. Thus a query such as `(NP* exists)` will not find any NPs which contain only traces.

finding non-pronominal NPs

Do not search for non-pronominal NPs with the following query:

```
(NP* iDoms !PRO)
```

This will also eliminate cases like ***Robin and me*** and ***he and I***, since these contain a `PRO`. Instead use the non-pronominal_NP definition.

restricting searches to a single IP

CorpusSearch requires that you specify a node boundary within which to search. The node boundary includes **everything** dominated by the node, no matter how deeply embedded. Thus, if `IP*` is specified as the node boundary and an `IP` contains a subordinate clause `IP`, the contents of the embedded subordinate clause are also within the node. A common error is to write a query such as

```
query: ((IP* iDomsNumber1 NP-OB*)
AND (finite_verb iPrecedes NP-SBJ*))
```

with the intent of finding V2 clauses with a topicalized object. The first function looks for `IP`s which have an object as the first element; the second for a finite verb immediately preceding the subject. This query will, in fact, find V2 clauses with a topicalized object, but it may also find some other clauses as well. It will find (if there are any) `IP`s which contain one clause in which the first element is an object, and another different clause within the same node boundary in which the finite verb precedes the subject. Either, one of these clauses may be the main clause and the other a embedded clause, or, they may both be embedded `IP`s within a dominating `IP`.

There are two ways to avoid this error and force all parts of the query to apply within the same IP.

1. Make use of the built-in same instance feature. Same instance means that if you use a node label in the query more than once in exactly the same form, CorpusSearch assumes that you intend each use to apply to the same instance of that node. Same instance applies across query clauses conjoined by AND. You can use same instance to keep all the queries inside the same `IP` (for instance, or any other node) by "tying" one term of the query to the node, as in the first element of the query above, and then making sure that in every subsequent search function, either that "tied" term or the node is used. For instance, we could fix the query above, by writing it as:

   ```
   query: (((IP* iDomsNumber1 NP-OB*)
   AND (NP-OB* iPrecedes finite_verb))
   AND (finite_verb  iPrecedes NP-SBJ*))
   ```

   or alternatively:

   ```
   query: (((IP* iDomsNumber1 NP-OB*)
   AND (NP-OB* iPrecedes finite_verb))
   AND (finite_verb iPrecedes NP-SBJ*))
   ```

   The repeated instances of `NP-OB*` in the first example and `IP*` in the second refer to the same instance of `NP-OB*` and `IP*` respectively, thus forcing all parts of the query to be immediately dominated by the node.

2. The second solution is to use the `remove_nodes` output format option. The default setting for `remove_nodes` is false, so to activate it you must include the line `remove_nodes: t` in the query file preamble. Removing nodes removes any embedded structure whose root matches the specified boundary node. When `IP*` is specified, all embedded `IP`s will be removed. If the boundary node is set as `NP*`, all NPs embedded within another NP will be removed. **Note that all that is required for a match is that the syntactic category of the label (the part before the hyphen) matches**. Thus, if the node is `IP-MAT*`, any node whose label starts with `IP` will be removed, including in this case, `IP-SUB`, `IP-SMC`, `IP-PPL`, etc. Thus, for instance, you cannot set the boundary node as `IP-MAT*` and **not** have `IP-SUB`s removed. When "remove nodes" is in force, any node that doesn't match the query is removed completely from the output; any embedded node that matches the query is removed from its matrix and printed below it.

   To solve our problem the "remove nodes" way, we would first create a file with only single clauses with all embedded nodes removed, by a query such as

   ```
   remove_nodes: t
   query: (IP* iDoms finite_verb)
   ```

   This query will produce a file in which every token is an IP containing a finite verb with all embedded IPs removed. The following query:

```
query: ((IP* iDomsNumber1 NP-OB*)
AND (finite_verb iPrecedes NP-SBJ*))
```

can then be used on the output of the first query and will yield only the cases intended. (But note that this query is not actually going to produce all V2 clauses with a topic object anyway, since many such clauses begin with a conjunction or other introductory type word and thus the object will be the second element in the `IP*`; for a solution to this problem, see ignoring certain nodes).

counting words and remove_nodes

Note that if you have remove_nodes turned on, the string RMV:<rmv_string>, counts as text so you can search for it. It will not, however, be counted as a word when doing word counts (like traces, which likewise are not counted). But, if you count the number of words in a node that contains RMV:<rmv_string>, you will, of course, get the wrong answer, since RMV:<rmv_string> replaces a clause full of words. In order to avoid this result, either don't use remove_nodes when counting, or, use a query like the following which won't count any node containing RMV:<rmv_string>. Nodes containing RMV:<rmv_string> can then be counted separately.

```
query: (((IP* iDoms NP-OB*)
AND (NP-OB* domsWords3))
AND (NP-OB* doms !RMV:*))
```

Another way to do this is to add `RMV:*` to the ignore list and then, as before, count the nodes containing `RMV:*` separately.

```
add_to_ignore: RMV:*
query: ((IP* iDoms NP-OB*)
AND (NP-OB* domsWords3))
```

Top of page
Table of Contents

Using Definition Files

Contents of this chapter:

Table of Contents
CorpusSearch Home

What are definition files for?

Definition files are an optional convenience for the CorpusSearch user. If you find yourself writing the same long argument list for many different queries, you would probably benefit from having a definition file. A definition file allows you to assign a label to a list of arguments. When you write a query, you can refer to this label instead of writing out the argument list.

definition file example

Here's an example of a definition file written by Ann Taylor. The name of the file is "Ann.def". Definition file names must always have the extension **.def**.

```
// definition file written by Ann Taylor.

non_finite_verb:  *VB|V*N|*HV|H*N|*DO|D*N|*BE|BEN

finite_verb:  *MD|*HVP|*HVD|*DOP|*DOD|*BEP|*BED|*VBP|*VBD
```

query file example

Here's an example of a query file that refers to the definition file:

```
define: Ann.def

query:  (finite_verb precedes non_finite_verb)
```

The first line, "define: Ann.def", tells CorpusSearch where to find the definitions of "finite_verb" and "non_finite_verb". Without this line, "finite_verb" and "non_finite_verb" will not be replaced by their definitions.

recursive definitions

Terms may be defined recursively, that is, one term may be a list of other terms. To make a recursive reference use the dollar sign "$". Here's an example of a recursive definition file:

```
first:  there's|a|somebody|I'm
second: longing|to|see
third: I|hope|that|he
fourth: turns|out|to|be
fifth: someone|to|watch|over|me

first2: $first|$second
next3: $third|$fourth|$fifth
```

```
     whole: $first2|$next3
```

Notice that the one term "whole" refers to the entire stanza.

output file example

Here's the preface from an output file resulting from "def.q".

```
     /*
         PREFACE:
         CorpusSearch copyright Beth Randall 2000.
         Date:  Thu Apr 13 08:57:07 EDT 2000

         command file:       def.q
         output file:        def.out

         definition file:  Ann.def
         node:   IP*
         query:  (*MD|*HVP|*DOP|*DOD|*BEP|*BED|*VBP|*VBD precedes
                  *VB|V*N|*HV|H*N|*DO|D*N|*BE|BEN)
     */
```

Notice that the query appears in two forms: first in the shorthand form found in the command file, and second in its expanded form after the installation of the definitions. The second form of the query is the one that CorpusSearch uses for the search. If you still see your definition labels in the second form of the query, it means that the definitions were not installed. You may have forgotten to put "define: <def.file>" in your command file, or perhaps misspelled the labels.

Top of page
Table of Contents

Using Preference Files

Contents of this chapter:

Table of Contents
CorpusSearch Home

What are preference files for?

Preference files are a way for the CorpusSearch user to set custom default values. If you find yourself continually copying the same information into your query files, you would most likely benefit from a preference file.

CorpusSearch sets its own default values for certain variables but if a preference file exists, the commands contained in it override the defaults set within CorpusSearch. If the current query file contains commands of the same type as the preference file, the commands in the query file override those in the preference file. So, for instance, CorpusSearch sets the command line comment delimiter "//", your preference file might set it to "/!/", and your query file might set it to "/>". It is the last value that CorpusSearch will use as the comment delimiter.

a preference file example

Here's an example of a basic preference file, designed for searching the Penn Korean TreeBank (Han, Han, and Ko (2001)). The name of the file is "korean.prf". Preference file names must always have the extension **.prf** and must be stored in the same directory as the query file.

```
// a preference file for the Korean corpus.
corpus_file_extension: fid
corpus_comment_begin: <
corpus_comment_end: >
corpus_line_comment: ;;
node: S
```

an output file example

Here's the preface from an output file using "korean.prf". Notice the line

```
preference file:  korean.prf
```

This line is how you know that CorpusSearch accessed your preference file. If you don't see this line, your preference file was not found. Check that your preference file name ends with ".prf", and that it's stored in the same directory as your query file.

```
/*
PREFACE:
CorpusSearch copyright Beth Randall 2000.
Date:  Mon Feb 26 09:28:55 EST 2001

command file:     kor.q
preference file:  korean.prf
input file:       add.out
output file:      kor.out
```

```
node:   S
query:  (NNC iPrecedes NNC)
*/
```

Top of page
Table of Contents

Coding

Contents of this chapter:

Table of Contents

CorpusSearch Home

What is coding?

Coding is used for creating input to multivariate analysis programs like Varbrul; general statistical programming environments like S, Splus, and R; and statistical analysis packages like Datadesk, JMP, SAS, and SPSS.

Coding string values in a coding file may be in part automatically determined with coding queries and in part hand entered in a text editor. The resultant files can then be inputs to further searches.

a coding file example

Here's an example of a basic coding file, called "obj.c". All coding file names must end with ".c". To simplify our discussion, we show only the first four columns of an originally more complicated coding system.

```
node: IP*
coding_query:

1: {
        s: (IP*SPE* iDoms NP-OB*)
        n: ELSE
   }

2: {
        m: (IP-MAT* iDoms NP-OB*)
        s: (IP-SUB* iDoms NP-OB*)
        i: (IP-INF* iDoms NP-OB*)
        e: ELSE

   }

3: {
        t: ((IP* iDoms NEG)
          AND (NEG iDoms !ne))
        p: (IP* iDoms !NEG)
        n: ELSE
   }
4: {
        \1: (NP-OB* domsWords 1)
        \2: (NP-OB* domsWords 2)
        \3: (NP-OB* domsWords> 2)
        \0: ELSE
   }
```

In general, coding files have this form:

```
<PREAMBLE>
coding_query:

column_number: {
        label: condition
        label: condition
        .
        .
        .
        }
```

The coding file begins with the preamble commands (see Command File chapter), which must include the obligatory bounding node for the coding queries. The obligatory query specification "coding_query:" then introduces the coding queries for each column of the output coding string.

In the present example, column 1 of the coding string will contain an "s" if `IP*SPE* iDoms NP-OB*`. Everywhere else, due to the presence of the "ELSE" function (used only in coding queries), the column will contain an "n".

Note that when numerals (0-9) are used as codes, they must be introduced with the backslash character ("\"), as illustrated in column 4 above.

Coding query files are alternatives to ordinary query files in a CorpusSearch run. So, to code a file, invoke CorpusSearch as follows:

```
java CorpusSearch <coding_file.c> <file_to_code>
```

an output file example

Output files resulting from coding will carry the extension **.cod**. They contain every token of the input file, with coding nodes inserted at every boundary node. A coding node has the form:

```
(CODING <coding_string>)
```

If a given sentence contains more than one boundary node, the output sentence will contain multiple coding nodes. Here's a sentence from the output file resulting from the above coding file:

```
/~*
knewe kyndes & complexciones of men & of bestus
(CMHORSES,85.2)
*~/


( (IP-SUB (CODING n:s:p)
          (NP-SBJ *T*-1)
          (VBD knewe)
          (NP-OB1 (NS kyndes)
                  (CONJ &)
                  (NS complexciones)
                     (PP
                     (PP (P of)
                         (NP (NS men)))
                     (CONJP (CONJ &)
                     (PP    (P of)
                            (NP (NS bestus)))))))
   (ID CMHORSES,85.2))
```

how to search coding strings

Coding strings may be searched using column. For instance, to find all boundary nodes whose coding string contains "m" or "p" in the 7th column, use this query:

```
query:  (CODING column 7 m|p)
```

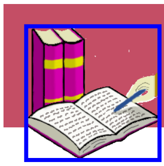
just the codes

To obtain a file with only the coding strings, use print_only as follows:

```
print_only: CODING
```

The extension of the resultant output file will be **.ooo**.

Top of page
Table of Contents



Lexicon Building

◀ ▶

Contents of this chapter:

Table of Contents
CorpusSearch Home

What is a lexicon?

A lexicon is a list of the words used in an input file or files. Following each word is the number of times the word was found, followed by the part-of-speech labels associated with the word and the number of times each part of speech was found. Word identity is determined by spelling. No morphological analysis or spelling normalization is performed. However, spellings that vary only by capitalizations are listed on the same line. Also, initial "$" is ignored.

In the following example, this line:

```
a-boute 11: [9 P] [1 RP] [1 ADV]
```

means that the word "a-boute" was found 11 times, 9 times with the part of speech label "P", 1 time with the part of speech label "RP", and 1 time with the part of speech label "ADV".

make_lexicon

This is the basic command that causes a lexicon to be built. On its own, the following will generate a lexicon of every word in the input file(s).

```
     make_lexicon: t
```

pos_labels

This command restricts the lexicon to words with certain part of speech tags. For instance, to obtain a list of words labelled as prepositions:

```
     make_lexicon: t
     pos_labels: P|P#
```

text_labels

This command restricts the lexicon to certain words. For instance, to find only words beginning with "th" or "+t":

```
     make_lexicon: t
     text_labels: th*|+t*|Th*|+T*
```

Both pos_labels and text_labels can be specified in one query. For instance, to obtain prepositions beginning "in":

```
     make_lexicon: t
     pos_labels: P|P#
```

```
    text_labels: in*
```

an example

The following query:

```
make_lexicon: t
```

results in this output:

```
/*
PREFACE:
CorpusSearch copyright Beth Randall 2000.
Date:  Tue Sep 21 09:55:12 EDT 2004

command file:     test/lex.q
output file:      test/lex.out

Lexicon:
*/

/*  ~A~  */
a A $a 3713: [3421 D] [10 FW] [104 HV] [15 VAN21] [24 ADV21] [25 P21] [8 VBD21]
[15 P] [1 RP21] [1 N21] [4 CONJ] [5 VB21] [6 N] [4 ADJ21] [68 INTJ] [1 VBN21]
[1 NUM21]
a+gen 15: [9 ADV] [6 P]
a+gennyst 1: [1 P]
a+gens 4: [4 P]
a+genst 2: [2 P]
a+geyne 10: [10 ADV]
a-+gen 63: [52 ADV] [11 P]
a-+gens 12: [12 P]
a-bak 1: [1 P+ADV]
a-bakke 1: [1 P+ADV]
a-baschyd 2: [2 VAN]
a-basshed 1: [1 VAN]
a-basshyd 1: [1 VAN]
a-beyn 1: [1 VB]
a-bod 1: [1 VBD]
a-bode 5: [5 VBD]
a-bood 4: [4 VBD]
a-boode 1: [1 VBD]
a-bouen 1: [1 P]
a-boute 11: [9 P] [1 RP] [1 ADV]
.
.
.
.
.
.
.
/*  ~Z~  */
zacari 1: [1 NPR]
zacharie 1: [1 NPR]
zaram 1: [1 NPR]
zebede 1: [1 NPR]
zelator 1: [1 N]
zelatoris 1: [1 NS]
zele 6: [6 N]
zelose 2: [2 ADJ]
zelously 1: [1 ADV]
```

zeno 1: [1 NPR]
zenocrates 1: [1 NPR]
zenon 1: [1 NPR]
zepherine 1: [1 NPR]
zorobabel Zorobabel 3: [3 NPR]
zorobabell Zorobabell 4: [4 NPR]
zozime 1: [1 NPR]

Top of page
Table of Contents

Local Frames

◀ ▶

Contents of this chapter:

Table of Contents
CorpusSearch Home

What are local frames?

Local frames are an approximation to subcategorization and selectional contexts for lexical heads. The user specifies a head whose contexts are to be found and displayed. This head consists of a part-of-speech category and a lexical item of that category. CorpusSearch then returns a list of the contexts in which the specified POS,word pair occurs in the corpus.

Each context is a string of sister nodes of the POS label(s) specified in the query. The list of contexts is organized into subgroups that share the same "kernel," where a kernel is the subset of the sister nodes whose phrasal category is NP*; that is, subjects and objects. If a PP occurs in a local frame, its head preposition is given immediately following the PP label in the output.

This functionality is under development. Use it with care. Suggestions for improvement are welcome.

an example

The following query:

```
node: IP*

local_frames: (VB* over +tank*|thank*)
```

results in this output (only the beginning is shown):

```
/*
PREFACE:
CorpusSearch copyright Beth Randall 2000.
Date:  Tue Sep 21 10:08:43 EDT 2004

command file:     test/frames.q
output file:      test/frames.out

local frames:  (VB* over  +tank*|thank*)
*/

/*
NP-SBJ VB thankyn NP-OB2
*/

ALSO NP-SBJ MD VB NP-OB2 (ID CMKEMPE,57.1270)
CONJ NP-SBJ MD TO VB NP-OB2 PP for (ID CMKEMPE,46.1024)

/*
NP-SBJ VBD +tankyd NP-OB2
*/
```

```
NP-SBJ VBD NP-OB2 PP for (ID CMKEMPE,58.1308)
VBD NP-OB2 ADVP PP-RSP +terfore (ID CMKEMPE,78.1765)

/*
NP-SBJ VBD thankyd NP-OB2
*/

ADVP-TMP NP-SBJ IP-PPL VBD NP-OB2 PP as IP-PPL (ID CMKEMPE,224.3620)
ADVP-TMP NP-SBJ IP-PPL VBD NP-OB2 PP of (ID CMKEMPE,133.3113)
ADVP-TMP NP-SBJ VBD NP-OB2 PP of (ID CMKEMPE,25.533)
CONJ ADVP-TMP NP-SBJ PP wyth VBD NP-OB2 PP of IP-PPL (ID CMKEMPE,87.1978)
CONJ ADVP-TMP NP-SBJ VBD NP-OB2 ADVP PP of IP-PPL (ID CMKEMPE,94.2146)
CONJ ADVP-TMP NP-SBJ VBD NP-OB2 PP of IP-PPL (ID CMKEMPE,13.255)
```

Top of page
Table of Contents

Automated Corpus Revision

Contents of this chapter:

Table of Contents

CorpusSearch Home

revision feature

CorpusSearch 2 has a corpus-revision feature, which allows the user to make automatic changes to a corpus. This is useful, for instance, in correcting parser errors, or revising a corpus to fit new annotation guidelines.

Revisions are linked to a standard CS query, which is decorated with curly-bracket tags indicating where revisions should take place. The curly brackets contain an index which correlates an argument in the query to a revision instruction. I'll call the curly-bracket construction a "flag". This is the general idea:

```
     query: ({x}A function B) AND (C function {y}D)

     revise{x}: info
     revise{y}: info
```

Also see the examples.

don't repeat flags

Suppose you have a query where the same node is mentioned several times. You may be tempted to flag the node every time it appears in the query, as below:

```
WRONG!
query: (NP* iDoms {1}[1]Q)
        AND (NP* iDoms {2}[2]Q)
        AND ({1}[1]Q iPrecedes {2}[2]Q)
add_internal_node{1, 2}: QP
```

The problem with this is that CorpusSearch only needs to have the arguments flagged once, and repeating the flags just increases the possibility of error (for instance, the same flag might wind up referring to two different nodes). For this reason, CorpusSearch ignores repeated flags, and issues a warning when they are encountered. The above query produces these WARNING messages:

```
WARNING!  Subsequent flag {1} has been ignored.

WARNING!  Subsequent flag {2} has been ignored.
```

This version of the query is preferred:

```
query: (NP* iDoms {1}[1]Q)
       AND (NP* iDoms {2}[2]Q)
       AND ([1]Q iPrecedes [2]Q)
add_internal_node{1, 2}: QP
```

label changes

The simplest way to change a tree is to change labels, leaving the structure intact. CS has the following label-changing revision functions:

replace_label

    replace_label{x}: new_label

append_label

    append_label{x}: label_to_append

prepend_label

    prepend_label{x}: label_to_prepend

post_crop_label

    post_crop_label{x}: label_to_crop

pre_crop_label

    pre_crop_label{x}: label_to_crop

replace_label

This query:

```
    node: IP*
    query:  ({1}NP-ACC iDoms N*)

    replace_label{1}: BULLWINKLE
```

applied to this input:

```
    ( (IP-MAT (NP-SBJ (PRO You))
              (MD must)
              (NEG not)
              (VB exspecte)
              (NP-ACC (Q no) (ADJ greate) (NS matters))
              (NP-TMP (D this) (N time))
              (. ,)) (ID KNYVETT-1630,87.25))
```

produces this output:

```
    ( (IP-MAT (NP-SBJ (PRO You))
              (MD must)
```

```
                (NEG not)
                (VB exspecte)
                (BULLWINKLE (Q no) (ADJ greate) (NS matters))
                (NP-TMP (D this) (N time))
                (. ,))
        (ID KNYVETT-1630,87.25))
```

append_label

This appends the given label to the flagged argument. This query:

```
node: $ROOT

query: ({1}WPRO iDoms what|What) AND (WPRO iPrecedes IP*)

append_label{1}: -THAT
```

applied to this input:

```
( (IP-MAT (CONJ but)
          (CP-QUE (WNP-1 (WPRO what))
                  (IP-SUB (NP-TMP *T*-1)
                          (NP-SBJ (PRO I))
                          (MD shall)
                          (VB returne)
                          (NP-DIR (N home))))
          (NP-SBJ (PRO I))
          (BEP am)
          (ADJP (NP-MSR (D a) (Q little))
                (ADJ doubtfull))
          (. .)) (ID KNYVETT-1630,94.268))
```

produces this output:

```
/~*
but what I shall returne home I am a little doubtfull.
(KNYVETT-1630,94.268)
*~/
/*
1 IP-MAT:  6 WPRO, 7 what, 8 IP-SUB
*/


( (IP-MAT (CONJ but)
          (CP-QUE (WNP-1 (WPRO-THAT what))
                  (IP-SUB (NP-TMP *T*-1)
                          (NP-SBJ (PRO I))
                          (MD shall)
                          (VB returne)
                          (NP-DIR (N home))))
          (NP-SBJ (PRO I))
          (BEP am)
          (ADJP (NP-MSR (D a) (Q little))
                (ADJ doubtfull))
          (. .))
  (ID KNYVETT-1630,94.268))
```

prepend_label

This prepends the given label to the flagged argument. This query:

```
node: $ROOT
ignore_nodes: null
query: ({1}[1], iDoms [2],) AND ([1], iPres *-PRN)
       AND (*-PRN iPres [3],) AND ({2}[3], iDoms [4],)

prepend_label{1}: PRN-
prepend_label{2}: PRN-
```

applied to this input:

```
( (IP-MAT (CONJ &)
          (NP-SBJ (PRO$ my) (NS horsses))
          (, ,)
          (IP-MAT-PRN (NP-SBJ (PRO I))
                      (VBP thinke))
          (, ,)
          (MD $wil)
          (BE $be)
          (CODE {TEXT:wilbe})
          (VBN gone)
          (PP (P to)
              (NP (N morrowe)))
          (. ,)) (ID KNYVETT-1630,93.228))
```

produces this output:

```
/~*
& my horsses, I thinke, $wil $be gone to morrowe,
(KNYVETT-1630,93.228)
*~/
/*
1 IP-MAT:  9 ,, 10 ,, 11 IP-MAT-PRN, 17 ,, 18 ,
*/


( (IP-MAT (CONJ &)
          (NP-SBJ (PRO$ my) (NS horsses))
          (PRN-, ,)
          (IP-MAT-PRN (NP-SBJ (PRO I))
                      (VBP thinke))
          (PRN-, ,)
          (MD $wil)
          (BE $be)
          (CODE {TEXT:wilbe})
          (VBN gone)
          (PP (P to)
              (NP (N morrowe)))
          (. ,))
  (ID KNYVETT-1630,93.228))
```

pre_crop_label

This crops the label ending at the given character. This query:

```
node: $ROOT

query: (ADVP* iDoms {1}ADV+*)

pre_crop_label{1}: +
```

applied to this input:

```
( (IP-MAT (CONJ &)
          (NP-SBJ (Q many))
          (VBD lost)
          (NP-ACC (PRO$ ther) (NS lifes))
          (PP (PP (P aboute)
                  (NP (D the) (NS Teames)))
              (CONJP (CONJ &)
                     (ADVP-LOC (ADV+WADV elsewher))))
          (. .)) (ID KNYVETT-1630,87.21))
```

results in this output:

```
/~*
& many lost ther lifes aboute the Teames & elsewher.
(KNYVETT-1630,87.21)
*~/
/*
1 IP-MAT:  26 ADVP-LOC, 27 ADV+WADV
*/

( (IP-MAT (CONJ &)
          (NP-SBJ (Q many))
          (VBD lost)
          (NP-ACC (PRO$ ther) (NS lifes))
          (PP (PP (P aboute)
                  (NP (D the) (NS Teames)))
              (CONJP (CONJ &)
                     (ADVP-LOC (WADV elsewher))))
          (. .))
  (ID KNYVETT-1630,87.21))
```

post_crop_label

This crops the label beginning at the indicated character.

This query:

```
node: IP*
query:  ({1}NP-ACC iDoms N*)

post_crop_label{1}: -
append_label{1}: -OBJ
```

applied to this input:

```
( (IP-MAT (NP-SBJ (PRO You))
          (MD must)
          (NEG not)
          (VB exspecte)
          (NP-ACC (Q no) (ADJ greate) (NS matters))
          (NP-TMP (D this) (N time))
          (. ,)) (ID KNYVETT-1630,87.25))
```

produces this output:

```
( (IP-MAT (NP-SBJ (PRO You))
          (MD must)
          (NEG not)
          (VB exspecte)
          (NP-OBJ (Q no) (ADJ greate) (NS matters))
```

```
                (NP-TMP (D this) (N time))
                (. ,))
        (ID KNYVETT-1630,87.25))
```

structural changes

CS has the following structure-changing revision functions. Use them with care, and always keep a backup copy of your original file.

add_leaf_before

add_leaf_before{x}: (pos text)

add_leaf_after

add_leaf_after{x}: (pos text)

move_up_node

move_up_node{x}:

move_up_nodes

move_up_nodes{x, y}:

add_internal_node

add_internal_node{x, y}: new_label

delete_leaf

delete_leaf{x}:

delete_node

delete_node{x}:

delete_subtree

delete_subtree{x}:

It is possible for the described change to result in an illegal tree, that is, a tree with crossing branches, or a tree containing an internal node with no leaf descendants (a pollarded tree?) If this is the case, a warning is given and the tree is not changed.

add_leaf_before, add_leaf_after

This query:

```
node: IP*
query:  (PP iDoms {1}P)

add_leaf_before{1}: (X BULLWINKLE)
add_leaf_after{1}: (Q ROCKY)
```

applied to this input:

```
( (IP-MAT (PP (P Unto)
              (NP (D that)))
          (NP-SBJ (PRO they)
                  (QP (Q all)))
          (ADVP (ADV well))
                (VBD accordyd))
  (ID CMMALORY,5.110) )
```

produces this output:

```
/~*
BULLWINKLE Unto ROCKY that they all well accordyd
(CMMALORY,5.110)
*~/
/*
1 IP-MAT:  2 PP, 3 P
*/

( (IP-MAT (PP (X BULLWINKLE)
              (P Unto)
              (Q ROCKY)
              (NP (D that)))
          (NP-SBJ (PRO they)
                  (QP (Q all)))
          (ADVP (ADV well))
          (VBD accordyd))
  (ID CMMALORY,5.110))
```

move_up_node

This query:

```
node: IP*
query:  (NP iDoms {1}D)

move_up_node{1}:
```

applied to this input:

```
( (IP-MAT (ADVP-TMP (ADV Thenne))
          (PP (P in)
              (NP (Q all) (N haste)))
          (VBD came)
          (NP-SBJ (NPR Uther))
          (PP (P with)
              (NP (D a) (ADJ grete) (N hoost))))
   (ID CMMALORY,3.37))
```

produces this output:

```
( (IP-MAT (ADVP-TMP (ADV Thenne))
          (PP (P in)
              (NP (Q all) (N haste)))
          (VBD came)
          (NP-SBJ (NPR Uther))
          (PP (P with)
              (D a)
              (NP (ADJ grete) (N hoost))))
  (ID CMMALORY,3.37))
```

Notice that the direction of movement is constrained by word order. If the node to move is a middle or only child, a warning is given and the tree is not changed.

move_up_nodes

This query:

```
node: IP*
query:  ({1}Q iprecedes {2}ADJ)
```

```
     move_up_nodes{1, 2}:
```

applied to this input:

```
     ( (IP-MAT (NP-SBJ (PRO You))
               (MD must)
               (NEG not)
               (VB exspecte)
               (NP-ACC (Q no) (ADJ greate) (NS matters))
               (NP-TMP (D this) (N time))
               (. ,)) (ID KNYVETT-1630,87.25))
```

produces this output:

```
     ( (IP-MAT (NP-SBJ (PRO You))
               (MD must)
               (NEG not)
               (VB exspecte)
               (Q no)
               (ADJ greate)
               (NP-ACC (NS matters))
               (NP-TMP (D this) (N time))
               (. ,))
       (ID KNYVETT-1630,87.25))
```

If the indicated move would leave an internal node with no leaf descendants, a warning is given and the tree is not changed.

add_internal_node

This query:

```
     node: IP*
     query:  ({1}MD HasSister {2}VB)

     add_internal_node{1, 2}: MDVP
```

applied to this input:

```
     ( (IP-MAT-SPE (' ')
                   (NP-VOC (N Sir))
                   (, ,)
                   (' ')
                   (IP-MAT-PRN (VBD said)
                               (NP-SBJ (NPR Ulfius)))
                   (, ,)
                   (' ')
                   (NP-SBJ (PRO he))
                   (MD wille)
                   (NEG not)
                   (VB dwelle)
                   (NP-MSR (ADJ long))
                   (E_S .)
                   (' '))
       (ID CMMALORY,3.66))
```

produces this output:

```
     ( (IP-MAT-SPE (' ')
                   (NP-VOC (N Sir))
```

```
                    (, ,)
                    (' ')
                    (IP-MAT-PRN (VBD said)
                                (NP-SBJ (NPR Ulfius)))
                    (, ,)
                    (' ')
                    (NP-SBJ (PRO he))
                    (MDVP (MD wille) (NEG not) (VB dwelle))
                    (NP-MSR (ADJ long))
                    (E_S .)
                    (' '))
          (ID CMMALORY,3.66))
```

If the addition of the indicated node would produce crossing branches in the tree, a warning is given and the tree is not changed.

To add an internal node spanning just one existing node, list the same index twice. For instance, this query:

```
query: (IP* iDoms {1}BE*)

add_internal_node{1, 1}: VP
```

applied to this input:

```
( (IP-MAT-SPE (CONJ but)
              (ADVP (ADV truly))
              (NP-VOC (N gossip))
              (NP-SBJ (PRO you))
              (BEP are)
              (ADJP (ADJ welcome))
              (. ,))
  (ID DELONEY,69.9))
```

produces this output:

```
/~*
but truly gossip you are welcome,
(DELONEY,69.9)
*~/
/*
1 IP-MAT-SPE:  1 IP-MAT-SPE, 13 BEP
*/
( (IP-MAT-SPE (CONJ but)
              (ADVP (ADV truly))
              (NP-VOC (N gossip))
              (NP-SBJ (PRO you))
              (VP (BEP are))
              (ADJP (ADJ welcome))
              (. ,))
  (ID DELONEY,69.9))
```

delete_leaf

The argument specified in the query can match either a part of speech or text node: in either case, the entire part-of-speech/text pair is deleted.

If the indicated leaf is an only child, a warning is given and the tree is not changed.

This query:

```
node: IP*
ignore_nodes: null
query: (NP* iDoms {1}\**)

delete_leaf{1}:
```

applied to this input:

```
( (CP-QUE-SPE (INTJP (INTJ Tush))
              (NP-VOC (N woman))
              (, ,)
              (WNP-1 (WPRO what))
              (IP-SUB-SPE (NP-ACC *T*-1)
                          (VBP talke)
                          (NP-SBJ (PRO you))
                          (PP (P of)
                              (NP (D that))))
              (. ?)) (ID DELONEY,70.40))
```

produces this output:

```
/~*
Tush woman, what talke you of that?
(DELONEY,70.40)
*~/
/*
13 IP-SUB-SPE:  14 NP-ACC, 15 *T*-1
*/

( (CP-QUE-SPE (INTJP (INTJ Tush))
              (NP-VOC (N woman))
              (, ,)
              (WNP-1 (WPRO what))
              (IP-SUB-SPE (VBP talke)
                          (NP-SBJ (PRO you))
                          (PP (P of)
                              (NP (D that))))
              (. ?))
  (ID DELONEY,70.40))
```

delete_node

This is what syntacticians call "pruning". An internal node is deleted, but its descendants remain.

This query:

```
node: FRAG*

query: ({1}ADVP* iDoms ADV*)

delete_node{1}:
```

applied to this input:

```
( (FRAG-SPE (WNP (WPRO What))
            (ADVP-TMP (ADV neuer))
            (NP (D a) (ADJ great) (N belly))
            (ADVP (ADV yet))
            (. ?)) (ID DELONEY,69.5))
```

yields this output:

```
/~*
What neuer a great belly yet?
(DELONEY,69.5)
*~/
/*
1 FRAG-SPE:  5 ADVP-TMP, 6 ADV
1 FRAG-SPE:  15 ADVP, 16 ADV
*/

( (FRAG-SPE (WNP (WPRO What))
            (ADV neuer)
            (NP (D a) (ADJ great) (N belly))
            (ADV yet)
            (. ?))
  (ID DELONEY,69.5))
```

delete_subtree

This deletes the indicated node and all its descendants.

This query:

```
node: IP*
query:  ({1}CONJP* iDoms CONJ*)

delete{1}:
```

applied to this input:

```
( (IP-MAT (NP-SBJ (PRO I))
          (VBP hear)
          (CP-THT (C 0)
                  (IP-SUB (NP-SBJ (NP (N Lady) (N Banbery))
                                  (CONJP-1 (CONJ and)
                                           (NP (D y=e=)
                                           (N Wardon)
                                           (PP (P of)
                                               (NP (NPR All) (NPRS
          Souls))))))
                          (BEP is)
                          (ADJP (ADJ dead))))
          (. .)) (ID ALHATTON,2,242.21))
```

results in this output:

```
( (IP-MAT (NP-SBJ (PRO I))
          (VBP hear)
          (CP-THT (C 0)
                  (IP-SUB (NP-SBJ (NP (N Lady) (N Banbery)))
                          (BEP is)
                          (ADJP (ADJ dead))))
          (. .))
  (ID ALHATTON,2,242.21))
```

examples

Here is an example from a Portuguese corpus. The contraction "dos" had been treated as one word, but the corpus-builders later decided to split it into two pieces, a preposition "$de", and a determiner "os":

Old:

```
(PP (P+D-P dos)
    (NP (ADJ-P grandes)
        (N-P homens)
```

New:

```
(PP (P $de)
    (NP (D-P os)
        (ADJ-P grandes)
        (N-P homens)
```

To make the above change, use this query file:

```
node: IP*
//copy_corpus: t
query: (PP iDoms {1}P+D-P) AND
       (P+D-P iDoms {2}dos) AND
       (P+D-P iPres NP) AND
       (NP iDomsFirst {3}*)

replace_label{1}: P
replace_label{2}: $de
add_leaf_before{3}: (D-P os)
```

The query file as shown will produce a standard CS output file. To produce a file containing the input corpus file, with the changes described, un-comment "copy_corpus: t".

Top of page
Table of Contents

Requirements for Corpus Compatibility

Contents of this chapter:

Table of Contents
CorpusSearch Home

your corpus

With the invention of trainable parsers more corpora are being built. So far, CorpusSearch has been used to search Middle, Old and early Modern English, Chinese, Korean and Yiddish corpora. If you're building a corpus, here's what you need to know to ensure that you can use CorpusSearch to search it.

parse completely

CorpusSearch expects sentences to be completely parsed. That is, every piece of text is expected to have a label affixed to it. If your sentence is only partially parsed, CorpusSearch won't break, but you won't have any way to search the partially parsed areas of text.

labels must be single words

CorpusSearch expects labels to be single strings, that is, containing no spaces (" "). If your label consists of multiple strings, the first string will be interpreted as the label and the next string will be ignored (in the case of a phrase label), or picked up as original text (in the case of a word label). For instance, if you try to use "NOUN PHRASE" as a label, CorpusSearch will interpret "NOUN" as the label and ignore "PHRASE". On the other hand, "NOUN_PHRASE" will be interpreted as a label and could be found using CorpusSearch.

labels must not begin with digits

Labels must not begin with digits ("0", "1", ..."9"). Digits before labels will be interpreted as indices left over from a previous search, and so will be ignored. Labels are allowed to *end* with digits, though. So "PP1" is an acceptable label, but "1PP" is not.

no square brackets

Square brackets ("[" and "]") are used in CorpusSearch to enclose prefix indices. They were a safe choice because the Middle English corpus doesn't contain any square brackets to search for. If your corpus contains square brackets, they will probably have to be changed, or they will be difficult to search for.

tree must be described with round parentheses

CorpusSearch expects the structure of the sentence to be described with round parentheses ("(", ")"). If your tree is described with "{" or "[" or some other system, you will have to convert it to "(" and ")".

wrap your sentences

CorpusSearch expects every sentence to have a "wrapper", that is, a pair of parentheses surrounding the sentence. The wrapper is a useful place to store items that are extraneous to the sentence but linked to it, for instance ID nodes. Here's an example: the "wrapper" consists of the first and last parentheses seen here:

```
(
  (IP-MAT
          (ADVP-TMP (ADV Thenne))
          (NP-SBJ (NPR quene) (NPR Igrayne))
          (VBD waxid)
          (ADVP-TMP (ADV dayly))
          (ADJP (ADJR gretter) (CONJ and) (ADJR gretter))
          (E_S .))
      (ID CMMALORY,5.120))
```

use identification nodes

Although CorpusSearch can function without identification nodes (labelled "ID"), it's better to have them. When CorpusSearch searches the output of a previous search, it uses the ID nodes to keep statistics for the header, footer and summary blocks. Here's an example of an ID node:

```
(ID CMMALORY,5.120)
```

Here, the CMMALORY identifies the source file, 5 is the page number, and 120 is the sentence number in that file. In general, an ID node should have this form:

(ID <source_name>,<free_space>.<sentence_number>)

The information between the source_name and the sentence_number is actually not referenced by CorpusSearch. It could be used to store page numbers (as in the Middle English Corpus), or some other information, or not used at all. The important thing is that the ID_string must begin with a string followed by a comma (to be picked up as the source_name), and end with a "." followed by a sentence number.

Notice that there are no spaces (" ") in the information following the label "ID". This is crucial, because it ensures all the information will be picked up as one string.

CorpusSearch expects to find the ID node just after the sentence ending but inside the sentence wrapper.

an example of an incompatible corpus

In 1994, Beatrice Santorini of the University of Pennsylvania built a corpus of parsed and annotated Yiddish texts. Like Phase 1 of the Middle English corpus, the Yiddish corpus was parsed only to the first level of constituents. This "flat parsing" was searchable using Perl scripts that matched regular expressions.

One passage from the corpus tells a joke that begins this way:

> "When you tell a story to a peasant, he laughs three times. He laughs the first time when someone tells him the story. The second time, when it is explained to him. And the third time, when he understands the story."

I'll examine one sentence from that passage:

> "He laughs the first time when someone tells him the story."

Here it is as it appears in the corpus. (For this discussion, we don't need the definitions of the words and their labels, so I have omitted them.)

```
 (
   [t dem ershtn mol ] [v0 lakht ] [s er ] ,
   [B [c ven ] [s men ] [v0 dertseylt ] [i im ] [d di mayse ] , B]
   )
   (RO,1)
```

The first problem here is the existence of square brackets ("[", "]"), which CorpusSearch doesn't recognize. So the first task is to convert the square brackets to round parentheses:

```
 (
   (t dem ershtn mol ) (v0 lakht ) (s er ) ,
  (B (c ven ) (s men ) (v0 dertseylt ) (i im ) (d di mayse ) , B)
   )
   (RO,1)
```

This form of the sentence can be partly searched by CorpusSearch. For instance, this query:

```
node: *
query:  (v0 iPrecedes s)
```

will find the structure (v0 lakht) (s er), as expected. Notice that the node boundary had to be set to *; if you set the node boundary to IP*, as is normal for the Penn corpora, nothing will be found, because the sentence does not contain IP*.

However, the sentence is still not fully compatible with CorpusSearch because it is not completely parsed. For instance, the phrase "dem ershtn mol" ("the first time") has been parsed as one object. So if you run this query:

```
node: *
query:  (ershtn precedes mol)
```

the structure will not be found. This is because CorpusSearch expects every leaf node to contain exactly two objects: a label and a single-string piece of text. Any extra information will be stored as part of the node but it will usually not be examined by the search functions. These extra pieces of information (in this case, the strings "ershtn" and "mol") behave as useless baggage that is carried along by the sentence vector but never opened.

Similarly, the ", B" that marks the end of the B-labelled clause, and the "," that separates the B-labelled clause from the rest of the sentence, are never actually referenced, so they may as well be removed. The parentheses are enough to convey the information that the B-labelled clause ends, and that the B-labelled clause is separate from the rest of the sentence.

Here is the sentence, fully parsed, and with extraneous labels removed:

```
 (
   (t (det dem) (adj ershtn) (n mol)) (v0 lakht ) (s er )
  (B (c ven ) (s men ) (v0 dertseylt ) (i im ) (d (det di) (n mayse)))
   )
   (RO,1)
```

Now, the query

```
node: *
query:  (ershtn precedes mol)
```

will find the structure as expected:

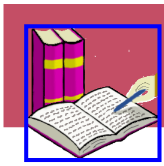
```
/~*
dem ershtn mol lakht er ven men dertseylt im di mayse
(RO,1.3)
*~/

/*
    1 t: 3 adj ershtn, 4 n mol
*/



(0
  (1 t (2 det dem) (3 adj ershtn) (4 n mol))
  (5 v0 lakht)
  (6 s er)
  (7 B (8 c ven)
       (9 s men)
       (10 v0 dertseylt)
       (11 i im)
       (12 d (13 det di) (14 n mayse)))
  (15 ID RO,1.3))
```

Finally, there is the node (RO,1). This identifies the sentence as being part of the first story told by informant Royte Pomerantsen. This needs to be given the standard CorpusSearch ID node form and stuck inside the wrapper. I'll make it sentence number 3:

```
 (
   (t (det dem) (adj ershtn) (n mol)) (v0 lakht ) (s er )
  (B (c ven ) (s men ) (v0 dertseylt ) (i im ) (d di) (n mayse))
(ID RO,1.3)
   )
```

and our sentence is now fully compatible with CorpusSearch.

Top of page
Table of Contents



The CorpusSearch Quick Reference Page

Table of Contents

CorpusSearch Home

To run CorpusSearch:

for automatic output file (command.out)

```
     ...CorpusSearch <command.q> <input-files>
```

for output file with your choice of name (my_name.out)

```
     ...CorpusSearch <command.q> <input-files> -out my_name.out
```

Query file names must end in ".q". Output file names must end in ".out". Preference file name must end in ".prf" and be located in the same directory as the query being run.

some commonly used search functions

X exists	(a node labeled X exists anywhere in token)
X precedes Y	(X precedes Y without overlapping)
X iPrecedes Y	(X immediately precedes Y)
X dominates Y	(X dominates Y with any length path between them)
X domsWords #	(X dominates # of words)
X iDominates Y	(X immediately dominates Y)
X iDomsLast Y	(X immediately dominates Y as last child)
X iDomsMod Y Z	(X dominates Z with at most Y's intervening)
X iDomsNumber # Y	(X immediately dominates Y as first, second, etc. child)
X iDomsOnly Y	(X immediately dominates Y as only child)
X iDomsTotal #	(X immediately dominates # of daughters)
X inID	(X occurs in ID node)
X isRoot	(X is the root node of the token's parse tree)
X sameIndex Y	(X and Y have the same numerical index)
CODING column # X	(X occurs in coding node in column #)

logical operators

A AND B	(predicates A and B - possibly complex - must both be true of hits)
NOT A	(expression A - possibly complex - must be false of hits)
A OR B	(at least one of expressions A and B must be true of hits)
!	(negate one argument to a predicate)
|	(link alternative matches in an argument to a predicate)

wild cards:

*	matches any character
#	matches any digit(s) (0, 1, ... 9)

command-file components:

command	**default**	**command**	**default**
print_complement:	false	query:	required: no default

<table>
<tr><td>print_indices:</td><td>false</td><td>node:</td><td>required: no default</td></tr>
<tr><td>nodes_only:</td><td>false</td><td>ignore_nodes:</td><td>COMMENT|CODE|ID|LB|'|\"|,|E_S|.|/|RMV:*</td></tr>
<tr><td>remove_nodes:</td><td>false</td><td>add_to_ignore:</td><td>adds labels to default</td></tr>
<tr><td>only_ur_text:</td><td>false</td><td>define: <my.def></td><td>includes definition file</td></tr>
</table>

remarks:

begin_remark:
<remark> includes remark in search output file
end_remark

Top of page
Table of Contents

Contents of this chapter:

Table of Contents
CorpusSearch Home

Format of a part-of-speech tagged corpus

CorpusSearch can search files that are tagged for part of speech but not further parsed for syntactic structure.

Here is a template for the format of a POS-tagged corpus file:

```
#!FORMAT=POS_1


Insert header information, if any, below format line above, which must be
the first line in the file.


<text>


WORD1/TAG WORD2/TAG WORD3/TAG ..... ./.



WORD1/TAG WORD2/TAG WORD3/TAG ..... ?/.



.....


</text>
```

Every word in a POS-tagged file after the initial "" tag should end with a backslash and tag, followed by a space. There can be no spaces within a word or between a word and its tag. Note that every sentence of the corpus must end with a punctuation mark. Also, there must be a blank line between sentences. If the format of the file is "POS_1", the tag for sentence final punctuation must be a period. If the format is "POS_0", an alternative format, the tag for sentence final punctuation is "PONFP". Sentence internal punctuation must also be treated as a separate word with a tag, which should be different from the sentence final punctuation tag.

Search functions

The query file for searching a POS-tagged corpus looks much like that for a parsed corpus. The node boundary, however, is always $ROOT. CorpusSearch treats POS-tagged files as containing sentences parsed with a completely flat structure, with every word/tag pair as an immediate daughter of the root node. The tag for a word is treated as its mother, so that a query like "(N iDoms king)" returns sentences containing the word/tag pair "king/N". Because of the flat structure of a POS-tagged file, many CorpusSearch functions cannot be used. Below is a list of those that are ordinarily appropriate. The function "Neighborhood" works only on POS-tagged files.

Exists (variants: exists)

Exists searches for a POS tag or text anywhere in the sentence. For instance, this query:

```
(MD0 exists)
```

will find this sentence:

```
/~*
I shal not conne wel goo thyder ./. (ID CMREYNAR,14.261)
*~/

/*
    4 MD0 conne
*/

( (PRO I) (MD shal) (NEG not) (MD0 conne) (ADV wel)) (VB goo) (ADV thyder) )
```

iDominates (variants: idominates, iDoms, idoms)

iDominates means "immediately dominates". That is, x dominates y if y is a child of x. So this query:

```
((PRO iDominates he) AND (FP iDominates ane))
```

finds this sentence:

```
/~*
Sythen he ledes +tam by +tar ane,
(CMROLLEP,118.978)
*~/

/*
    2 PRO he, 7 FP ane
*/
( (ADV Sythen) (PRO he) (VBP ledes) (8 PRO +tam) (10 P by) (12 PRO$ +tar) (13
FP ane) (. ,) )

/*
```

Notice that "iDominates" describes the relationship between a POS tag and its associated text (e.g., "FP" and "ane").

iPrecedes (variants: iprecedes, iPres, ipres)

This function is true if and only if its first argument immediately precedes its second argument in the text/tag string.

The following query:

```
query: (as iPrecedes sone) AND (sone iPrecedes P)
```

finds this sentence:

```
/~*
and as sone as he myght he toke his horse .
(CMMALORY,206.3401)
*~/
/*
2  as, 3 sone, 4 P as
*/

( CONJ and) (ADVR as) (ADV sone) (P as) (PRO he) (MD myght) (PRO he) (VBD toke)
(PRO$ his) (N horse) (. .) )
```

Neighborhood (variant: neighborhood)

Neighborhood takes three arguments, two words or tags and a number. It searches for sentences in which the two words/tags occur within a certain number of words of one another. For instance, this query:

```
query: (whoreson Neighborhood 2 wilt)
```

will return all tokens in the corpus in which the word "whoreson" is within two words of the word "wilt," for instance, the following sentence:

```
/~*
why thou whoreson when wilt thou be maried?
(DELONEY,79.296)
*~/
/*
3 whoreson,  5 wilt
*/

(  (WADV why) (PRO thou) (N whoreson) WADV when) (MD wilt) (PRO thou) (BE be)
(VAN maried)  (. ?) )
  (ID DELONEY,79.296))
```

Precedes (variants: precedes, Pres, pres)

"x precedes y" means "x comes before y in the sentence but perhaps not immediately". So this query:

```
(VB precedes N)
```

finds this case:

```
/~*
thenne have ye cause to make myghty werre upon hym.
(CMMALORY,2.25)
*~/

/*
    6 VB make, 8 N werre
*/

( (ADV thenne) (HV have) (PRO ye) (N cause) (TO to) (VB make) (ADJ myghty) (N
werre) (P upon)
(PRO hym) (. .) )
      (ID CMMALORY,2.25))
```

Top of page
Table of Contents

CorpusDraw Basic Concepts

◀ ▶

Contents of this chapter:

Table of Contents

CorpusSearch Home

What is CorpusDraw?

CorpusDraw displays the tree structures assigned to sentences in a parsed corpus and allows an annotator to edit these trees in the course of corpus construction or revision. It can also be used to display parse trees for presentation purposes.

Input to CorpusDraw

CorpusDraw accepts the following command line arguments:

1. an optional specification of structural constraints on what sentences to display (command file).
2. the corpus file to display (source file).

In addition, CorpusDraw will read in a file of legal syntactic and part-of-speech tags, if one is supplied. The corpus file, command file, and CorpusSearch program itself must reside in different directories. The recommended directory configuration has a root corpus directory with three sister subdirectories, one for CorpusSearch itself, one for the corpus source files and one for the command files and for the file of legal tags. When starting CorpusDraw, the current directory should normallybe the root directory of the corpus, with the path to the corpus file being worked on specified on the command line.

source file

A source file is any file that contains parsed, labelled sentences. This could be a file from the Penn Parsed Corpora of Historical English or from another parsed corpus.

command file

The command file contains a query, which describes a structure that every sentence must meet to be displayed by CD. The use of such a command file allows the annotator to view only those sentences relevant for a given editing change being implemented on the corpus.

file of legal tags

If CorpusDraw is given a file of legal tags, it will constrain node labels, both phrasal and part-of-speech tags, to come from the list in this file. This constraint prevents the accidental introduction of ill-formed labels.

To create a file of legal tags, the following lines should be inserted into a command file with the name "tags.q":

```
corpus_encoding: UTF-8

make_tag_list: t
```

The corpus_encoding line should be changed if the corpus font encoding is other than UTF-8. The command file may not contain any other contents. When CorpusSearch is run on the entire parsed corpus with this command file, a file with the name "tags.tag" is created. This is the legal tags file used by CorpusDraw.

The CorpusDraw graphical user interface

The CorpusDraw GUI is intended to be largely self-explanatory. The display, which can be seen by clicking here, contains the follow parts:

- a tree display window
- a window containing the text of the displayed sentence
- a top row of buttons for editing the tree
- a second row of buttons for modifying the display for ease of use

The scroll bars at the bottom and on the right edge of the tree display window allow different parts of the tree to be centered in the window. This can also be accomplished by clicking on the word in the text window that the user wishes to place in the center of the display. The arrows at left of the editing button row move the display from one sentence to the next.

The editing buttons allow the annotator to change node labels, to move nodes and their descendants around in the tree, to coindex nodes, and to add empty categories the various types specified in the legal tags file. CorpusDraw will not permit the annotator to accidently change the order of words in the sentence or to delete any.

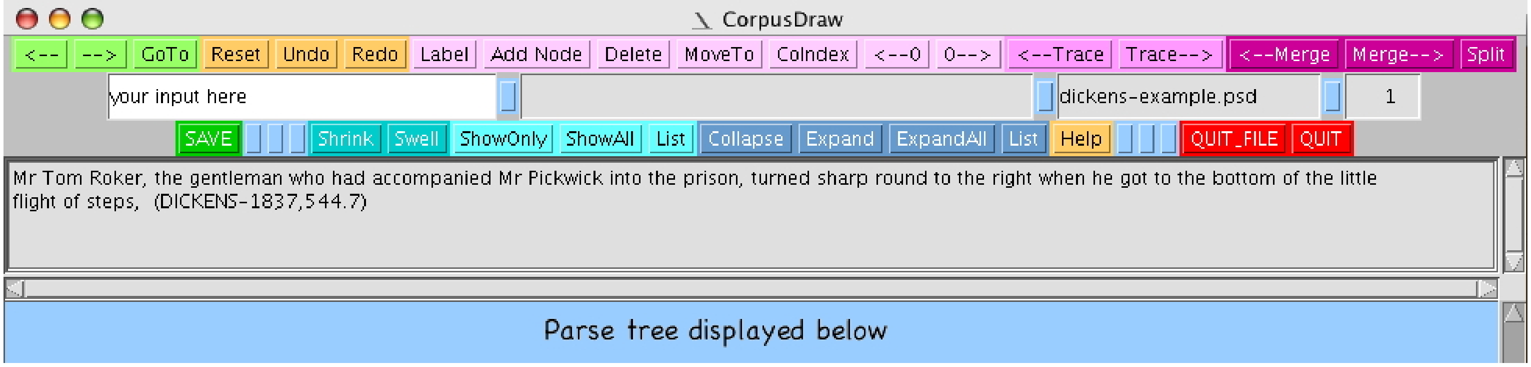
The actions controlled by the editing buttons can also be triggered by the use of shortcuts, both keystrokes and mouse clicks. Some of these require a sequence of keystrokes or clicks. A current list of these shortcuts can be found in the next section of the manual. Here is a QuickTime movie of these shortcuts in action:

Output of CorpusDraw

When CorpusDraw is displaying a file with the name "foo.psd" and the file is saved after certain changes are made, the saved file has the name "foo.psd.new." This change in name guarantees that changes can easily be discarded.

Top of page
Table of Contents

The CorpusDraw Editing Buttons



Mr Tom Roker, the gentleman who had accompanied Mr Pickwick into the prison, turned sharp round to the right when he got to the bottom of the little flight of steps, (DICKENS-1837,544.7)

Parse tree displayed below

Editing buttons

Table of Contents
CorpusSearch Home

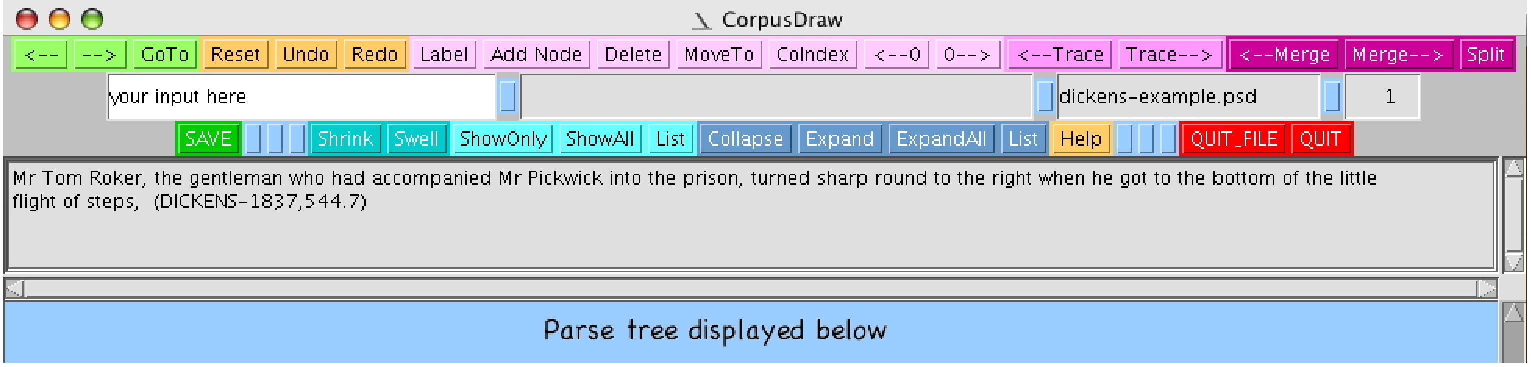
The corpus is displayed one token at a time, where a token is a matrix sentence with an ID number. Most of the editing functions require the user first to select a node or nodes in the parse tree. The action indicated by the button name is taken with respect to the selected node(s). When two nodes are selected, the order of selection is normally significant. Often, after clicking on a button, the user must enter information into the input box located at the left edge of the second row of the toolbar.

<--	Previous Token	Moves to the previous token.
-->	Next Token	Moves to the next token.
GoTo	Go To	Moves to the token whose number is subsequently entered into the input box.
Reset	Reset	Resets the current token to its state when the current file was last saved. **Disabled.**
Undo	Undo	Undoes the last action. May be repeated.
Redo	Redo	Redoes the last undone action. May be repeated.
Label	Change Label	Changes the label of the selected node. Will ask for user input.
Add Node	Add Node	Adds a node with label to be supplied as the mother of the selected node.
Delete	Delete	Deletes the selected node. Part-of-speech tags and text elements may not be deleted.

MoveTo	Move To	After two nodes are selected, moves the first node to be a daughter of the second. Fails if this movement would cause a change in word order.
CoIndex	CoIndex	co-indexes the second of two selected nodes with the first.
<--0	Insert Zero Left	Inserts a zero empty category as the left sister of the second of two selected nodes.
0-->	Insert Zero Right	Inserts a zero empty category as the right sister of the second of two selected nodes.
<--Trace	Insert Trace Left	Inserts a trace empty category as the left sister of the second of two selected nodes.
Trace-->	Insert Trace Right	Inserts a trace empty category as the right sister of the second of two selected nodes.
<--Merge	Merge With Previous Token	Merges the current token as the daughter of the root node of the previous token.
Merge-->	Merge With Next Token	Merges the current token as the daughter of the root node of the next token.
Split	Split Token	Splits the current token into two tokens at the selected node. **Splits cannot be undone.**

Top of page
Table of Contents

The CorpusDraw File and Display Buttons



your input here

dickens-example.psd

1

Mr Tom Roker, the gentleman who had accompanied Mr Pickwick into the prison, turned sharp round to the right when he got to the bottom of the little flight of steps, (DICKENS-1837,544.7)

Parse tree displayed below

File and display buttons

Table of Contents

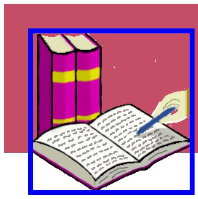
CorpusSearch Home

The file buttons are for saving the working file and quitting the program. The display buttons allow the user to alter the display to make working on the token easier. When nodes are collapsed, only the root of the collapsed subtree is visible.

SAVE	Save	Saves the file being viewed in its current state.
Shrink	Shrink	Shrinks the token being viewed to make more of it visible. **Disabled.**
Swell	Swell	Undo a prior shrink operation. **Disabled.**
ShowOnly	Show Only Listed Nodes	Displays only node labels on the Show list.
ShowAll	Show All Nodes	Undoes a Show Only operation and displays all nodes.
List	Node List for Show Only	Requests user to input a list of node labels for Show Only.
Collapse	Collapse Node(s)	Collapses previously selected node or nodes.
Expand	Expand Node(s)	Expands previously selected and collapsed nodes.
ExpandAll	Expand All Nodes	Expands all nodes in the current tree, including nodes with labels on the collapse list.
List	List of Nodes to Collapse	Asks the user to enter a list of node labels to collapse in the current and subsequently displayed tokens.
Help	Help	Puts up a list of keyboard shortcuts.
QUIT_FILE	Quit Current File	Puts up a quit dialog. The user is prompted to save before quitting.

QUIT	Quit	Puts up a quit dialog. The user is prompted to save before quitting.

Top of page
Table of Contents



CorpusDraw Shortcuts

CorpusDraw Users Guide: Shortcuts

Table of Contents

CorpusSearch Home

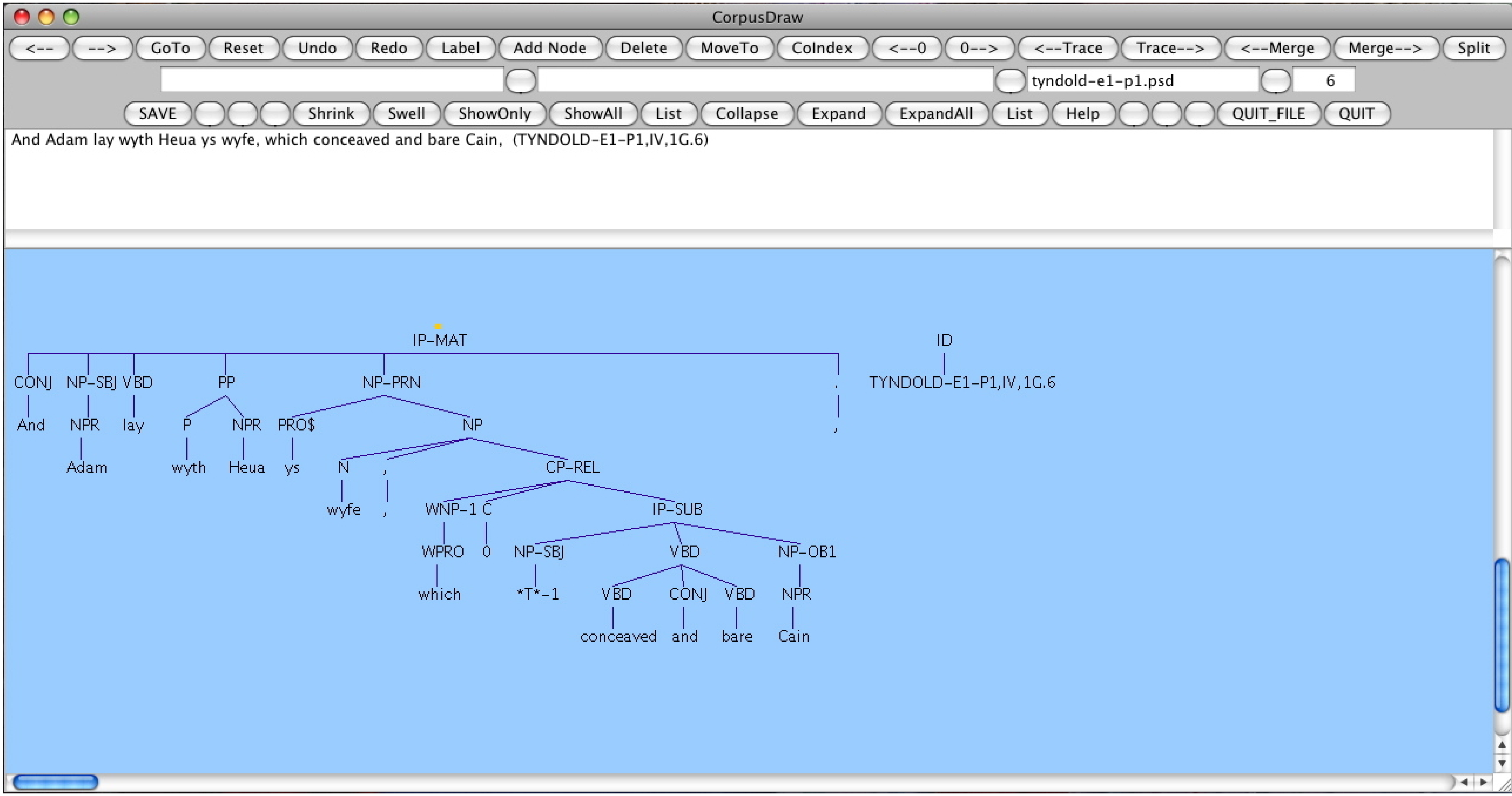
Scrolling shortcut: to scroll to a different part of the tree, left-click on the corresponding word in the text window.

Function	Keystroke	(optional) Mouseclick
<--	esc 2	
-->	esc 1	
GoTo	esc g	
Undo	esc u	right-click on background
Redo	esc r	shift right-click on background
Label	esc l	shift left-click
Add Node	esc n	ctrl left-click
Delete	esc d	shift ctrl left-click
MoveTo	esc m	left-click on node to move, right-click on target node
CoIndex	esc c	
<--0	esc b	

0-->	esc a	
<--Trace	esc x	left-click on following sibling, ctrl right-click on antecedent
Trace-->	esc y	left-click on preceding sibling, ctrl shift right-click on antecedent
<--Merge	esc p	
Merge-->	esc f	
Split	esc s	

To select nodes using the keyboard, use the arrow keys to move the orange dot to the desired node. Then hit the space bar to select the node (you'll see it surrounded by a red box.) Then continue as usual.

Top of page
Table of Contents



tyndold-e1-p1.psd

6

And Adam lay wyth Heua ys wyfe, which conceaved and bare Cain, (TYNDOLD-E1-P1,IV,1G.6)