LiLT volume 18, issue 4 July 2019

Adding linguistic information to parsed corpora

Susan Pintzuk, *University of York*

Abstract

No matter how comprehensively corpus builders design their annotation schemes, users frequently find that information is missing that they need for their research. In this methodological paper I describe and illustrate five methods of adding linguistic information to corpora that have been morphosyntactically annotated (=parsed) in the style of Penn treebanks. Some of these methods involve manual operations; some are executed by CorpusSearch functions; some require a combination of manual and automated procedures. Which method is used depends almost entirely on the type of information to be added and the goals of the user. Of course the main goal, regardless of method, is to record within the corpus additional information that can be used for analysis and also retained through further searches and data processing.

Introduction

No matter how comprehensively corpus builders design their annotation schemes, users frequently find that information is missing that they need for their research, and so they must add it on their own. In this methodological paper I discuss and illustrate five methods of adding linguistic information of all types (lexical, phonological, morphological, syntactic, semantic, discourse) to corpora that have been morphosyntactically annotated (=parsed) in the style of Penn treebanks, and the advantages and disadvantages of each method. These five methods are the following: 1) adding information to the ur-text; 2) inserting CODE nodes into the token structure; 3) embedding in-

-

formation in coding strings; 4) modifying node labels and structure; and 5) importing token information and other corpus data from the corpus into spreadsheets. Method 1 is necessarily manual, while methods 2 through 5 may involve a combination of manual and automated procedures, functions and tools. Of course the main goal, regardless of method, is to record within the corpus additional information that can be used for analysis and also retained through further searches and data processing. The search engine used for many treebanks, and the one used for the searches and the automated annotation described in this paper, is CorpusSearch (CS).[1]

The manual addition of information may be the simplest procedure but, being manual, it is the most prone to error. Information can be added to the two areas of CS output that are reproduced each time CS is run under default conditions: 1) the token ur-text, which contains the token text and ID without any annotation, and 2) the token structure, including the lexical items. The main difference between the two locations is that material internal to the ur-text is not searchable by CS, while the token structure is the object that is searched and modified by CS queries.

A word of warning is appropriate here: annotation that is added manually cannot be reproduced except by repeating the same manual procedure. Annotation added by CS (i.e. coding strings, structure changes, label changes) can easily be reproduced – unless it is based on annotation that was previously added manually. The availability of automated reproduction is important for three reasons:

1) Files can be lost or damaged. Automated reproduction of annotation is relatively simple; manual reproduction is painful and time-consuming.
2) For most users, whenever we look at the output of a new CS query, we find problems, either in the query or else in the corpus; we then must find and fix the source of the problem and run CS again. One way to facilitate this repetition is to use annotated batch files so that the same processes can be documented and repeated. The use of batch files permits the effortless repetition of what may be a long and complex string of searches. An example of a batch file is given in Appendix.
3) We want other scholars to be able to reproduce our research. With this end in mind, it is encouraging to see that many researchers are making their CS queries available, either in an appendix or

[1]The CS software and manual can be downloaded from its sourceforge site: http://corpussearch.sourceforge.net/.

on the web, along with their search results.

In the remainder of this paper, I describe and evaluate the five methods listed above, presenting case studies for each method from my own recent collaborative research.[2] For readers who are not familiar with CS, some details of the search methodology will be given where space permits; interested readers are referred to the online CS manual. Because of space limitations, the background information and results for each case study are necessarily brief; interested readers are referred to the publications themselves for details and clarifications.

1 Method 1: Adding information manually to the ur-text.

The ur-text consists of the words of the token and the token ID without morphosyntactic annotation; CS outputs the ur-text above the structure for each token in the output file. As mentioned above, adding information manually to the ur-text is arguably the simplest procedure, at least in concept, but it has (at least) three major drawbacks: 1) because it is manual, it is prone to error; 2) it must be applied to CS output, not to the original corpus, because the original corpus does not contain ur-text to accompany the token structure; and 3) the ur-text is not searchable by CS, and therefore any added information can be used only by looking directly at the individual tokens in the data file, one by one.[3] This method was used for some of the tokens in the database for Haeberli et al. 2017, described as Case study 1 below.

Case study 1: Haeberli et al. 2017, investigating verb second (V2) in Old English, looked at fronted pronominal objects to determine whether they can be analyzed as the result of Formal Movement (Frey 2006a,b; Light 2012). CS was used to retrieve all clauses with fronted pronominal objects, but the preceding context was needed to determine the topic type (familiar, aboutness, contrastive, as in Frascarelli and Hinterhölzl 2007). Examples (1) and (2) below show text manually inserted in the ur-text (the text in the area between '/ *' and '* /'). In (1) below, the ur-text is enclosed in a box, and the information added manually is in red. The original Old English token, including the token ID, is in black. In (1), the added information is the preceding context and its gloss and the gloss of the token itself.

[2] In some cases the procedures documented in the case studies have been simplified for clarification purposes.

[3] Of course CS output, being straight ASCII text, can be read by word-processing software such as Word and TextEdit, so these manual additions can be searched for

(1)

```
/~*
Preceding context:
+Ta cw+a+d Iulianus +te +t+at eal wyste . to martine . mid micelre blisse.
Then said Julianus who that all knew . to Martianus . with great joy.

Gang into +tinum godum
Go unto your gods

+te hi clypia+d to him.
You they summon to themselves.
(coaelive,+ALS_[Julian_and_Basilissa]:160.1036)
*~/
( (1 IP-MAT-SPE (2 NP (3 PRO +te))
                (5 NP-NOM (6 PRO^N hi))
                (8 VBPI clypia+d)
                (10 PP (11 P to)
                       (13 NP-DAT-RFL (14 PRO^D him)))
                (16 . .))
  (18 ID coaelive,+ALS_[Julian_and_Basilissa]:160.1036))
```

In (2), the added information is the gloss of the token and a comment about structure and word order in the token.

(2)

```
/~*
Min modor Claudia, me h+af+d gebroht min h+alend Crist to his halgena blysse,
my mother Claudia, me has brought my lord Christ to his holy bliss
(coaelive,+ALS_[Eugenia]:415.445)
Note low subject with transitive verb
*~/
( (1 IP-MAT-SPE (4 NP-NOM-VOC (5 PRO$^N Min) (7 N^N modor)
                              (9 NP-NOM-PRN (10 NR^N Claudia)))
                (12 , ,)
                (14 NP (15 PRO me))
                (17 HVPI h+af+d)
                (19 VBN gebroht)
                (21 NP-NOM (22 PRO$^N min) (24 N^N h+alend)
                           (26 NP-NOM-PRN (27 NR^N Crist)))
                (29 PP (30 P to)
                       (32 NP (33 NP-GEN (34 PRO$ his) (36 N^G halgena))
                       (38 N blysse)))
                (40 . ,))
  (42 ID coaelive,+ALS_[Eugenia]:415.445))
```

Haeberli et al. 2017 used the preceding context to determine the topic type and then manually added it to the coding string for each token. The counts of the different topic types are shown in Table 1 below; it is clear from these data that non-contrastive object pronouns that serve as familiar topics can be found clause-initially in early English,

in this way.

contra Light 2012; Haeberli et al. 2017 showed that this pattern could be analysed as in Walkden 2017.

Topic type	N	%
contrastive	18	14.4%
aboutness	42	33.6%
familiar	65	52.0%
Total	125	100.0%

TABLE 1: Topic type of fronted pronominal objects in Old English main clauses

2 Method 2: Inserting CODE nodes into the token structure

Since inserting nodes is a change in the structure of the token, it can be done at least partially by CS,[4] and the information added can be accessed by CS in subsequent queries.

Case study 2: Crisma and Pintzuk 2016, building on research developed in Crisma 2015, investigated the development of the indefinite article in Middle English. Table 2 lists the codes and their definitions; Middle English examples are given in Appendix B.

(3) Three-step process for coding and counting NP types

Step 1: Add CODE node with a CS corpus-revision query:

Step 1 Query: This query inserts a CODE node as the first constituent of an NP object

```
node: NP-OB*
query: (NP-OB* idomsfirst {1}*)

add_leaf_before{1}: (CODE <NPTYPE:>)
```

[4]In fact, while the research for this project was carried out, both the CODE nodes and their contents were inserted manually; it would have been less work if CS had been used to insert the nodes and to determine whether or not the relevant NP contained the lexeme *an* 'one'. In the procedure presented below, the CODE node is inserted by CS, while all other information is inserted manually.

AN-	*an* 'one' introduces NP
BSG-	bare singular
-EXS	existential
-GNR	generic
-AMB	ambiguous between existential and generic
-NPE	no presupposition of existence (generics and existentials with narrow scope have this characteristic in common)
-SDF	semantically definite (the NP receives a definite interpretation but is not formally marked as definite)
-SCOPE-nrw	narrow scope
-SCOPE-wd	wide scope
-SCOPE-amb	ambiguous scope
-SPC	the NP is overtly made specific (e.g. by a relative clause)
-NG	the clause contains a negative element

TABLE 2: Coding for CODE-NPTYPE node

Step 1 Input:

```
( (IP-MAT (NP-SBJ (NPR Eue))
          (VBD heold)
          (PP (P+NPR iparais))
          (NP-OB1 (ADJ long) (N tale))
          (PP (P wi+d)
              (NP (D +te) (N neddre)))
          (E_S .)) (ID CMANCRIW-1,II.54.519))
```

Step 1 Output:

```
/~*
Eue heold iparais long tale wi+d +te neddre.
(CMANCRIW-1,II.54.519)
*~/
(0  (1 IP-MAT (2 NP-SBJ (3 NPR Eue))
              (5 VBD heold)
              (7 PP (8 P+NPR iparais))
              (10 NP-OB1 (11 CODE <NPTYPE:>)
                         (13 ADJ long)
                         (15 N tale))
              (17 PP (18 P wi+d)
                     (20 NP (21 D +te) (23 N neddre)))
              (25 E_S .))
    (27 ID CMANCRIW-1,II.54.519))
```

Step 2: Add NP characteristics manually to CODE node

```
/~*
Eue heold iparais long tale wi+d +te neddre.
Eve held in-paradise (a) long conversation with the serpent
(CMANCRIW-1,II.54.519)
*~/
(0  (1 IP-MAT (2 NP-SBJ (3 NPR Eue))
              (5 VBD heold)
              (7 PP (8 P+NPR iparais))
              (10 NP-OB1 (11 CODE <NPTYPE:BSG-EXS>)
                         (13 ADJ long)
```

```
                (15 N tale))
          (17 PP (18 P wi+d)
                 (20 NP (21 D +te) (23 N neddre)))
          (25 E_S .))
   (27 ID CMANCRIW-1,II.54.519))
```

In Crisma and Pintzuk 2016, all nominal objects were coded in this way. Each type of NP was then 'counted' by searching for each type of CODE node; an example is given in Step 3 Query below. The quantitative results are shown in Table 3.

Step 3: Count NP types

Step 3 Query:

```
node: IP*
query: (NP-OB* idoms CODE)
AND (CODE idoms <NPTYPE:*GNR*>)
```

		GNR		NPE		EXS-SCOPE-nrw		EXS-SCOPE-amb		EXS		AMB		EXS-SCOPE-wd		EXS-SPC		TOTAL	
ME PERIOD	TEXT	BSG	AN	BSG	AN	BSG	AN	BSG	AN	BSG	AN	BSG	AN	BSG	AN	BSG	AN	BSG	AN
M1	cmpeterb	0	0	2	1	1	0	0	0	1	2	0	0	0	0	0	6	4	9
	cmhali	9	2	12	0	1	0	0	0	1	2	0	0	0	0	0	0	23	4
	cmjulia	2	0	0	1	1	1	0	0	1	11	0	0	0	0	0	0	4	13
	cmkathe	1	0	0	0	1	4	0	0	2	7	0	0	0	1	0	2	4	14
	cmmarga	1	0	2	0	3	3	0	0	1	8	0	0	0	0	0	2	7	13
	cmsawles	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	2	1	3
	cmancriw	54	2	10	8	12	3	0	0	11	5	0	1	0	0	0	2	87	21
	TOTAL M1	68	4	26	11	19	11	0	0	17	35	0	1	0	1	0	14	130	77
M2	cmayenbi	1	2	0	2	0	1	0	0	0	1	0	1	0	0	0	1	1	8
M3(4)	cmctmeli	1	10	2	17	0	6	0	0	0	9	0	4	0	0	0	4	3	50
	cmctpars	5	30	1	33	0	4	0	2	0	8	0	1	0	0	0	3	6	81
	cmvices4	0	7	0	5	0	1	0	0	0	0	0	1	0	0	0	2	0	16
	TOTAL M3(4)	6	47	3	55	0	11	0	2	0	17	0	6	0	0	0	9	9	147
TOTAL		75	53	29	68	19	23	0	2	17	53	0	8	0	1	0	24	140	232

TABLE 3: The distribution of bare singular (BSG) nominals and nominals with an (AN) in Middle English (Crisma and Pintzuk 2016: Table 1)

The texts in Table 3 are arranged in chronological order. The columns are arranged left to right in order of increasing saliency of presupposition of existence: there is no presupposition of existence with GNR, NPE, EXS-SCOPE-nrw; there is a clear presupposition of existence with EXS-SCOPE-wd, EXS-SPC; and finally there is a 'gray' area in the middle: EXS-SCOPE-amb, EXS, AMB.

According to Crisma 2015, *an* develops in three stages in the history of English. In Stage 1, *an* is the numeral 'one'; in Stage 2, *an* is an overt

existential operator used when an indefinite noun phrase is interpreted as specific or when it takes wide scope over another operator; in Stage 3, *an* is an expletive used with all singular noun phrases. Crisma notes that in Stages 1 and 2, *an* is never used with generics.

The numbers are quite small in most of the cells in Table 3; presenting frequencies would be misleading. Nevertheless, clear patterns emerge. We can see that in the M1 period, *an* acts as an overt existential operator in the following types of nominals: 1) indefinite nominals that are interpreted as specific (EXS-SPC: 0 BSG, 14 AN); 2) nominals that take wide scope over some other operator (EXS-SCOPE-wd: 0 BSG, 1 AN). For nominals in the absence of other logical operators, *an* is favoured by about 2 to 1 over BSG (EXS: 17 BSG, 35 AN). For NPE nominals, either generic or narrow scope existential, as well as for existential nominatives taking narrow scope, BSG is favoured by about 2 to 1 over *an* (EXS SCOPE-nrw: 19 BSG, 11 AN; NPE: 26 BSG, 11 AN). In addition, we also see the first sign of change: in two texts, *Ancrene Riwle* and *Hali Meidhad*, there are two examples each of *an* used with generics (GNR).

In the M3 period, we see a number of changes: 1) for generics (GNR), a sharp reversal in the distribution of BSG (6 BSG, 47 AN); 2) for nominals with no presupposition of existence (NPE), there is also a reversal, with only 3 BSG and 55 AN; 3) similarly for existential nominals with narrow scope (EXS-SCOPE-nrw) and existential nominals in the absence of other logical operators (EXS), with all 11 and 17 tokens, respectively, using AN. Our conclusion is that in this period, the use of *an* with singular nouns has generalised to all contexts, with very few exceptions.

3 Method 3: Embedding information in coding strings

Coding strings are strings of characters, each character representing a linguistic or extralinguistic variable, which are inserted as nodes in the tokens of a corpus file. Method 3, the construction of coding strings, is the traditional and perhaps most widely used method of adding information to corpus data. Coding strings had their origin in quantitative sociolinguistic research and were used decades before the creation of parsed corpora. The CODING function of CS is used to construct coding strings based on the morphosyntactic annotation and the lexical content of the token; once created, coding strings may be manually extended to encode information that is not represented in the corpus. Since coding strings are part of the token structure, they may be searched and manipulated by CS. Coding strings may also be used

as input to software for statistical analysis, like R; this is perhaps their most important function.

Case study 3: Taylor and Pintzuk 2015 (T&P 2015) examine the position of objects in Old English and look at the effect of verb order and the length and information structure of the object to support their conclusion that there are two sources for post-verbal objects in Old English, object postposition and base-generation. As shown in (4) below, objects can appear both before and after the verb cluster in clauses with non-finite main verbs before finite auxiliaries (O V Aux and V Aux O), and before and after non-finite verbs in clauses with auxiliary – main verb order (Aux O V and Aux V O). In these examples, finite auxiliaries are underlined, non-finite main verbs are italicised, and objects are in bold face.

(4) a. O V Aux
gif heo **þæt bysmor** *forberan* wolde
if she that disgrace tolerate would
'if she would tolerate that disgrace'
(coaelive,+ALS_[Eugenia]:185.305)

b. V Aux O
þæt he *friðian* wolde **þa leasan wudewan**
that he make-peace-with would the false widow
'that he would make peace with the false widow'
(coaelive,+ALS_[Eugenia]:209.315)

c. Aux O V
þurh þa heo sceal **hyre scippend** *understandan*
through which it must its creator understand
'through which it must understand its creator'
(coaelive,+ALS_[Christmas]:157.125)

d. Aux V O
swa þæt heo bið *forloren* **þam ecan life**
so that it is lost the eternal life
'so that it is lost to the eternal life'
(coaelive,+ALS_[Christmas]:144.117)

Table 4 shows the distribution of objects in Old English texts that have more than 100 clauses with finite auxiliaries, non-finite main verbs

[5]There are additional constraints on the object; see T&P 2015.

and non-pronominal objects.[5]

Text	**VAux**		**AuxV**	
	N	**%VO**	**N**	**%VO**
Orosius	66	4.5	47	31.9
Bede	58	6.9	46	10.9
Boethius	74	8.1	49	53.1
Cura Pastoralis	51	21.6	72	55.6
Catholic Homilies I	49	10.2	95	47.4
Catholic Homilies II	42	7.1	80	46.3
Lives of Saints	33	45.5	91	62.6
Gregory's Dialogues (C)	36	27.8	66	68.2
total	409	13.9	546	49.5

TABLE 4: Frequency of VO order by verb order in texts with more than 100 tokens
(T&P 2015, Table 1)

T&P 2015 present the following analysis of these data. They assume that in the Old English period, there was variation in underlying structure: head-initial/final IPs (AuxV/VAux) and VPs (VO/OV). V Aux O can be derived only from head-final IP/VP structure by postposition of O from preverbal position, as shown in (5)a-b. In contrast, Aux V O order can be derived in two different ways: a) head-initial IP, head-final VP structure with postposition of O, as shown in (5)c-d; b) head-initial IP/VP structure (i.e. O is merged in post-verbal position), as shown in (5)e.

(5) a. O V Aux: $[_{TP}$ ... $[_{T'}$ $[_{VP1}$ $[_{VP2}$ O V] t_{Aux}] Aux+T]]
b. V Aux O: $[_{TP}$ ... $[_{T'}$ $[_{VP1}$ $[_{VP2}$ t_{O} V] t_{Aux}] Aux+T] O]
c. Aux O V: $[_{TP}$ ... $[_{T'}$ Aux+T $[_{VP1}$ t_{Aux} $[_{VP2}$ O V]]]]
d. Aux V O: $[_{TP}$ ... $[_{T'}$ Aux+T $[_{VP1}$ t_{Aux} $[_{VP2}$ t_{O} V]]] O]
e. Aux V O: $[_{TP}$ ... $[_{T'}$ Aux+T $[_{VP1}$ t_{Aux} $[_{VP2}$ V O]]]]

If all post-verbal objects were derived by postposition in both V Aux and Aux V clauses, i.e. if structure (5)e didn't exist, we would expect the factors influencing post-verbal position to be the same in both clause types. To test this null hypothesis, T&P 2015 looked at the influence of weight (as measured by the length of the object in words) and informational status (given vs. new) on the position of objects in AuxV and VAux clauses. This was a four-step process. As a first step, CS was used to code each token for three factors: the order of finite

auxiliary and non-finite main verb (auxv vs. vaux); the position of the object with respect to the non-finite main verb (ov vs. vo); the length of the object in words (1 ... 11). The coding query file is given below in (6);[6] an example of a token coded for the first three factors is given in (7).

(6) **Coding Query:**

```
node: IP*

coding_query:

1: {
auxv: (IP* idoms verb-finite)
      AND (IP* idoms verb-non-finite)
      AND (verb-finite precedes verb-non-finite)
vaux: (IP* idoms verb-finite)
      AND (IP* idoms verb-non-finite)
      AND (verb-non-finite precedes verb-finite)
1x: ELSE[7]
}

2: {
ov: (IP* idoms verb-non-finite)
    AND (IP* idoms oblique-argument)
    AND (oblique-argument precedes verb-non-finite)
vo: (IP* idoms verb-non-finite)
    AND (IP* idoms oblique-argument)
    AND (verb-non-finite precedes oblique-argument)
2x: ELSE
}

3: {
\1: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 1)
\2: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 2)
\3: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 3)
\4: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 4)
\5: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 5)
\6: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 6)
\7: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 7)
\8: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 8)
\9: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 9)
\10: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords 10)
\11: (IP* idoms oblique-argument)
    AND (oblique-argument idomswords> 10)
3x: ELSE
```

[6] The terms 'verb-finite', 'verb-non-finite', and 'oblique-argument' are defined in a condition file, available to CS when queries are run.

```
}
```

(7)

```
/~*
+t+at we god  don sceolon
that  we good do  should
(coaelive,+ALS_[Auguries]:245.3644)
*~/

(0 (1 IP-SUB-SPE (2 CODING vaux:ov:1)
                 (4 NP-NOM (5 PRO^N we))
                 (7 NP-ACC (8 N^A god))
                 (10 VB don)
                 (12 MDPI sceolon))
   (14 ID coaelive,+ALS_[Auguries]:245.3644))
```

The second step was to manually code the informational status of the object (given vs. new). Examples of tokens coded for all four factors are given in (8) through (11); (8) is the token in (7) with informational status added as the fourth factor.

(8)

```
/~*
+t+at we god  don sceolon
that  we good do  should
(coaelive,+ALS_[Auguries]:245.3644)
*~/

(0 (1 IP-SUB-SPE (2 CODING vaux:ov:1:new)
                 (4 NP-NOM (5 PRO^N we))
                 (7 NP-ACC (8 N^A god))
                 (10 VB don)
                 (12 MDPI sceolon))
   (14 ID coaelive,+ALS_[Auguries]:245.3644))
```

(9)

```
/~*
Gif +du +t+as ecan    lifes  ges+al+te habban wylt
If  you the   eternal life's happiness have   will
(coaelive,+ALS_[Alban]:65.4038)
*~/

(0 (1 IP-SUB-SPE (2 CODING vaux:ov:4:new)
                 (4 NP-NOM (5 PRO^N +du))
                 (7 NP (8 NP-GEN (9 D^G +t+as) (11 ADJ^G ecan)
                                 (13 N^G lifes))
                       (15 N ges+al+te))
                 (17 HV habban)
                 (19 MDPI wylt))
   (21 ID coaelive,+ALS_[Alban]:65.4038))
```

[7] An elsewhere condition is frequently used to find unexpected conditions and data in the corpus.

(10)

```
/~*
+t+at se  l+ace sceolde asceotan +t+at geswell
that  the leech should  lance    the   tumour
(coaelive,+ALS_[+Athelthryth]:61.4177)
*~/

(0 (1 IP-SUB (2 CODING auxv:vo:2:given)
             (4 NP-NOM (5 D^N se) (7 N^N l+ace))
             (9 MDD sceolde)
             (11 VB asceotan)
             (13 NP-ACC (14 D^A +t+at) (16 N^A geswell)))
   (18 ID coaelive,+ALS_[+Athelthryth]:61.4177))
```

(11)

```
/~*
for +tan +te he sylf    sceal sw+arran witu     +trowian
because      he himself shall heavier  torments suffer
(coaelive,+ALS_[Vincent]:89.7849)
*~/

(0 (1 IP-SUB-SPE (2 CODING auxv:ov:2:new)
          (4 NP-NOM (5 PRO^N he)
                    (7 ADJP-NOM (8 ADJ^N sylf)))
          (10 MDPI sceal)
          (12 NP-ACC (13 ADJR^A sw+arran) (15 N^A witu))
          (17 VB +trowian))
   (19 ID coaelive,+ALS_[Vincent]:89.7849))
```

The third step was to use the print_only function of CS to create an output file containing only the coding strings of the data file. The file is shown in (12) below. CS separates the factors by ':', and the user must manually insert a header naming the factors for input to statistical processing, the last step. The results for this study are shown in Table 5.

(12) file of coding strings

```
AUXV|VAUX:OBJ_POS:OBJ_LEN:OBJ_IS
vaux:ov:1:new
vaux:ov:4:new
auxv:vo:2:given
auxv:ov:2:new
...
```

As shown in Table 5, the effect of weight is significant in both clause types, but slightly weaker in AuxV clauses: each additional word in VAux clauses increases the likelihood of VO order by 2.68, in AuxV clauses by 2.43. Informational status is significant only in VAux clauses: the distance between given and new is .9 in VAux clauses, but only .08 in AuxV clauses. T&P 2015 interpret these results as follows: VAuxO clauses are derived only by postposition of the object, and postposition is strongly influenced by weight and informational status: heavy objects

Variable	Level	V Aux O		Aux V O	
		Log odds	Odds ratio	Log odds	Odds ratio
weight	per additional word	0.99	2.68	0.89	2.43
info status	given	-0.45	0.64	-0.04	0.96
	new	0.45	1.57	0.04	1.04

VAuxO: C = 0.88; AuxVO: C = 0.78

TABLE 5: Results of multivariate analysis, effects in log odds and odds ratios. Shaded cells indicate non-significant results (from T&P 2015, Table 5)

and new objects are much more likely to postpose than light objects and given objects. Since AuxVO clauses are derived by two different processes, postposition and base-generation, the effects of weight and informational status are weakened; this is why the effect of weight is weaker in AuxV clauses and the effect of informational status is reduced to non-significance.

4 Method 4: Modifying corpus annotation (node labels and structure)

Method 4, the modification of corpus annotation, may be done manually, but it is much more efficient (and safer) to use the corpus-revision tool of CS. This tool enables the addition, deletion, and modification of annotation in the corpus, including not only node labels but also structure. Any search that can be made using CS can act as the basis for corpus revision; the output of corpus revision is a new version of the corpus – i.e., the original version of the corpus is not deleted, in case of catastrophic errors. Corpus revision can be used to build an annotated corpus starting from a straight text corpus with only part-of-speech. I frequently find it useful to mark particular structures so that they are easy to identify, and also to evade some CS restrictions, as will be seen in Case study 4 below.

Case study 4: Haeberli and Pintzuk 2017 (H&P) look at verb placement in 'true V2' contexts in Old English. H&P analyse in detail one particular clause type: clauses with an initial *gif/þa/þonne* 'if/when/when' subordinate clause, followed by a resumptive adverb (e.g. *þa/þonne* 'then') and the rest of the main clause; an example is given in (13). Note that initial *þa/þonne* in Old English main clauses is considered a 'true V2' context: 97.4% (6546/6719) of these clauses exhibit strict V2 order, with the verb in second position followed by the subject.

(13) [IP-MAT [CP-ADV þa When [IP-SUB þæt that Placidas geseah]], þa Placidas saw, then
gewilnode he wanted he [CP-THT þæt that [IP-SUB he hine gefenge]]]
he it took

'When Placidas saw that, then he wanted to take it'
(coeust,LS_8_[Eust]:31.26)

In order to simplify the searches for and coding of these clauses, I wanted to flag the relevant IP-MATs and CP-ADVs by modifying the label. In addition, I wanted to 'remove'[8] the subtrees of all subordinate clauses other than the IP-SUB dominated by the relevant CP-ADV. Three steps were necessary, as shown below; a red font is used for highlighting.

Step 1: Flag the relevant IP-MAT, CP-ADV, and IP-SUB using the query file below. The IP-MAT and CP-ADV are flagged by appending '-z' to the label; the IP-SUB is flagged by prepending 'x-' to the label. Notice that the first token (coeust ... 26) contains an IP-SUB that is not dominated by the clause-initial CP-ADV; the second token (cobede ... 3530) contains an IP-MAT that does not dominate a CP-ADV as the first constituent; and the third token (cocanedgX ... 82) contains both a CP-ADV that is not the first constituent of the IP-MAT and an IP-SUB that is not dominated by the relevant CP-ADV. These are all nodes that are irrelevant to the investigation.

Step 1 Query:

```
node: IP-MAT*
nodes_only: t
remove_nodes: f
query: ({1}IP-MAT* idoms {2}CP-ADV*)
AND (CP-ADV* idoms P)
AND (P idoms GIF|THA|THONNE[9])
AND (CP-ADV* idoms {3}IP*SUB*)
...
append_label{1}: -z
append_label{2}: -z
prepend_label{3}: x-
```

[8]Setting the CS command-file option 'remove_nodes' to true results in removing recursive structure by removing the subtrees of all nodes that are a) embedded within an instance of the 'node' setting and b) are of the same type as the 'node' setting. The subtrees are replaced by the label 'RMV' and the first three lexical words of the node.

[9]GIF, THA and THONNE are defined in a condition file as all of the variant spellings of these lexemes.

Step 1 Input:

```
(0  (1 IP-MAT (2 CP-ADV (3 P +Ta)
              (5 C 0)
              (7 IP-SUB (8 NP-ACC (9 D^A +t+at))
                    (11 NP-NOM (12 NR^N Placidas))
                    (14 VBDI geseah)))
              (16 , ,)
              (18 ADVP-TMP (19 ADV^T +ta))
              (21 VBD gewilnode)
              (23 NP-NOM (24 PRO^N he))
              (26 CP-THT (27 C +t+at)
              (29 IP-SUB (30 NP-NOM (31 PRO^N he))
                         (33 NP-ACC (34 PRO^A hine))
                         (36 VBDS gefenge)))
              (38 . ,))
    (40 ID coeust,LS_8_[Eust]:31.26))

( (3 IP-MAT (4 CP-ADV (5 P +Ta)
            (7 C 0)
            (9 IP-SUB (10 NP-NOM (11 PRO^N he))
                      (13 ADVP-TMP (14 ADV^T +da))
                      (16 NP (17 PRO$ his) (19 N scylde))
                      (21 VBD gehyrde)))
            (23 , ,)
            (25 ADVP-TMP (26 ADV^T +ta))
            (28 VBDI cw+a+d)
            (30 NP-NOM (31 PRO^N he))
            (33 , :)
            (35 IP-MAT-SPE (36 NP-NOM (37 Q^N Micel) (39 N^N wund))
                           (41 VBPI behofa+d)
                           (43 NP-GEN (44 Q^G micles) (46 N^G l+acedomes)))
            (48 . :))
  (50 ID cobede,Bede_4:26.350.19.3530))

( (3 IP-MAT (4 CP-ADV (5 P Gyf)
            (7 C 0)
            (9 IP-SUB (10 NP-NOM (11 PRO^N he))
                      (13 NP-ACC (14 PRO$ his) (16 N^A lif))
                      (18 VBPS misfadige)))
            (20 , ,)
            (22 VBPS wanige)
            (24 NP-NOM (25 PRO$ his) (27 N^N wyr+dscipe))
            (29 CP-ADV (30 P be)
                       (32 D^D +dam)
                       (34 C +te)
                       (36 IP-SUB (37 NP-NOM (38 D^N seo))
                                  (40 ADJP-NOM-PRD (41 ADJ^N d+ad))
                                  (43 BEPS sy)))
            (45 . .))
  (47 ID cocanedgX,WCan_1.1.2_[Fowler]:68.82))
```

Step 1 Output:

```
(0  (1 IP-MAT-z (2 CP-ADV-z (3 P +Ta)
                (5 C 0)
                (7 x-IP-SUB (8 NP-ACC (9 D^A +t+at))
                            (11 NP-NOM (12 NR^N Placidas))
                            (14 VBDI geseah)))
                (16 , ,)
                (18 ADVP-TMP (19 ADV^T +ta))
                (21 VBD gewilnode)
                (23 NP-NOM (24 PRO^N he))
```

```
                (26 CP-THT (27 C +t+at)
                (29 IP-SUB (30 NP-NOM (31 PRO^N he))
                           (33 NP-ACC (34 PRO^A hine))
                           (36 VBDS gefenge)))
                (38 . ,))
    (40 ID coeust,LS_8_[Eust]:31.26))

( (1 IP-MAT-z (2 CP-ADV-z (3 P +Ta)
              (5 C 0)
              (7 x-IP-SUB (8 NP-NOM (9 PRO^N he))
                          (11 ADVP-TMP (12 ADV^T +da))
                          (14 NP (15 PRO$ his) (17 N scylde))
                          (19 VBD gehyrde)))
              (21 , ,)
              (23 ADVP-TMP (24 ADV^T +ta))
              (26 VBDI cw+a+d)
              (28 NP-NOM (29 PRO^N he))
              (31 , :)
              (33 IP-MAT-SPE (34 NP-NOM (35 Q^N Micel) (37 N^N wund))
                             (39 VBPI behofa+d)
                             (41 NP-GEN (42 Q^G micles)
                                        (44 N^G l+acedomes)))
              (46 . :))
  (48 ID cobede,Bede_4:26.350.19.3530))

( (1 IP-MAT-z (2 CP-ADV-z (3 P Gyf)
                          (5 C 0)
                          (7 x-IP-SUB (8 NP-NOM (9 PRO^N he))
                                      (11 NP-ACC (12 PRO$ his)
                                      (14 N^A lif))
              (16 VBPS misfadige)))
              (18 , ,)
              (20 VBPS wanige)
              (22 NP-NOM (23 PRO$ his) (25 N^N wyr+dscipe))
              (27 CP-ADV (28 P be)
                         (30 D^D +dam)
                         (32 C +te)
                         (34 IP-SUB (35 NP-NOM (36 D^N seo))
                                    (38 ADJP-NOM-PRD (39 ADJ^N d+ad))
                                    (41 BEPS sy)))
              (43 . .))
  (45 ID cocanedgX,WCan_1.1.2_[Fowler]:68.82))
```

Step 2: Remove the nodes of the embedded IPs using the query file below; the output from Step 1 serves as the input to Step 2. Note that the nodes of the IP-SUB embedded under CP-ADV-z are not removed since its label begins with 'x-'.

Step 2 Query:

```
node: IP-MAT*-z
remove_nodes: t
query: (IP-MAT*-z exists)
```

Step 2 Output:

```
( (1 IP-MAT-z (2 CP-ADV-z (3 P +Ta)
                          (5 C 0)
                          (7 x-IP-SUB (8 NP-ACC (9 D^A +t+at))
                                      (11 NP-NOM (12 NR^N Placidas))
```

```
                                    (14 VBDI geseah)))
              (16 , ,)
              (18 ADVP-TMP (19 ADV^T +ta))
              (21 VBD gewilnode)
              (23 NP-NOM (24 PRO^N he))
              (26 CP-THT (27 C +t+at)
                         (29 IP-SUB RMV:he_hine_gefenge...))
              (38 . ,))
  (40 ID coeust,LS_8_[Eust]:31.26))

( (1 IP-MAT-z (2 CP-ADV-z (3 P +Ta)
                          (5 C 0)
                          (7 x-IP-SUB (8 NP-NOM (9 PRO^N he))
                                                (11 ADVP-TMP (12 ADV^T +da))
                                                (14 NP (15 PRO$ his) (17 N scylde))
                                                (19 VBD gehyrde)))
              (21 , ,)
              (23 ADVP-TMP (24 ADV^T +ta))
              (26 VBDI cw+a+d)
              (28 NP-NOM (29 PRO^N he))
              (31 , :)
              (33 IP-MAT-SPE RMV:Micel_wund_behofa+d...)
              (46 . :))
  (48 ID cobede,Bede_4:26.350.19.3530))

( (1 IP-MAT-z (2 CP-ADV-z (3 P Gyf)
                          (5 C 0)
                          (7 x-IP-SUB (8 NP-NOM (9 PRO^N he))
                                      (11 NP-ACC (12 PRO$ his)
                                                 (14 N^A lif))
                                      (16 VBPS misfadige)))
              (18 , ,)
              (20 VBPS wanige)
              (22 NP-NOM (23 PRO$ his) (25 N^N wyr+dscipe))
              (27 CP-ADV (28 P be)
                         (30 D^D +dam)
                         (32 C +te)
                         (34 IP-SUB RMV:seo_d+ad_sy...))
              (43 . .))
  (45 ID cocanedgX,WCan_1.1.2_[Fowler]:68.82))
```

Step 3: Remove 'x-' prepended to IP-SUB labels using the query file below; the output from Step 2 serves as the input to Step 3.

Step 3 Query:

```
node: IP-MAT*-z
query: (1x-IP* exists)

pre_crop_label1: -
```

Step 3 Output:

```
( (1 IP-MAT-z (2 CP-ADV-z (3 P +Ta)
                          (5 C 0)
                          (7 IP-SUB (8 NP-ACC (9 D^A +t+at))
                                    (11 NP-NOM (12 NR^N Placidas))
                                    (14 VBDI geseah)))
              (16 , ,)
              (18 ADVP-TMP (19 ADV^T +ta))
              (21 VBD gewilnode)
```

```
              (23 NP-NOM (24 PRO^N he))
              (26 CP-THT (27 C +t+at)
                         (29 IP-SUB RMV:he_hine_gefenge...))
              (31 . ,))
  (33 ID coeust,LS_8_[Eust]:31.26))

( (1 IP-MAT-z (2 CP-ADV-z (3 P +Ta)
                          (5 C 0)
                          (7 IP-SUB (8 NP-NOM (9 PRO^N he))
                                    (11 ADVP-TMP (12 ADV^T +da))
                                    (14 NP (15 PRO$ his) (17 N scylde))
                                    (19 VBD gehyrde)))
              (21 , ,)
              (23 ADVP-TMP (24 ADV^T +ta))
              (26 VBDI cw+a+d)
              (28 NP-NOM (29 PRO^N he))
              (31 , :)
              (33 IP-MAT-SPE RMV:Micel_wund_behofa+d...)
              (35 . :))
  (37 ID cobede,Bede_4:26.350.19.3530))

( (1 IP-MAT-z (2 CP-ADV-z (3 P Gyf)
              (5 C 0)
              (7 IP-SUB (8 NP-NOM (9 PRO^N he))
                        (11 NP-ACC (12 PRO$ his)
                                   (14 N^A lif))
                        (16 VBPS misfadige)))
              (18 , ,)
              (20 VBPS wanige)
              (22 NP-NOM (23 PRO$ his) (25 N^N wyr+dscipe))
              (27 CP-ADV (28 P be)
                         (30 D^D +dam)
                         (32 C +te)
                         (34 IP-SUB RMV:seo_d+ad_sy...))
              (36 . .))
  (38 ID cocanedgX,WCan_1.1.2_[Fowler]:68.82))
```

The result of this three-step procedure is a file of tokens that are simple to search further and code, since the relevant IP-MATs and CP-ADVs have labels that end in '-z' and the relevant IP-SUB is the only IP-SUB in the token that has its content preserved intact.

5 Method 5: Copying coding strings from the corpus into a spreadsheet

Finally, Method 5 copies coding strings from the corpus into a spreadsheet, the content of which may be ordered, manipulated, and displayed in ways that corpus data cannot be. For example, the data in the cells of a spreadsheet can be interpreted as numbers and used for simple calculations like totals, means, and frequencies; in contrast, the content of coding strings within a corpus are characters, not numerical values, and cannot be used as numbers. From the spreadsheet users can create output, e.g. a csv file, that is formatted for statistical analysis. Method 5 provides perhaps the most flexible way of working with and analyzing corpus data, but it should be used with caution, for at least two obvi-

ous reasons: it involves manual manipulation of the data, and therefore is prone to error; in addition, it is not always possible to go from a spreadsheet back to a corpus format.

Case study 5: Taylor and Pintzuk 2017 (T&P 2017) look at the effect of weight, among other variables, on split coordination in Old English. Almost all coordinated constituents in early stages of English can be split, as illustrated in (14).

(14) a. DP subject
oðþæt **þæt ad** wæs forburnen, **and ealle þa tunnan**
until the pile was burned and all the casks

'until the pile and all the casks were burned up'
(coaelive,+ALS_[Julian_and_Basilissa]:332.1143)

b. DP object
God sende ða **fyr** on merigen **and fulne swefel**
God sent then fire in morning and foul brimstone
him to
him to

'God then sent fire and foul brimstone to him in the morning'
(coaelive,+ALS[Pr_Moses]:211.2976)

c. PP
& **on sorhge** leofodon **&** **on geswincum** siþþan
and in grief lived and in torment afterwards

'and [they] lived afterwards in grief and torment'
(colsigewZ,+ALet_4_[SigeweardZ]:117.49)

T&P 2017 focus on subjects, aiming to measure the effect of length (as measured in number of words) on splitting. They need the length of the first conjunct, the length of the second conjunct (which includes both the conjunction and the nominal), and the length of the entire coordinated nominal in order to determine which of the three, if any, has an effect on splitting. But because of the way coordination is annotated in the corpus, these measurements are not at all straightforward.

If the nominal is not split and the coordinated nouns are bare, with no modification, then the nominal is annotated as a flat structure, with the two nouns and the conjunction immediately dominated by the NP, as shown in (15). In these cases the length of the entire subject can be measured, since it is a constituent (NP-NOM). Although the lengths of the two conjuncts can't be measured individually, since they are not

constituents, it can be assumed that the length of the first conjunct is 1. [10] The tokens in this section are coded as follows: file name : token number : flat vs. non-flat : split vs. non.split : final vs. non.final (position within the clause) : length of 1st conjunct : length of 2nd conjunct: length of entire conjoined phrase. '/' is used when it is not possible to measure or assume the length.

(15) non-split flat structure
Þær is **wop and wanung**
There is weeping and wailing

(coaelive,+ALS_[Sebastian]:77.1254)

```
(0 (1 IP-MAT-SPE (2 CODING aelive:1254:flat:non.split:final:1:/:3)
                 (4 ADVP-LOC (5 ADV^L +T+ar))
                 (7 BEPI is)
                 (9 NP-NOM (10 N^N wop) (12 CONJ and) (14 N^N wanung)))
   (16 ID coaelive,+ALS_[Sebastian]:77.1254))
```

In split flat structures, we can measure the length of the 1st conjunct and the length of the 2nd conjunct, since each of these is a constituent; but we cannot measure the length of the entire subject. An example is given in (16).

(16) split 'flat' structure
ac **foxunga** wæron wunigende on him, **and upahefednys**
but foxlike-wiles were dwelling in him, and haughtiness,
swilce healice fugelas,
like soaring birds,

'but foxlike wiles and haughtiness were dwelling in him, like soaring birds'
(coaelive,+ALS_[Memory_of_Saints]:160.3418)

[10] The reader might think that the length of the 2nd conjunct could be estimated as 2 (conjunction + noun). However, some conjoined constituents have more than two conjuncts, as shown below in (i); in these cases, the length of the 2nd and following conjuncts cannot be assumed or measured.

(i)

```
(CP-ADV (P Gif)
(C 0)
(IP-SUB (NP-NOM (N^N fot) (CONJ o+d+de) (N^N cneow)
                (CONJ o+d+de) (N^N scancan))
        (VBPS swellan))
...
```

```
(0 (1 IP-MAT (2 CODING aelive:3418:flat:split:non.final:1:2:/)
                (4 CONJ ac)
                (6 NP-NOM (7 N^N foxunga)
                          (9 CONJP *ICH*-1))
                (11 BEDI w+aron)
                (13 VAG wunigende)
                (15 PP (16 P on)
                       (18 NP-DAT (19 PRO^D him)))
                (21 , ,)
                (23 CONJP-1 (24 CONJ and) (26 N^N upahefednys))
                (28 PP (29 P swilce)
                       (31 CPX-CMP (32 IPX-SUB RMV:healice_fugelas...)))
                (34 . ,))
   (36 ID coaelive,+ALS_[Memory_of_Saints]:160.3418))
```

In non-split non-flat structures, we can measure all three, since the two conjuncts and also the entire conjoined structure is a constituent, as shown in (17).

(17) non-split hierarchical structure
Eubolus se uðwyta and þa yldostan preostas stoden
Eubolus the philosopher and the principal priests stood
æt þæra dura
at the door ...

(coaelive,+ALS_[Basil]:132.537)

```
(0 (1 IP-MAT (2 CODING aelive:537:non.flat:non.split:non.final:3:4:7)
             (4 NP-NOM (5 NP-NOM (6 NR^N Eubolus)
                                 (8 NP-NOM-PRN (9 D^N se)
                                               (11 N^N u+dwyta)))
             (13 CONJP (14 CONJ and)
                       (16 NP-NOM (17 D^N +ta)
                                  (19 ADJS^N yldostan)
                                  (21 N^N preostas))))
             (23 VBDS stoden)
             (25 PP (26 P +at)
             (28 NP-DAT (29 D^D +t+ara) (31 N^D dura)))
             ...
   (48 ID coaelive,+ALS_[Basil]:132.537))
```

And finally, in split hierarchical structures, we can measure the length of the 1st conjunct and the length of the 2nd conjunct, since each of these is a constituent; but we cannot measure the length of the entire subject. An example is given in (18).

(ID colacnu,Med_3_[Grattan-Singer]:85.1.454))(CP-ADV (P Gif)

(18) split hierarchical structure
oðþæt **þæt ad** wæs forburnen, **and ealle þa tunnan**
until the pile was burned-up, and all the tuns

(coaelive,+ALS_[Julian_and_Basilissa]:332.1143)

```
(0 (1 IP-SUB (2 CODING aelive:1143:non.flat:split:final:2:4:/)
             (4 NP-NOM (5 NP-NOM (6 D^N +t+at) (8 N^N ad))
                       (10 CONJP *ICH*-1))
             (12 BEDI w+as)
             (14 VBN forburnen)
             (16 , ,)
             (18 CONJP-1 (19 CONJ and)
                         (21 NP-NOM (22 Q^N ealle)
                                    (24 D^N +ta) (26 N^N tunnan))))
   (28 ID coaelive,+ALS_[Julian_and_Basilissa]:332.1143))
```

Once the tokens are coded in this way, CS can be used to output the coding strings to a file as shown below:

Query:

```
node: $ROOT[11]
print_only: CODING*
```

Input:

```
(0 (1 IP-MAT-SPE (2 CODING aelive:1254:flat:non.split:final:1:/:3)
                 (4 ADVP-LOC (5 ADV^L +T+ar))
                 (7 BEPI is)
                 (9 NP-NOM (10 N^N wop) (12 CONJ and) (14 N^N wanung)))
   (16 ID coaelive,+ALS_[Sebastian]:77.1254))

(0 (1 IP-MAT (2 CODING aelive:3418:flat:split:non.final:1:2:/)
             (4 CONJ ac)
             (6 NP-NOM (7 N^N foxunga)
                       (9 CONJP *ICH*-1))
             (11 BEDI w+aron)
             (13 VAG wunigende)
             (15 PP (16 P on)
                    (18 NP-DAT (19 PRO^D him)))
             (21 , ,)
             (23 CONJP-1 (24 CONJ and) (26 N^N upahefednys))
             (28 PP (29 P swilce)
                    (31 CPX-CMP (32 IPX-SUB RMV:healice_fugelas...)))
             (34 . ,))
   (36 ID coaelive,+ALS_[Memory_of_Saints]:160.3418))
```

[11] The node `$ROOT` is used to refer to the highest node in the token.

```
(0 (1 IP-MAT (2 CODING aelive:537:non.flat:non.split:non.final:3:4:7)
             (4 NP-NOM (5 NP-NOM (6 NR^N Eubolus)
                                 (8 NP-NOM-PRN (9 D^N se)
                                               (11 N^N u+dwyta)))
                       (13 CONJP (14 CONJ and)
                                 (16 NP-NOM (17 D^N +ta)
                                            (19 ADJS^N yldostan)
                                            (21 N^N preostas))))
             (23 VBDS stoden)
             (25 PP (26 P +at)
             (28 NP-DAT (29 D^D +t+ara) (31 N^D dura)))
             ...
   (48 ID coaelive,+ALS_[Basil]:132.537))

(0 (1 IP-SUB (2 CODING aelive:1143:non.flat:split:final:2:4:/)
             (4 NP-NOM (5 NP-NOM (6 D^N +t+at) (8 N^N ad))
                       (10 CONJP *ICH*-1))
             (12 BEDI w+as)
             (14 VBN forburnen)
             (16 , ,)
             (18 CONJP-1 (19 CONJ and)
                         (21 NP-NOM (22 Q^N ealle)
                                    (24 D^N +ta) (26 N^N tunnan))))
   (28 ID coaelive,+ALS_[Julian_and_Basilissa]:332.1143))
```

Output:

```
aelive:1254:flat:non.split:final:1:/:3
aelive:3418:flat:split:non.final:1:2:/)
aelive:537:non.flat:non.split:non.final:3:4:7)
aelive:1143:non.flat:split:final:2:4:/)
```

The output file can then be read into a spreadsheet, as shown below in Excel spreadsheet 1, with the header ("TEXT | TOKEN | (NON)FLAT | ...") added manually as the first row. When the data are sorted by the missing length, as shown in spreadsheet 2, then the formulae for the missing lengths can be inserted manually in the empty cells, as shown in spreadsheet 3.

Excel spreadsheet 1: coding strings read from the CS output file, with header added

	A	B	C	D	E	F	G	H
	TEXT	TOKEN	(NON)FLAT	(NON)SPLIT	(NON)FINAL	LEN(C1)	LEN(C2)	LEN(C1+C2)
1	aelive	1254	flat	non.split	final	1		3
2	aelive	3418	flat	split	non.final	1	2	
3	aelive	537	non.flat	non.split	non.final	3	4	7
4	aelive	1143	non.flat	split	final	2	4	
5	...							

Excel spreadsheet 2: coding strings sorted by which length is missing

	A	B	C	D	E	F	G	H
	TEXT	TOKEN	(NON)FLAT	(NON)SPLIT	(NON)FINAL	LEN(C1)	LEN(C2)	LEN(C1+C2)
1	aelive	537	non.flat	non.split	non.final	3	4	7
2	aelive	1254	flat	non.split	final	1		3
3	aelive	3418	flat	split	non.final	1	2	
4	aelive	1143	non.flat	split	final	2	4	
5	...							

Excel spreadsheet 3: coding strings with formulae for missing length

	A	B	C	D	E	F	G	H
1	TEXT	TOKEN	(NON)FLAT	(NON)SPLIT	(NON)FINAL	LEN(C1)	LEN(C2)	LEN(C1+C2)
2	aelive	537	non.flat	non.split	non.final	3	4	7
3	aelive	1254	flat	non.split	final	1	=H3-F3	3
4	aelive	3418	flat	split	non.final	1	2	=F4+G4
5	aelive	1143	non.flat	split	final	2	4	=F5+G5
6	...							

Once the missing lengths have been calculated in the Excel spreadsheet 3 above, it is straightforward to calculate the average length of the 1st conjunct, 2nd conjunct and complete subject by sorting on column D and then adding the lengths of the split / non-split subjects and dividing by the total number of these tokens. T&P 2017 found that in the Old English data, the average total length of the coordination was 6.88 for split subjects and 5.99 for non-split subjects, a statistically significant difference. Similarly, the average length of the 2nd conjunct was 5.06 for split subjects and 4.08 for non-split subjects; again, a statistically significant difference. But for the 1st conjunct, the average length of the split subjects was less than the average length of the non-split subjects: 1.82 vs. 1.92.

6 Conclusions

In this article I have described and illustrated five methods for adding information to corpora that have been parsed in the Penn treebank style. These methods may involve manual operations, or they may be executed by CS functions, or they may require a combination of manual and automated procedures. Some of the methods overlap: for example, method 2 (inserting CODE nodes) is functionally equivalent to method 3 (embedding information in coding strings). Which method is used depends almost entirely on the type of information to be added and the goals of the user.

Acknowledgements

This paper was presented at the international symposium entitled "Exploiting Parsed Corpora: Application in Research, Pedagogy, and Processing" held at the National Institute for Japanese Language and Linguistics (NINJAL) on December 9-10, 2017. I thank the organizers for inviting me to that symposium, the audience for questions and comments, two anonymous reviewers for helpful suggestions, and all of my co-authors for giving me the opportunity to participate in so many interesting research projects.

References

Crisma, P. 2015. The 'indefinite article' from cardinal to operator to expletive. In A. J. Gianollo, C. and D. Penka, eds., *Language Change at the Syntax-Semantics Interface*, 125–151. Berlin: De Gruyter Mouton.

Crisma, P. and S. Pintzuk. 2016. *An* from Old to Middle English. In S. Vikner, H. Jørgensen, and E. van Gelderen, eds., *Let Us Have Articles Betwixt Us: Papers in Historical and Comparative Linguistics in Honour of Johanna L. Wood*, 31–53. Aarhus: Department of English, School of Communication and Culture, Aarhus University.

Frascarelli, M. and R. Hinterhölzl. 2007. Types of topics in German and Italian. In K. Schwabe and S. Winkler, eds., *On Information Structure, Meaning and Form: Generalizations across Languages*, 87–116. Amsterdam: John Benjamins.

Frey, W. 2006a. Contrast and movement to the German prefield. In V. Molnár and S. Winkler, eds., *The Architecture of Focus (Studies in Generative Grammar 82)*, 235–264. Berlin, New York: Mouton de Gruyter.

Frey, W. 2006b. How to get an object-es into the German prefield. In P. Brandt and E. Fuss, eds., *Form, Structure, and Grammar - A Festschrift Presented to Günther Grewendorf on Occasion of His 60th Birthday*, 159–185. Berlin: Akademie Verlag.

Haeberli, E. and S. Pintzuk. 2017. V3 in true V2 contexts in Old English. Presented at the workshop *V3 and Resumptive Adverbials*. Universiteit Gent, 5 October 2017.

Haeberli, E., S. Pintzuk, and A. Taylor. 2017. Object pronoun fronting and the nature of V2 in early English. Ms. University of Geneva, University of York.

Light, C. 2012. *The Syntax and Pragmatics of Fronting in Germanic*. Ph.D. thesis, University of Pennsylvania.

Taylor, A. and S. Pintzuk. 2015. Verb order, object position and information status in Old English. In T. Biberauer and G. Walkden, eds., *Syntax over Time: Lexical, Morphological and Information-Structural Interactions*. Oxford: OUP.

Taylor, A. and S. Pintzuk. 2017. Split coordination in Early English. In

B. Los and P. de Haan, eds., *Word order change in acquisition and language contact: Essays in honour of Ans van Kemenade*, 155–183. Amsterdam: John Benjamins.

Walkden, G. 2017. Language contact and V3 in Germanic varieties new and old. *JCGL* 20:49–81.

Appendix

Partial batch file for Old English data for Haeberli et al. 2017

Within a UNIX platform, batch files are run by typing 'source [filename]'. Blank lines and lines that start with the hash character (#) are ignored; in this way comments can be added to describe the procedures and searches. In the file below, the executable lines are shown in red.

```
# 0. remove old output files
rm *.out *.cmp *.cod

# 1. get all clauses with cp-adv xp sbj v and mark with 'z'
#    The cpadv must idom an IP-SUB with a finite verb.
corpussearch cpadv-sbj-v.q $york/*.psd
# output: 4457 hits

# 2. flag embedded IP-SUB under relevant CP-ADV with x- at start of label
corpussearch flag.q cpadv-sbj-v.out
# output: 4466 hits

# 3. remove (i.e. replace with RMV) all other embedded IP-SUBs
corpussearch remove-embedded-ips.q flag.out
# output: 4475 hits

# 4. remove x- attached to IP-SUBs
corpussearch removex.q remove-embedded-ips.out
# output: 4475 hits

# the result is a clean output file with all irrelevant embedded IP-SUBs
# removed

# 5. remove clauses with more than one constituent between cp-adv and sbj v
corpussearch cpadv1-sbj-v.q removex.out
# output files: cpadv1-sbj-v.out (4346 hits)
#               cpadv1-sbj-v.cmp (129 hits)

# 6. create dummy coding string for perl script to use
corpussearch code1.c cpadv1-sbj-v.out

# 7. code text and token ID using perl script
codeText0E-sp.prl code1.cod > code2-3.cod
```

```
# 8. code P of CP-ADV, main|conjunct, order of verb and subject
corpussearch code4-6.c code2-3.cod
```

Appendix B

Examples of NP types from Crisma and Pintzuk 2016

(II.1) a. existential interpretation (CODE: <NPTYPE:BSG-EXS>)
Eue heold iparais **long tale** wið þe
Eve held in-paradise (a) long conversation with the
neddre
serpent

(CMANCRIW-1,II.54.520)

b. existential interpretation (CODE: <NPTYPE:AN-EXS>)
As ha þeos bone hefde ibeden com **a kempe**
When she this plea had made came a champion
of helle on englene heowe
from hell in angel's guise

'When she had made this plea, a champion came from hell in the guise of an angel'
(CMJULIA,107.187)

(II.2) a. generic interpretation (CODE: <NPTYPE:BSG-GNR>)
þu seist þt muche confort haueð **wif** of hire
you say that much comfort has (a) wife from her
were
husband

'you say that a wife has much comfort from her husband ...'
(CMHALI,147.282)

b. generic interpretation (CODE: <NPTYPE:AN-GNR>)
Certes **a shadwe** hath the liknesse of the thyng of
Certainly a shadow has the likeness of the thing of
which it is shadwe
which it is (the) shadow

(CMCTPARS,292.C2.187)

(II.3) ambiguous between existential and generic
(CODE: <NPTYPE:AN-AMB>)
And therfore seith a **philosophre** in this wise
And therefore says a philosopher in this manner

'And therefore a (particular) philosopher says / philosophers say in this manner ...'
(CMCTMELI,224.C1.277)

(II.4) existential specific nominal
(CODE: <NPTYPE:AN-EXS-SPC>)
A yong man called Melibeus myghty and riche bigat
A young man called Melibee mighty and rich begat
upon his wyf that called was Prudence a **doghter** which
upon his wife who called was Prudence a daughter who
that called was Sophie
that called was Sophie

'A young man called Melibee, mighty and rich, begat upon his wife, who was called Prudence, a daughter who was called Sophie.'
(CMCTMELI,217.C1b.5)

(II.5) existential nominal with narrow scope
(CODE: <NPTYPE:AN-EXS-SCOPE-nrw>)
Ich chulle lete makie þe of golt **an ymage** as cwen
I shall let make thee of gold an image as queen
icrunet
crowned

'I will have a golden image made of you as a crowned queen'
(CMKATHE,36.269)

(II.6) existential nominal with wide scope
(CODE: <NPTYPE:AN-SCOPE-wd>)
& seide to hire þus. haue cwen **acrune** isent te
and said to her thus. have queen a-crown sent to-you
of heouene
from heaven

'and (he) said the following to her: Queen, have a crown, sent to you from heaven'
(CMKATHE,38.308)

(II.7) existential nominal with ambiguous scope
(CODE: <NPTYPE:AN-SCOPE-amb>)
thanne seketh he **an ydel solas** of worldly
then seeks he a useless consolation from worldly
thynges
things

'then he seeks a useless consolation from worldly things'
(CMCTPARS,313.C1.1073)

(II.8) nominals that are ambiguous, either generic or narrow scope existential

a. (CODE <NPTYPE:BSG-NPE>)
tis put he hat þt beo ilided þt **beast**
this pit he commanded that be covered that beast
þrin ne falle
therein NEG fall

'he commanded that this pit be covered lest (a) beast fall therein'
(CMANCRIW-1,II.48.446)

b. (CODE <NPTYPE:AN-NPE>)
whan **a gret lord** haþ no child he may chese a pore
when a great lord has no child he may choose a poor
mannes sone ȝif he wole and make of hym his eir bi
man's son if he will and make of him his heir by
adopcioun
adoption

'when a great lord has no child, he may choose a poor man's son if he wants and make him his heir by adoption'
(CMVICES4,100.63)