Contents

1 CORPUSSEARCH USER'S MANUAL

1.1 Getting Started

CorpusSearch is a search program that searches for linguistic structures in a corpus of parsed, labelled sentences. The following diagram describes the system of input and output to CorpusSearch:

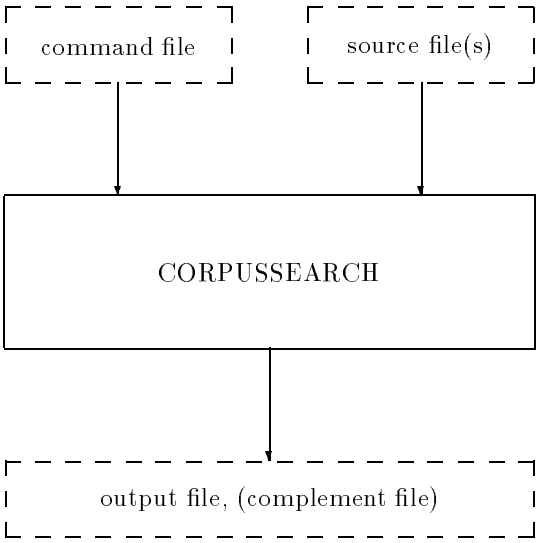

Figure 1: input and output to CorpusSearch

1.1.1 input to CorpusSearch

CorpusSearch needs two pieces of information:

1.) what sentences to search (source file(s)).

2.) 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 Middle English (or other) corpus, an output file from a previous search, or perhaps a file of sentences

that the user has cut and pasted together.

command file

The command file contains a query, which describes the structures being searched for, and possibly other material, describing what node boundaries in which to search, and various options for printing the output (see The Command File).

1.1.2 output of CorpusSearch

CorpusSearch prints out an output file, and optionally, a complement file.

output file(s)

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 sentences found with the structure, the total number of sentences searched, and the number of distinct boundary nodes containing the structure ("hits"). Notice that the number of hits may change depending on the definition of the boundary node (see The Command File).

complement file(s)

A complement file is produced if the command file contains this line:

```
print_complement: true
```

The complement file, if there is one, contains all the sentences in the source file that do *not* contain the searched-for structure. The output file and complement file are complementary sets that together contain all the sentences in the source file.

1.1.3 running CorpusSearch on babel

babel is a mainframe computer run by the Linguistics Department at the University of Pennsylvania. The following instructions are for those who have an account on babel.

To run CorpusSearch on babel, add these lines to your .cshrc file:

```
prepend         PATH /pkg/java-1.2ea6/bin
setenv CLASSPATH /pkg/ling/MIDENG/PPCME2/clean\_search
set mecorpus = /home/ataylor/MIDENG/PPCME2/SearchMe
```

The line beginning "prepend PATH" enables your account to run java programs.

The line beginning "setenv CLASSPATH" ensures that java will be able to find CorpusSearch when you call it from any directory in your account.

The line beginning "set mecorpus" saves typing. Instead of typing "/home/ataylor/MIDENG/PPCME2/SearchM (where the corpus is stored) in your java command, you can type "$mecorpus" to get the same result.

your query/output directory

Make a new directory in your account; you might call it "corpus_stuff". This directory will hold your query files (ending with ".q"), your output files (ending with ".out"), and possibly your complement files (ending with ".cmp").

your command file

Make a new file in your directory, using emacs, vi or any standard editor. Give the file a name ending in ".q". This will be your command file. The only thing this file *must* contain is a query — all other commands are optional (see The Command File). To see how the program runs, you might want to try using an extremely simple command file. Your command file , let's call it "NP.q", could contain just this line:

```
query:  (NP* iDominates PP*)
```

This query searches for noun phrases that immediately dominate prepositional phrases.

running the search

This is the general form for running CorpusSearch:

```
java CorpusSearch <command file> <source file(s)>
```

Here's an example:

```
java CorpusSearch NP.q $mecorpus/*
```

This command will search the entire corpus (because of the "/*" after "$mecorpus".. The output will appear in a file called "NP.out".

Be patient; a search of the entire corpus currently takes about 5 minutes, depending on the complexity of the query. To run a search in the background, write "&" at the end of your command:

```
java CorpusSearch NP.q $mecorpus/* &
```

To run a search only on Malory, use this command:

```
java CorpusSearch NP.q $mecorpus/*malory* &
```

In general, to run a search on a subset of the entire corpus, describe your subset using standard Unix terminology as it applies to the names of the particular files you want to search.

1.2 The Rocche Sentence

I chose a simple sentence to use as an example throughout the user's manual. I'll call it "the rocche sentence". Here it is as Malory wrote it:

and so hit londid undir that rocche.

The sentence describes Percivale's ship, landing under a cliff ("rocche").

Here it is, parsed and labelled, as it appears in the corpus:

```
( (IP (CONJ and)
      (ADVP (ADV so))
      (NP-SBJ (PRO hit))
      (VBD londid)
      (PP (P undir)
          (NP (D that) (N rocche)))
      (E_S .)) )
```

and here it is drawn as a tree:

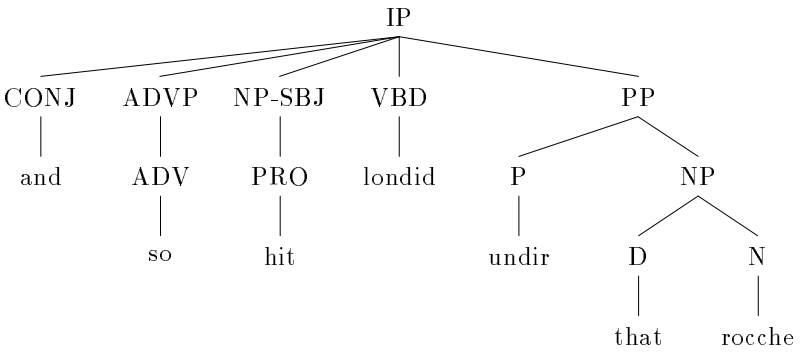

Figure 2: the rocche sentence shown as a tree

1.3 CorpusSearch General Principles

1.3.1 labels and text

"Labels" are the all upper-case tags inserted by the linguists who prepared the corpus (e.g., "IP", "CONJ", "N") "Text" refers to the mostly lower-case original words of text (e.g. "so", "hit"). Every node in the tree has a label, and the leaf nodes also have text. CorpusSearch can conduct searches on labels or text, as described below. When searching for text, spelling and upper-case/lower-case variations must be described explicitly (usually with an argument list.) For instance:

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

1.3.2 fuzzy tree structure

For the purposes of dominance, text and its associated node label are considered separate objects. Thus, "PRO" dominates "hit" in the rocche sentence. For the purposes of precedence, text and its associated label are considered to be one object. Thus, "that" sister-precedes "rocche" in the rocche sentence, because the labels associated with "that" and "rocche" are sisters.

1.3.3 wild cards

CorpusSearch supports two wild cards, namely * and #.

1.3.4 * (character wild card)

The operator '*' works as in regular expressions, that is, it stands for any combination 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 is the wild card and will match any label or text. For instance,

```
(PP iDomsOnly *)
```

will return all sentences containing a PP with a single child (not the rocche sentence). * may be used anywhere in the function argument; beginning, middle or end.

1.3.5 searching for *

Some labels, for example "*con*". 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 command:

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

1.3.6 # (digit wild card)

The # operator is the wild card for digits. For instance, `(PP iDominates P#)` will return nodes like this:

```
(20 PP (21 P21 wi+t)
       (22 P22 ynne)
       (23 NP (24 D +tat)
              (25 N citee)))
```

1.3.7 node boundary command

The node boundary command tells the program what kind of node to search for to contain the described structures. If the command file doesn't list a "node:" command, CorpusSearch uses the default node boundary IP*.

CorpusSearch can treat one instance of a label as the node command and also the argument to a search function, as in:

```
node:  PP*
query:  (PP iDomsNumber1 RP)
```

If you don't have a particular node in mind, use the node command "*".

CorpusSearch will accept a list of nodes for the node boundary command. For instance, this is a legitimate command:

```
node:  PP*|NP*|ADJP*
```

This structure is contained in the rocche sentence:

```
node:  PP
query:  (NP iDominates N)
```

```
(PP (P undir)
    (NP (D that) (N rocche)))
```

By default, only the nodes specified in the node command will be printed out (not the entire sentence containing them). To print the entire parsed sentence, include this line in your command file:

```
nodes_only:  false
```

1.3.8 nodes to ignore

There are some nodes in the corpus that linguists usually don't want to consider as part of the strucure of the sentence, for instance, punctuation, line breaks, page numbers, and comments. CorpusSearch will ignore all nodes whose labels are contained in the "ignore- list". This is the default version of the ignore-list:

```
COMMENT|CODE|ID|LB|'|"|,|E_S|/
```

For instance, if you run this query:

```
query: (NP* iPrecedes PP*)
```

This sentence will be returned:

```
**********************************************************************begin_comments

 1 IP-MAT-SPE: 5 NP-1, 9 PP

************************************************************************end_comments

**********************************************************************begin_ur_text

There ar two bretheren beyond the see,
(CMMALORY,15.439)

************************************************************************end_ur_text

 (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.

To add labels to the default ignore-list, include this command in your command_file:

```
add_to_ignore:  <list_of_labels>
```

For instance, if you want to ignore traces, include this command in your command_file:

```
add_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
```

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

1.3.9 searching output

The output of one search may be used directly as input to the next search. CorpusSearch recognizes output files as those ending in ".out" or ".cmp".

1.4 Query Language

1.4.1 search function arguments

The arguments to a search function are usually node labels or lists of node labels (e.g. "NP", "CP", "VB*|HV*"). Text can also be used (e.g. "Percivale, "that|That".)

1.4.2 wild cards

CorpusSearch supports two "wild cards" for use in search function arguments, namely *, which represents any (or no) characters, and #, which represents digits.

1.4.3 search function calls

The most basic query is a search-function call. Each one of the following search-function calls is a correct query in itself. Any number of these calls can be combined into more complex queries.

```
(NP-SBJ iDomsLast N)

(VBD|VBG iPrecedes NEG)

(NP* iDominates !NPR)
```

1.4.4 logical operators

Search-function calls may be combined using the logical operator AND. Because of the constraints of the same-instance problem, search-function calls must be appended to the query one at a time:

```
(((NP-SBJ iDomsLast N) AND (VBD|VBG iPrecedes NEG)) AND (C dominates that))
```

AND acts on search-function calls. 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.)

1.4.5 a formal grammar of the query language.

arg — an argument to a search function. Examples: NP-SBJ, NP*, !NPR, Percivale.

un — a unary search function. Example: exists, domsWords#, iDomsTotal#.

bin — a binary search function. Examples: iDomsLast#, iPrecedes, precedes, iDomsNumber#.

AND — binary logical operator AND.

$< stmt > \longrightarrow < call >$

$\quad\quad | \ (< stmt >< append >)$

$< append > \longrightarrow \ AND < call >$

$< call > \longrightarrow$ (arg bin arg)

$\quad\quad |$ (arg un)

1.5 Search Functions

1.5.1 x search-function y

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

1.5.2 exists

searches for label or text anywhere in sentence. These structures are found in the rocche sentence:

`(rocche exists)`

`(PRO exists)`

1.5.3 precedes

precedes means"sister precedes". That is, x sister precedes y when x and y are immediately dominated by the same node, and x is previous to y. This function will accept label or text as any combination of x and y. These structures are found in the rocche sentence:

`(ADVP precedes VBD)`

`(that precedes rocche)` (see"fuzzy tree structure" above)

but this structure is *not* found in the rocche sentence:

`(ADVP precedes PRO)` (because ADVP and PRO are not sisters.)

1.5.4 iPrecedes

iPrecedes means "immediately sister precedes." That is, x immediately sister precedes y when x and y are immediately dominated by the same node, and x is immediately previous to y. "iPrecedes" is a subset of "precedes". These structures are found in the rocche sentence:

`(ADVP iPrecedes NP-SBJ)`

`(so iPrecedes hit)`

but this structure is *not* found in the rocche sentence:

`(ADVP iPrecedes VBD)` (because it does not immediately precede)

1.5.5 anyPrecedes

anyPrecedes means "precedes anywhere but does not dominate." That is, x precedes y somewhere in the sentence, but y is not contained in the sub-tree dominated by x. "anyPrecedes" is a superset of "precedes". The following structures are found in the rocche sentence:

```
(ADVP anyPrecedes PRO)
```

```
(hit anyPrecedes londid)
```

but this structure is *not* found in the rocche sentence:

```
(NP-SBJ anyPrecedes PRO)
```

1.5.6 dominates

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.) These structures are found in the rocche sentence:

```
(PP dominates N)
```

```
(PP dominates rocche)
```

but this structure is *not* found in the rocche sentence:

```
(D dominates N)
```

1.5.7 iDominates

iDominates means "immediately dominates". That is, x dominates y if y is a child (exactly one generation apart) of x. These structures are found in the rocche sentence:

```
(ADVP iDominates ADV)

(PRO iDominates hit)
```

but this structure is *not* found in the rocche sentence:

`(PP iDominates N)` (N and PP are more than one generation apart)

1.5.8 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. These structures are found in the rocche sentence:

```
(NP-SBJ iDomsOnly PRO)

(PRO iDomsOnly hit)
```

but this structure is *not* found in the rocche sentence:

`(PP iDomsOnly P)` (because P is not the only child)

1.5.9 iDomsNumber#

iDomsNumber# means "immediately dominates as the #th child" where # is tacked on to the end of iDomsNumber. "iDomsNumber#" must be picked up by the parser as one string.) That is, x immediately dominates y as the #th child if x immediately dominates y and y is the #th child of x. Notice that iDomsNumber1 is a superset of iDomsOnly. These structures are found in the rocche sentence:

```
(NP iDomsNumber2 N)

(VBD iDomsNumber1 londid)
```

but this structure is *not* found in the rocche sentence:

`(PP iDomsNumber` P)2 (because P is the number 1 child)

1.5.10 iDomsLast#

iDomsLast is similar to iDomsNumber but it counts backward from the last child. So iDomsLast1 means "immediately dominates as the last child", iDomsLast2 means "immediately dominates as the second-to-last child", and so on. These structures are found in the rocche sentence:

```
(IP iDomsLast1 PP)

(IP iDomsLast3 NP-SBJ)
```

but this structure is *not* found in the rocche sentence:

```
(IP iDomsLast2 NP-SBJ)
```

1.5.11 domsWords#

domsWords# counts the number of words dominated by the search-function argument. So "domsWords4" means "dominates 4 words", domsWords2 means "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:

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

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

So this query:

```
(NP domsWords4)
```

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

```
(NP
    (NP-POS
    (D the)
    (N$ modirs)
            (NP-PRN *ICH*-1))
    (N syde)
    (NP-PRN-1 (NPR Igrayne))))
```

1.5.12 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 domsWords<3)
```

will return this structure (ignoring the trace *ICH*-3):

```
(NP
    (D a)
    (N knyght)
    (CP-REL *ICH*-3)))
```

1.5.13 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 domsWords>3)
```

will return this structure:

```
(NP
    (N accord)
    (PP
        (P betwixe)
        (NP
            (NP
        (D the)
                (N lady)
                (NP-PRN
                        (NPR Igrayne)))
            (CONJP
                   (CONJ and)
                   (NP
                       (PRO hym)))))))
```

1.5.14 iDomsTotal#

iDomsTotal# counts the number of daughters immediately dominated by the search- function argument. So this query:

```
(PP iDomsTotal3)
```

will return this structure:

```
(PP
    (RP oute)
    (P of)
    (NP
        (D the)
        (N castel)))
```

Notice that the PP in this case immediately dominates a total of 3 daughters (RP, P, NP), but dominates 4 words (oute, of, the, castel).

1.5.15 iDomsTotal<#

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

```
(PP iDomsTotal<3)
```

will return this structure:

```
(PP
    (P within)
    (NP
        (ADJ forty)
        (NS dayes)))
```

Notice that in this case the PP immediately dominates a total of less than 3 daughters (P, NP) but dominates 3 words (within, forty, dayes).

1.5.16 iDomsTotal>#

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

```
(PP iDomsTotal>3)
```

will return this structure:

```
(PP
    (ADV clene)
    (RP oute)
    (P of)
    (NP
        (D the)
        (N sadyll)))
```

Notice that in this case PP immediately dominates a total of 4 daughters (ADV, RP, P, NP) but dominates 5 words (clene, oute, of, the, sadyll).

1.5.17 shorthand for search-function names

CorpusSearch allows shorthands and lower-case/upper-case variations for the names of search functions. For instance, "iDominates" may be written "idominates" or "iDoms". If you try a shorthand and it isn't allowed by CorpusSearch, you'll get an error message from the query parser. If you feel that a certain shorthand should be allowed, write to the SearchMistress, Beth Randall.

1.6 Logical Operators

1.6.1 about logical operators

CorpusSearch supports the following logical operators:

AND	(and search-function call)
!	(not argument)
|	(or argument)

Also, the printing command print_complement can be thought of as NOT applied to a query.

1.6.2 search-function operators vs. argument operators

AND acts on search-function calls; ! and | act on arguments to the search functions.

1.6.3 AND; time-saver

AND has a time-saving switch, so that if the first structure is not found in the sentence being searched, the second structure is not looked for. Therefore, if you know that one structure is rarer than the other, you can save time by listing the rarer structure first.

1.6.4 same-instance

AND has been implemented with same-instance as a default. So

```
((IP iDomsNumber1 VBP|VBD) AND (IP iDomsNumber2 ADVP|PP*))
```

will return only sentences where the same instance of IP has the described number 1 and 2 children. Sentences containing one IP with number 1 child VBP and some other IP with number 2 child ADVP will not be returned.

Same-instance is triggered by matching argument strings. So

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

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

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

will return sentences with the same instance or different instances (because the argument lists do not match as strings.)

1.6.5 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. Arguments with the same pre-index must coincide, arguments with different pre-indices must not coincide. For example, suppose you are looking for two noun-phrases which are sisters; each noun-phrase immediately dominates a pronoun. Use pre-indices as follows:

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

Or, suppose you're looking for one NP* which immediately dominates PRO and a different NP* which immediately precedes VBD. Use pre-indices as follows:

```
(([1]NP* iDominates PRO) AND ([2]NP* iPrecedes VBD))
```

1.6.6 ! (not-argument)

! is used to negate the argument to a search function. For instance,

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

will return sentences that contain the structure "something, not NP-SBJ, immediately precedes VBD" (*not* including the rocche sentence.)

1.6.7 ! (not-argument) reports last legitimate node

If there is more than one candidate for the !argument, CorpusSearch reports the last legitimate node encountered. For instance,

```
(IP iDominates !NP-OB1)
```

will report the last node iDominated by IP, if none of those nodes are NP-OB1. Thus, in the rocche sentence, IP iDominates CONJ, ADVP, NP-SBJ, VBD, and PP. After checking that none of those are NP-OB1, CorpusSearch reports PP as the result.

1.6.8 ! 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 will abort the search:

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

1.6.9 not before prefix indices

If you need to use both ! and prefix indices, put the ! before the indices. This is a legitimate query, that looks for two different noun phrases, neither of them immediately dominating a trace:

```
query:  (([1]NP* iDominates ![3]/**) AND ([2]NP* iDominates ![4]/**))
```

If you didn't use the prefix indices 3 and 4 in the above query, you wouldn't find any sentences. Without the indices, CorpusSearch would look for two different noun phrases, each immediately dominating the same not-trace object.

1.6.10 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*."

1.6.11 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*".

1.7 The Command File

1.7.1 optional commands:

Optional (non-query) commands must be written *before* the query. All the optional commands have default values which are used if no value is found in the command file.

1.7.2 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".

1.7.3 search commands

add_to_ignore: (String label_list)

default " " (empty string)

adds given labels to the ignore_list. For instance,

```
add_to_ignore:  \**
```

will tell CorpusSearch to ignore traces for this search.

ignore_nodes: (String ignore_list)

default COMMENT|CODE|ID|LB|'|',|E_S|/

tells CorpusSearch what nodes to ignore, usually punctuation and comments.

node: (String node_boundary)

default IP*

gives CorpusSearch a node boundary to search within.

The node boundary influences the statistics kept by CorpusSearch, since the number of hits is the number of boundary nodes containing the structure described in the query.

Also, the node boundary determines what nodes are removed if remove_nodes is true, and the nodes that are printed if nodes_only is true.

query: (String query)

default ERROR

Every command file must contain a query, although it need not contain anything else. The query must be the last item in the command file.

1.7.4 printing 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. 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: (String remark) end_remark

default " " (empty string)

tells CorpusSearch to print user's remark in the output Preface. This is a way for the user to write a note to herself, 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:

```
**********************************************************************begin_preface

    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)
************************************************************************end_preface
```

nodes_only: (boolean true or false)

default true

If true, CorpusSearch prints out only the nodes (as defined in "node". above) 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*
query:  (ADVP* iDominates ADVP*)
```

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

```
************************************************************************begin_comments

 2 ADVP: 3 ADVP

************************************************************************end_comments

************************************************************************begin_ur_text

certayn and wit-owte doute, Ihon is is name.
(CMAELR3,45.574)

************************************************************************end_ur_text

(NODE (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:

```
**********************************************************************begin_comments

 2 ADVP: 3 ADVP

**********************************************************************end_comments

**********************************************************************begin_ur_text

certayn and wit-owte doute, Ihon is is name.
(CMAELR3,45.589)

**********************************************************************end_ur_text

(
(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))
```

only_ur_text: (boolean true or false)

default false

If true, CorpusSearch prints out only the ur_text version of the sentences containing the searched-for structure. It also prints the ur_text version of the nodes in which the structures were found. This could be a useful step at the very end of a search, providing a file full of sentences ready to be copied into a research paper.

NOTE: Since the output of an only_ur_text search contains no parsed sentences, it cannot be used as the input to a new search.

Here's a piece of only_ur_text output resulting from this query:

```
node:  ADVP*
```

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

**********************************************************************begin_ur_text

certayn and wit-owte doute, Ihon is is name.
(CMAELR3,45.589)

ADVP:   certayn and wit-owte doute

**********************************************************************end_ur_text
```

print_comments: (boolean true or false)

default true

tells CorpusSearch whether or not to print a comment block before each output sentence.

Here's an example of a comment block, describing where the structure (NP* iPrecedes PP*) was found in the output sentence:

```
**********************************************************************begin_comments

 1 IP-MAT-SPE: 5 NP-1, 9 PP

**********************************************************************end_comments
```

print_complement: (boolean true or false)

default false

The idea behind print_complement is to split the input file into two complementary sets, the output file and the complement file.

If print_complement is true, CorpusSearch prints a separate file containing all the sentences found in the input that did *not* contain the searched-for structure. The name of the complement file is the same as the name of the output file, but with ".cmp" replacing ".out".

print_indices: (boolean true or false)

default true

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 sentence.

Here's a piece of an output sentence 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)))
```

print_parsed: (boolean true or false)

default true

tells CorpusSearch whether or not to print the parsed sentences which contain the searched-for structure.

print_ur_text: (boolean true or false)

default false

if true, CorpusSearch prints an ur_text block above every output sentence, containing the original sentence in text-only form.

If false, CorpusSearch omits the ur_text block.

Here's an example of an ur_text block:

```
**********************************************************************begin_ur_text

And the thyrd syster, Morgan le Fey, was put to scole in a nonnery,
(CMMALORY,5.117)

************************************************************************end_ur_text
```

remove_nodes: (boolean true or false)

default true

removes nodes of the same species as the node boundary, which did *not* contain the searched-for structure.

The purpose of this 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 this algorithm to find the node species: start with the node boundary. If the node boundary contains a hyphen ('-'), the node species is the substring of the node boundary up to the first hyphen, with a '*' tacked on. If the node boundary does not contain a '-', the node species is simply the node boundary with a '*' tacked on if the node boundary didn't already have one.

For instance, if the node boundary is IP-PRN*, the node species is IP*.

For example, consider this command file. Remove_nodes is true by default, and the node boundary is IP* by default, resulting in a node species of IP*:

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

Here's a piece of the output:

```
**********************************************************************begin_comments
 1 IP-MAT-SPE: 8 NP-OB1, 9 PRO the
**********************************************************************end_comments
**********************************************************************begin_ur_text
'And I shall defende the,' seyde the knyght.
(CMMALORY,39.1264)
**********************************************************************end_ur_text
 (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 REMOVED)
(13 E_S .))(ID CMMALORY,39.1264))
```

Notice that the sub-sentence "seyde the knyght" has been removed from the parsed sentence. A search on this output will be a search *only* on IP* nodes that contain a pronoun object, and on no other nodes.

set_margin: (int margin)

default 78

sets margin for CorpusSearch comments and ur_text, but not for parsed sentences, which wrap around the screen.

1.7.5 debugging commands:

The debugging commands are intended for the use of Corpus-Mistresses. The average user probably has no cause to use these commands.

debug_corpus_begin:, debug_corpus_end: (int sentence_number)

default 0

tells CorpusSearch to print (in the output file) the corpus sentences beginning with the begin number and ending with the end number.

For instance, to print sentences number 1 through 10 in the output file, put these lines in your command file:

```
debug_corpus_begin: 1
debug_corpus_begin: 10
```

debug_function_calls: (boolean true or false)

default false

tells CorpusSearch to print the function calls vector to the screen.

debug_report_numbers: (boolean true or false)

default false

reports numbers of sentences being searched. The sentence corresponding to the last number reported may have an error.

hunt_bugs: (boolean true or false)

default false

For use by the Corpus-Mistress. Sends the input files to the bug-hunter, and outputs any errors discovered. The bug-hunter is the one piece of CorpusSearch that is label-dependent.

comments

Comments may be added to the command file using // or /*. Do not add comments after the query!

1.8 Understanding the Output

1.8.1 general form of the output

CorpusSearch output files have the following structure:

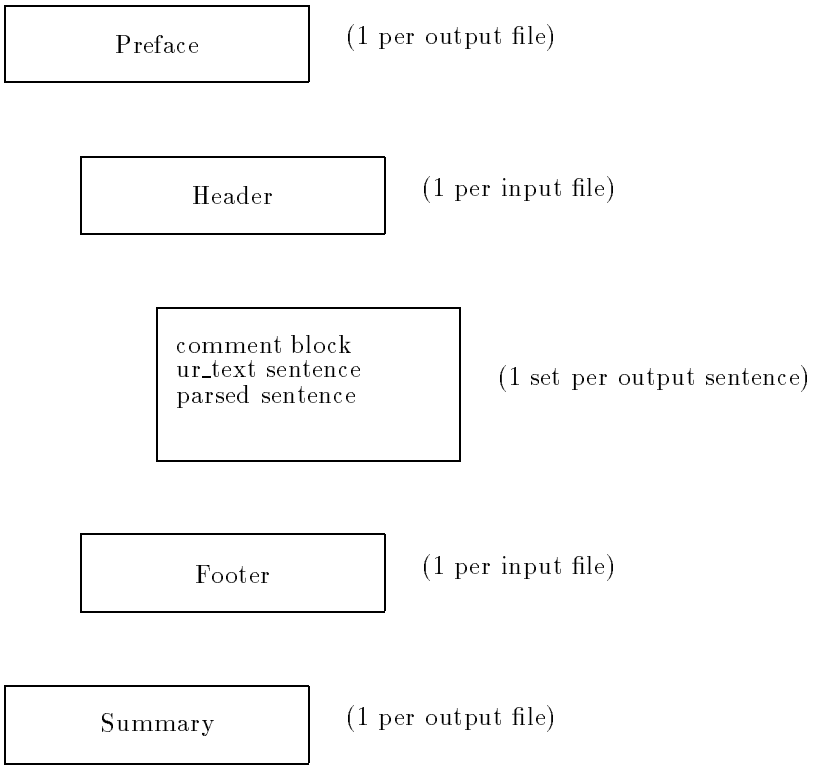

Figure 3: the structure of output files.

Since output files can be used as input to a search, everything that should not be searched (that is, everything that wasn't originally in the corpus) is explicitly labelled. Labels begin with a row of ***s. This includes headers, footers, comment blocks, text versions of corpus sentences (but not parsed and labelled versions!), and summary blocks.

1.8.2 a typical output file

As an example, I'll walk through a typical output file, from a search done by Ann Taylor. 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, these lines were added to the command file:

```
nodes_only: f
remove_nodes: f
```

I will discuss nodes_only and remove_nodes below.

1.8.3 preface

```
**********************************************************************begin_preface

    PREFACE:  regular output file.
    SearchMyCorpus copyright Beth Randall 1999.
    Date:  Sun Sep 12 15:34:42 EDT 1999

    command file:        invert.q
    output file:         invert.out

    remark:  this query searches for inverted pronoun subjects.

    node:  IP*
    query:  (((([1]NP*|ADJP*|ADVP*|PP* iPrecedes [2]*VB*|*HV*|*BE*|*DO*|*MD*)
          AND ([1]NP*|ADJP*|ADVP*|PP* iDominates !\*T*))
          AND ([2]*VB*|*HV*|*BE*|*DO*|*MD* iPrecedes [3]NP-SBJ*))
          AND ([3]NP-SBJ* iDominates PRO|MAN))

**********************************************************************end_preface
```

The preface begins with a label identifying this as a regular output file, that is, not a complement file. This is followed by 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,

```
(([1]NP*|ADJP*|ADVP*|PP* iPrecedes [2]*VB*|*HV*|*BE*|*DO*|*MD*)
          AND ([1]NP*|ADJP*|ADVP*|PP* iDominates !\*T\*))
```

requires a constituent (`NP*|ADJP*|ADVP*|PP*`) which immediately precedes the tensed verb (`*VB*|*HV*|*BE*|*DO*|*MD*`). 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 ([2]*VB*|*HV*|*BE*|*DO*|*MD* iPrecedes [3]NP-SBJ*))
          AND ([3]NP-SBJ* iDominates PRO|MAN))
```

describe the tensed verb (`*VB*|*HV*|*BE*|*DO*|*MD*`) which precedes the subject noun phrase (`NP-SBJ*`), which itself immediately dominates a pronoun (`PRO|MAN`), that is, the subject is a pronoun.

1.8.4 header

```
**********************************************************************begin_header

    HEADER:
    source file:  cmcapchr.m4.psd

**********************************************************************end_header
```

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.

1.8.5 comment block with output sentence

Here's an example of a comment block followed by an output sentence, first presented as the original text, then parsed and labelled as it appears in the corpus:

```
**********************************************************************begin_comments

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

**********************************************************************end_comments

**********************************************************************begin_ur_text

His fadir scheep kepte he ful mekly;
(CMCAPCHR,32.13)

**********************************************************************end_ur_text

(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 ;))
(14 ID CMCAPCHR,32.13))
```

Notice that the default word order would be "He kepte his fadir scheep ful mekly", but in this case the object "his fadir scheep" has been moved to the beginning of the sentence. Since only one constituent can precede the verb, the subject "he" must be moved after the verb "kepte" — that is, subject and verb have been inverted.

The first item in the list of indices and structures is the boundary node (in this case, 1 IP), which fit the "node: " line of the command file. It is followed by a colon to separate it from the rest of the list, which details the structures that correspond to the "query: " line of the command file. The list of indices and structures has been weeded out so that no node is reported more than once.

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.

1.8.6 footer

```
**********************************************************************begin_footer

    FOOTER
    source file:  cmcapchr.m4.psd
    hits found:  220
    sentences containing the hits:  220
    total sentences searched:  4175

**********************************************************************end_footer
```

"hits found" gives the number of hits, or distinct boundary nodes containing the looked-for sentence structure, found in the input file. "sentences containing the hits" gives the number of sentences which contained the hits. The number of hits is always greater than or equal to the number of sentences found in the input file. The number of sentences found in any given input file should not vary from search to search.

1.8.7 summary block

```
**********************************************************************begin_summary

    SUMMARY:  regular output file.

    command file:        invert.q
    output file:         invert.out

    source files, hits, sentences, 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
    grand total hits :  773
    grand total sentences:  771
    grand total sentences searched:  18772

**********************************************************************end_summary
```

The summary, like the preface, is labelled "regular output file" to show that it is not the summary of a complement file.

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 corpus files whose titles contain "m4", meaning they are from the fourth chronological period (1420 — 1500).

1.8.8 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

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 true by default, the output will print only the boundary nodes containing the structure, and irrelevant boundary nodes will be removed. The purpose of this is 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."

```
**********************************************************************begin_comments

 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

**********************************************************************end_comments

**********************************************************************begin_ur_text

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

**********************************************************************end_ur_text

(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 REMOVED)))
                 (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: \**
print_complement: t
query: (NP-OB* iDoms PRO)
```

The "suffirde" sentence shows up again, because it has a pronoun object "you".

```
**********************************************************************begin_comments

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

**********************************************************************end_comments
```

```
**********************************************************************begin_ur_text

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

**********************************************************************end_ur_text

 (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 REMOVED)))
                  (17 E_S .))(ID CMMALORY,35.1106))
```

Notice that the comments 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 add this line to the command files:

```
nodes_only: f
```

When nodes_only is false it makes remove_nodes false automatically.

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

```
**********************************************************************begin_comments

 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

**********************************************************************end_comments

**********************************************************************begin_ur_text

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

```
**********************************************************************end_ur_text

(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 "suffirde" sentence resulting from this search:

```
**********************************************************************begin_comments

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

**********************************************************************end_comments

**********************************************************************begin_ur_text

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

**********************************************************************end_ur_text

 (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 comments 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 counted when remove_nodes was true. This explains why the "remove_nodes: true" version of the search counts fewer objects than the "remove_nodes: false" version of the search.

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

```
**********************************************************************begin_summary

    SUMMARY:  regular output file.

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

    source files, hits, sentences, total:
        CMMALORY                177/176/875
    grand total hits :  177
    grand total sentences:  176
    grand total sentences searched:  875
```

```
**********************************************************************end_summary
```

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

```
**********************************************************************begin_summary

    SUMMARY:  regular output file.

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

    source files, hits, sentences, total:
        CMMALORY                290/249/875
    grand total hits :  290
    grand total sentences:  249
    grand total sentences searched:  875

**********************************************************************end_summary
```

1.9 How to Make Your Corpus Compatible with CorpusSearch

1.9.1 your corpus

With the invention of trainable parsers more corpora are being built. So far, CorpusSearch has been used to search Middle 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.

1.9.2 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.

1.9.3 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.

1.9.4 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.

1.9.5 no dashes preceded by a space

The java StreamTokenizer, which is used to process the input text file, has a few bugs. One of these is that a "-" preceded by a space is presumed to be a minus sign. If it is followed by anything other than a digit ("0", "1" ..., "9"), the Tokenizer chokes. So, `NP-SBJ` where the dash is preceded by a letter, is fine, but `(PUNCT -)`, will cause trouble. Notice that this is a fairly natural way to represent dashes encountered in the text.

It's entirely possible that later versions of java will have fixed this bug, but for now you must find some other way to represent dashes. You might consider changing dashes in the text to `DASH` or `\-`. So either one of these is acceptable: `(PUNCT DASH)` or `(PUNCT \-)`.

1.9.6 number trouble

A bug related to the dash problem is the problem of "." and "0", both of which are interpreted by the java StreamTokenizer as numbers whose value is 0. To distinguish between "." and "0" CorpusSearch looks at the environment surrounding them. If the preceding label was "E_S" (end of sentence), CorpusSearch records a "." If the preceding label was "NUM", CorpusSearch can handle any of these constructions correctly: (NUM .iij.) (this occurs in Middle English), (NUM 0.5), (NUM .8).

You may need to use the "E_S" and "NUM" labels to get "." and "0" handled correctly.

1.9.7 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 ")".

1.9.8 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 (see below). 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))
        (7 VBD waxid)
        (8 ADVP-TMP
                   (9 ADV dayly))
        (10 ADJP
                 (11 ADJR gretter)
                 (12 CONJ and)
                 (13 ADJR gretter))
        (14 E_S .))
(15 ID CMMALORY,5.120))
```

1.9.9 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. The sentence_number is used to keep the statistic "#sentences" in the output. It ensures that several nodes that were printed separately can still be identified as belonging to the same sentence.

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 by the StreamTokenizer.

The current version of CorpusSearch will find the ID node anywhere in the sentence, but the Middle English corpus puts the ID node just after the sentence ending but inside the sentence wrapper (see above). This standard may be enforced in later versions of CorpusSearch, so it would be wise to build your corpus according to it.

1.9.10 give corpus files a standard ending

CorpusSearch expects corpus files to have a standard ending. At the moment, CorpusSearch understands ".psd" (for "parsed") to indicate an original corpus file.

If an input file name does not end with ".psd" it is presumed to be an output file and treated somewhat differently. For instance, when searching output, CorpusSearch uses the ID nodes to keep statistics for the header, footer, and summary blocks. If you see "NO_FILE_ID" listed in the header, footer and summary blocks, it may be because your corpus files don't have names ending with ".psd" and don't contain ID nodes.

1.9.11 the corpus bug-hunter is label-dependent

The only part of CorpusSearch that is dependent on a particular set of labels is the corpus bug-hunter. This is the part of CorpusSearch that responds to errors in the corpus itself (as opposed to, for instance, errors in the query.) When CorpusSearch encounters a corpus error, it sends the

suspicious sentence to the corpus bug-hunter, which prints out an error message followed by the suspicious sentence. If your corpus has a different set of labels than the Middle English corpus, the error message might not be completely appropriate. However, the fact that an error message has appeared means that CorpusSearch found *some* problem with that sentence.

If you have a private copy of CorpusSearch and you're familiar with Java programming, you can try your hand at customizing the list of labels that the corpus bug-hunter responds to. The list is in a class called "Tags.java" and the code is quite straightforward.

1.9.12 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 put them in a separate file.)

```
 (
 [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 leave the node boundary at its default, IP*, 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 (see example command file and output.)

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.

2 CORPUSSEARCH QUICK REFERENCE SHEET

2.1 to run CorpusSearch

for automatic output file (command.out)
java CorpusSearch <command.q> <input-files>
for output file with your choice of name (my_name.out)
java CorpusSearch <command.q> <input-files> -out <my_name.out>

Query file names must end in .q. Output file names must end in .out

2.2 §query components:

search functions:

exists	(exists anywhere in sentence)
precedes	(sister precedes)
iPrecedes	(immediately sister precedes)
anyPrecedes	(precedes anywhere)
dominates	(dominates to any generation)
iDominates	(immediately dominates)
iDomsOnly	(immediately dominates only child)
iDomsNumber	(immediately dominates first, second, etc. child)
iDomsLast	(immediately dominates last, second-to-last, etc. child)
DomsWords#	(dominates # of words)
iDomsTotal#	(dominates # of daughters)

logical operators:

AND	(and search-function calls)
|	(or arguments)
!	(not argument)

wild cards:

	(matches any character)
#	(matches any digit)

2.3 §command-file components:

search commands:

command:	**default:**
query:	no default. must be last item in command file.
node:	*
ignore_nodes:	COMMENT|CODE|ID|LB|'|"|,|E_S
add_to_ignore:	<empty string>

printing commands:

command:	**default:**
print_indices:	true
print_comments:	true
nodes_only:	true
remove_nodes:	true
print_ur_text:	true
only_ur_text:	false
print_complement:	false
print_parsed:	false

3 PPCME2 Labels

3.1 Phrase Labels

ADJP	adjective phrase
ADJP-LOC	locative adjective phrase
ADJP-SPR	adjective phrase secondary predicate
ADJX	adjectival constituent, ambiguous level (ADJ, ADJ', or ADJP)
ADVP	adverb phrase
ADVP-DIR	directional adverb phrase
ADVP-LOC	locative adverb phrase
ADVP-LOC-LFD	left-dislocated locative adverb phrase
ADVP-TMP	temporal adverb phrase
ADVX	adverbial constituent, ambiguous level (ADV, ADV', or ADVP)
CONJP	conjunction phrase
CP-ADV	adverbial clause
CP-CAR	clause-adjoined relative
CP-CLF	it-cleft
CP-CMP	comparative clause
CP-DEG	degree complement
CP-EOP	empty operator complementizer phrase
CP-EXL	exclamation
CP-FRL	free relative
CP-QUE	question (direct or indirect)
CP-QUE-ADV	adverbial WHETHER question
CP-QUE-LFD	left-dislocated indirect question
CP-QUE-SBJ	indirect question subject
CP-REL	relative clause
CP-THT	that clause
CP-THT-LFD	left-dislocated that clause
CP-THT-SBJ	that clause subject
CP-TMC	tough-movement complement
FRAG	sentence fragment
FRENCH	French text
GREEK	Greek text
HEBREW	Hebrew text
INTJP	interjection phrase
IP-ABS	absolute clause
IP-IMP	imperative
IP-INF	complement infinitive
IP-INF-ABS	infinite absolute
IP-INF-ADT	adjunct infinitive
IP-INF-DEG	degree infinitive
IP-INF-LFD	left-dislocated infinitive
IP-INF-PRP	purpose infinitive
IP-INF-SBJ	infinitival subject
IP-MAT	matrix clause
IP-PPL	participial clause
IP-PPL-SBJ	participial clause subject
IP-SMC	small clause
IP-SUB	subordinate clause
LATIN	Latin text
LS	list item

NP	noun phrase
NP-ADT	adjunct noun phrase
NP-ADV	noun phrase adverb
NP-COM	noun phrase complement
NP-DIR	directional noun phrase
NP-DPS	dative of possession
NP-LOC	locative noun phrase
NP-LFD	left-dislocated noun phrase
NP-MSR	measure noun phrase
NP-OB1	first object
NP-OB2	second object
NP-POS	possessive noun phrase
NP-PRN	parenthetical or appositive noun phrase
NP-RFL	reflexive noun phrase
NP-SBJ	noun phrase subject
NP-SPR	noun phrase secondary predicate
NP-TMP	temporal noun phrase
NP-VOC	vocative noun phrase
NPX	nominal constituent, ambiguous level (N, N', or NP)
NUMP	number phrase
PP	prepositional phrase
PP-LFD	left-dislocated prepositional phrase
QP	quantifier phrase
QTP	quotation phrase
QX	quantifier phrase, ambiguous level (Q, Q', or QP)
REF	reference
RRC	reduced relative clause
VP	verb phrase
WADJP	wh- adjective phrase
WADVP	wh- adverb phrase
WNP	wh- noun phrase
WPP	wh- prepositional phrase
WQP	wh- quantifier phrase
X	unknown

3.2 Word Labels

,	non-final sentence punctuation
$	possessive ending
ADJ	adjective
ADJR	adjective, comparative
ADJS	adjective, superlative
ADV	adverb
ADVR	adverb, comparitive
ADVS	adverb, superlative
ALSO	the words ALSO (except when = AS) and EKE
C	complementizer
CODE	non-text material
CONJ	coordinating conjunction
D	determiner
ELSE	the word ELSE (in the collocation OR ELSE)
E_S	end of sentence
EX	existential THERE
FOR	infinitival FOR
FP	focus particle
FW	foreign word
ID	sentence identification
INTJ	interjection
LB	line break
MAN	indefinite subject pronoun (ME, MAN)
N	noun
N$	possessive noun
NEG	negation
NPR	proper noun, singular
NPR$	possessive proper noun
NPRS	proper noun, plural
NPRS$	possessive plural proper noun
NS	common noun, plural
NS$	possessive plural noun
NUM	cardinal number
NUM$	genitive number
ONE	the word ONE (except as focus particle)
ONE$	possessive ONE
OTHER	the word OTHER (except as conjunction)
OTHER$	possessive nominal use of OTHER
OTHERS	plural nominal use of OTHER
OTHERS$	possessive OTHERS
P	preposition or subordinating conjunction
PRO	personal pronoun
PRO$	possessive pronoun
Q	quantifier
Q$	possessive quantifier
QR	quantifier, comparative (MORE, LESS)
QS	quantifier, superlative (MOST, LEAST)
RP	adverbial particle
SUCH	the word SUCH
TO	infinitival TO and AT
WADV	wh-adverb

WARD	the morpheme WARD
WD	wh-determiner
WPRO	wh-pronoun
WPRO$	possessive wh-pronoun
WQ	WHETHER introducing indirect questions

3.3 Word-orPhrase Labels

BAG	present participle BE
BE	infinitive BE
BED	past BE (including past subjunctive)
BEI	imperative BE
BEN	perfect participle BE
BEP	present BE (including present subjunctive)
DAG	present participle DO
DAN	passive participle DO (verbal or adjectival)
DO	infinitive DO
DOD	past DO (including past subjunctive)
DOI	imperative DO
DON	perfect participle DO
DOP	present DO (including present subjunctive)
HAG	present participle HAVE
HAN	passive participle HAVE (verbal or adjectival)
HV	infinitive HAVE
HVD	past HAVE (including past subjunctive)
HVI	imperative HAVE
HVN	perfect participle HAVE
HVP	present HAVE (including present subjunctive)
MD	modal verb
MD0	untensed modal verb
NODE	printed in output when nodes_only is true
VAG	present participle
VAN	passive participle (verbal or adjectival)
VB	infinitive, all other verbs
VBD	past (including past subjunctive)
VBN	perfect participle
VBI	imperative
VBP	present (including present subjunctive)
X	unknown

3.4 Trace Labels

0	empty operator
	unspecified empty constituent
arb*	arbitrary PRO subject in ECM infinitives
con*	subject elided under conjunction
exp*	empty expletive subject
pro*	"small pro" subject
ICH*	non-wh trace
T*	wh-trace

3.5 Suffix Labels

PRN	parenthetical or appositive
RSP	resumptive element
SPE	direct speech
LFD	left-dislocated

"+" joins any two labels when more than one applies, as in (N+N mankind).
"-#" is used to coindex two constituents.
"=#" is used to coindex a clause, part of which has been elided, to the related full clause.
Separated parts of words are indicated as follows:

(ADV (ADV21 to) (ADV22 gether))

where the first number indicates the number of parts and the second number is the index of each part.