Corpus databases with feature pre-calculation

Erwin R. Komen

Radboud University Nijmegen // SIL-International
E-mail: E.Komen@Let.ru.nl

Abstract

Reliably coded treebanks are a goldmine for linguistics research. Answering a typical research question involves: (a) querying a treebank to extract sentences containing the feature to be investigated, (b) recognizing and keeping track of characteristics that determine the way in which the linguistic feature is encoded, and (c) using statistics to find out which (combination) of these characteristics determines the outcome of the linguistic feature. While sufficient tools are available for steps (a) and (c) in this process, step (b) has not received much attention yet. This paper describes how the programs "Cesax" and "CorpusStudio" can be used jointly to construct a "corpus research database", a database that contains the sentences of interest selected in step (a), as well as user-definable pre-calculated characteristics for step (b).

1 Introduction

Research into variation and change of syntactic constructions often consists of (1) automatically finding examples of the construction in a reliably coded treebank, (2) adding characteristics (features) to each of the examples, (3) gathering the results into a database, (4) manually editing the examples in the database, and (5) preparing the list of examples and their features for further statistical work with programs like "R" or "SPSS". The programs "Cesax" and "CorpusStudio" provide a windows-oriented relatively user-friendly way of achieving these goals [7].[1]

CorpusStudio facilitates queries written in the Xquery language [1], taking *xml* encoded treebank texts as input.[2] The program allows each "hit" to be accompanied by a user-definable number of features, and these features can be programmatically calculated, predicted, or given a default value. The results of a query project (which may involve multiple cascaded queries), together with the calculated features, can be saved as an *xml* database. The Cesax program is equipped with a feature to load such databases and contains an editor to work with the examples and their features. Cesax automatically adds a "Notes" field and a "Status" field to each database entry, allowing the user to annotate the database and to keep track of progress made. The database entries come with a predefined preceding and following context, as well as with the treebank syntax. Double-clicking an entry results in jumping

[1] CorpusStudio and Cesax are freely available from http://erwinkomen.ruhosting.nl.

[2] The *xml* format CorpusStudio deals with best is a TEI-P5 derivative using embedded hierarchy [11]. Labelled bracketing treebank files can be imported and transformed into this format using Cesax. CorpusStudio also allows working directly with the Negra and the Alpino formats, but the database features are not (yet) available for them. Future plans include conversion options for these formats.

to the actual location in the corpus file, which helps quickly looking for the larger context when this is needed (it is this simple feature that is perhaps most valued by the users). Cesax also allows exporting the database for use in statistics.

This paper provides a walk through the process described above, and it does so by taking the “progressive inversion” as an example.

2 The progressive inversion

The progressive inversion construction is a subtype of VP inversion [12]. It is similar to the locative inversion, except that the first constituent is a participle clause instead of a prepositional phrase, as for example (1a):

(1) a. [$_{\text{IP-PPL}}$ Trending away on either side of the port] was [$_{\text{NP-SBJ}}$ a bold rocky coast, varied here and there with shingly and sandy beaches]. [fayrer-1900:54]

b. ?[$_{\text{Sbj}}$ A bold rocky coast] was trending away on either side of the port.

The uninverted variant of (1a) would be (1b), but the question mark indicates that this is not quite okay for native speakers. The linguistic question I would like to posit for the sake of this walk-through is: “Which features could determine the appearance of a progressive inversion?”

3 Automatically finding examples

Having defined the research question, step (1) in the process of answering it (see Introduction) is to define a query that automatically locates the necessary examples of the linguistic feature that is being targeted. Sentences that contain a progressive inversion need to have the following three elements:

1) Subject
2) Finite verb
3) Participle

Once sentences containing these three elements are located, the order of these elements will show whether an inversion construction is being used (participle-finite verb-subject) or some other construction (such as: subject-finite verb-participle). The task of locating sentences and determining whether they contain a progressive inversion or not can be accomplished in CorpusStudio by using the Xquery code in (2).[3]

What the code does is: select main clauses into variable `$search` (line 2), put the subject of the main clause into `$sbj` (line 5), put the finite verb of the main clause into `$vfin` (line 8), any participle of the main clause is put into `$ptcp` (line 11), determine the word order (line 14,15), return this clause if all the elements are there (line 18-23). The result of running the Xquery code (2) consists of all the sentences containing the required elements for the

[3] The code makes use of standard Xquery functionality (`for-let-where-return`, `if-then-else`, the function “`exists()`”), some built-in Xquery functions (“`ru:matches`”, “`ru:relates`”, “`ru:back`”), user-defined functions that are elsewhere in the code (“`tb:SomeChildNo`”, “`tb:SomeChild`”), and user-defined global variables (“`$_matrixIP`”, “`$_subject`”).

progressive inversion (subject, finite verb, participle), and these sentences are divided over the word orders ‘Ptcp-Vfin-S’ and ‘Other’.

(2) *Xquery code to find the inversion examples*

```
 (: Look in all main clauses :)
 for $search in //eTree[ru:matches(@Label, $_matrixIP)]

 (: There must be a subject and a finite verb :)
 let $sbj := tb:SomeChildNo($search, $_subject, $_nosubject)
 let $vfin := tb:SomeChild($search, $_finiteverb)

 (: There must be a progressive or ptcp, but not an absolute :)
 let $ptcp := tb:SomeChildNo($search,
                               'IP-PPL*|[VB]AG*|PTP*', '*ABS*')

 (: Find out word order :)
 let $order := if ( ($vfin << $sbj) and ($ptcp << $vfin))
                then 'Ptcp-Vfin-S' else 'Other'

 (: Check conditions: subject, V-fin, progressive, word order :)
 where (  exists($sbj)
          and exists($vfin)
          and exists($ptcp)  )

(: Return the main clause, subcategorize on word order :)
return ru:back($search, '', $order)
```

While the Xquery code in (2) serves its purpose well, a few extensions are required that will show up later in the code. Two particular main clause types need to be excluded, since they skew the data: the quotations (QTP clauses) and main clauses with left dislocations (those with an LFD element); the algorithm should only look for non-empty subjects.

4 Adding features to the results

Step (2) in the process of addressing the linguistic question at hand (see Introduction) is to add characteristics, or ‘features’, to each of the examples we find. One way to do this in Xquery is to make a user-defined function. This function, which will receive the name `tb:ProgrInv()`, is called in line 23 of the extended version of the main query (3). The main query is also extended with a test for the exclusion of left-dislocated and quotative main clause type in lines 5-6 and 26, while lines 10 and 27 make sure that empty subjects (such as traces and dislocation markers) are excluded from consideration.

(3) *Add features to the progressive inversion*

```
(: Look in all main clauses :)
for $search in //eTree[ru:matches(@Label, $_matrixIP)]

(: Some clauses need to be excluded :)
let $clsOk := not(exists($search/child::eTree
                       [ru:matches(@Label, 'QTP*|*LFD*')]))

(: There must be a subject :)
let $sbj := tb:SomeChildNo($search, $_subject, $_nosubject)
let $sbjOk := not(exists($sbj[child::eLeaf/@Type = 'Star']))

(: There must be a finite verb :)
let $vfin := tb:SomeChild($search, $_finiteverb)

(: There must be a progressive or ptcp, but not an absolute :)
let $ptcp := tb:SomeChildNo($search,
                              'IP-PPL*|VAG*|BAG*|PTP*', '*ABS*')

(: Prepare subcategorization: ptcp type  :)
let $cat := ru:cat($ptcp, 'phrase')

(: Combine features into a CSV for database creation  :)
let $db := tb:ProgrInv($sbj, $vfin, $ptcp)

(: Check conditions: sbj, Vfin, progressive and word order :)
where (  $clsOk
         and exists($sbj) and $sbjOk
         and exists($vfin)
         and exists($ptcp)
      )

(: Return clauses found, subcategorize on the word order :)
return ru:back($search, $db, $cat)
```

The function `tb:ProgrInv()` is defined in such a way, that it returns a string array of the features. These features are subsequently passed on to the CorpusStudio engine through the `$db` variable as an argument of the built-in `ru:back()` function, where they will be available for the next step in the process.

Turning now to the feature calculation, there are two kinds of features the database should be equipped with: those that are going to be used for statistics (such as the kind of verb used, the size of the subject), and those that are important for visual inspection by the database user (such as the text of the subject, finite verb and participle). The code for the `tb:ProgrInv()` function where the features are calculated is provided in (4).

(4) *Xquery code that calculates the feature values for one example*

```
1.  (: ----------------------------------------------------------
2.     Name : tb:ProgrInv
3.     Goal : Provide features for the progressive inversion database
4.     History:
5.     13-06-2013     ERK    Created
6.     ---------------------------------------------------------- :)
7.  declare function tb:ProgrInv(
8.     $sbj as node()?, $vfin as node()?, $ptcp as node()?)as xs:string
9.  {
10.  (: ==========================================================
11.      Feature calculation starts here
12.      ========================================================== :)
13.   (: Feature #1-3: the text of the ptcp, V-finite and subject :)
14.   let $ptcpText := replace(tb:Sentence($ptcp), ';', ' ')
15.   let $vfinText := replace(tb:Sentence($vfin), ';', ' ')
16.   let $sbjText := replace(tb:Sentence($sbj), ';', ' ')
17.
18.   (: Feature #4: word order -- Ptcp-Vfin-S, or other? :)
19.   let $order := if ( ($vfin << $sbj) and ($ptcp << $vfin))
20.                 then 'Ptcp-Vfin-S' else 'Other'
21.
22.   (: Feature #6: the type of participle :)
23.   let $ptcpType := if (ru:matches($ptcp/@Label,
24.              'IP-PPL*|VAG*|BAG*|PTP*')) then 'Present' else 'Past'
25.
26.   (: Feature #7: the number of constituents after V-finite :)
27.   let $postVf := count($vfin/following-sibling::eTree[
28.                     not(ru:matches(@Label, $_ignore_nodes_conj))])
29.
30.   (: Feature #8: the number of words in the subject :)
31.   let $sbjSize := count($sbj/descendant::eLeaf[@Type = 'Vern'])
32.
33.   (: Feature #9: NPtype of the subject :)
34.   let $sbjType := ru:feature($sbj, 'NPtype')
35.
36.   (: Feature #10: estimate of referentiality of the subject :)
37.   let $sbjRef := ru:RefState($sbj)
38.
39.   (: ==========================================================
40.      Combine features into a CSV for database creation
41.      ========================================================== :)
42.   return concat($ptcpText, ';',
43.      $vfinText, ';',      $sbjText, ';',
44.      $order, ';',         $ptcp/@Label, ';',
45.      $ptcpType, ';',      $postVf, ';',
46.      $sbjSize, ';',       $sbjType, ';',
47.      $sbjRef )
48. } ;
```

As far as the features necessary for visual inspection, the function `tb:ProgrInv()` calculates the text of the participle (line 15), the text of the finite verb (line 18) and the text of the subject (line 21).

Statistically important is the dependent variable `$order` as calculated in lines 24-25: this feature either has the value “Ptcp-Vfin-S”, in which case the example is a progressive inversion, or it has the value “Other”, in which case the example is *not* an inversion. The features numbered 5-10 in the Xquery code (4) are independent variables that could all possibly influence the word order, and they are summarized in Table 1.

#	Feature	Explanation
5	PtcpLabel	syntactic label of the participle (VAG, IP-PPL etc)
6	PtcpTense	progressive is ‘Present’ or ‘Past’ tense[4]
7	PostVfNum	number of sibling-constituents following V_{finite}
8	SbjSize	number of words in the subject
9	SbjType	NPtype of the subject
10	SbjRef	Estimate for subject’s referential status

Table 1 Features that represent independent variables in a statistic analysis

The features numbered 5-8 are ‘fixed’ in the sense that they are calculated automatically and do not need manual correction. This is not the case for features 9 (SbjType) and 10 (SbjRef). These features are estimated automatically, but they may need manual correction.

The “SbjType” feature, for instance, makes use of the “NPtype” feature that has been added to the original Treebank texts. But this feature has not been determined for some of the Noun Phrases, which are distinguishable by having the feature value “unknown”.

The “SbjRef” feature makes use of the built-in CorpusStudio function “`ru:RefState`”, which has a success rate of approximately 85% in determining the referentiality of an NP. The values of this feature all need to be checked manually![5]

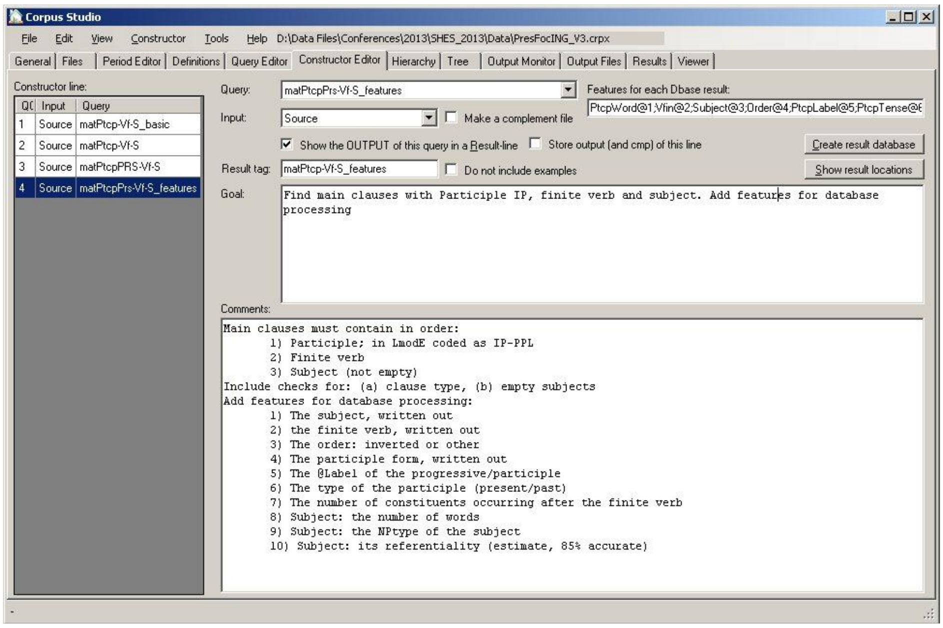
5 Making a database

Next in the process of a full-fledged linguistic analysis as mentioned in the introduction is step (3), making a database. It is to this end that the second argument of the “`ru:back`” function has been filled with a semicolon-separated list of feature values. When the queries have been run on the input texts within the CorpusStudio program, an *xml* file that contains all the important information on the result sentences is created, but this is not yet the database. This correct part of this result file can be transformed into an *xml* database by pressing a button within CorpusStudio, labelled “**create result database**”.

Figure 1 provides a screenshot of the relevant part of CorpusStudio, called the “ConstructorEditor”. This editor contains the queries that are to be processed for the currently loaded corpus research project, and it defines their hierarchical order. The CorpusStudio manual describes the process involved in generating a database from the results of a query line in more detail [5].

[4] This feature is unnecessary for the current example, where we only look at present-tense progressive inversion.

[5] The Xquery functions starting with the “`ru:`” prefix are all listed in the CorpusStudio manual. These functions have been hard-coded in CorpusStudio and approach the *xml* documents through the Microsoft *xml* library; the program makes use of the Saxon Xquery dll, which, in turn, allows host-programs to provide additional Xquery functions through a namespace declaration that points to the executable itself. The `ru:RefState` function is described more fully in [8].


Corpus Studio
File Edit View Constructor Tools Help
General | Files | Period Editor | Definitions | Query Editor | Constructor Editor | Hierarchy | Tree | Output Monitor | Output Files | Results | Viewer
Create result database
Show result locations

D:\Data Files\Conferences\2013\SHES_2013\Data\PresFocING_V3.crpx

Constructor line:

QC	Input	Query
1	Source	matPtcp-Vf-S_basic
2	Source	matPtcp-Vf-S
3	Source	matPtcpPRS-Vf-S
4	Source	matPtcpPrs-Vf-S_features

Query: matPtcpPrs-Vf-S_features

Input: Source

□ Make a complement file

Features for each Dbase result: PtcpWord@1;Vfin@2;Subject@3;Order@4;PtcpLabel@5;PtcpTense@6

☑ Show the OUTPUT of this query in a Result-line

□ Store output (and cmp) of this line

Result tag: matPtcp-Vf-S_features

□ Do not include examples

Goal:

```
Find main clauses with Participle IP, finite verb and subject. Add features for database
processing
```

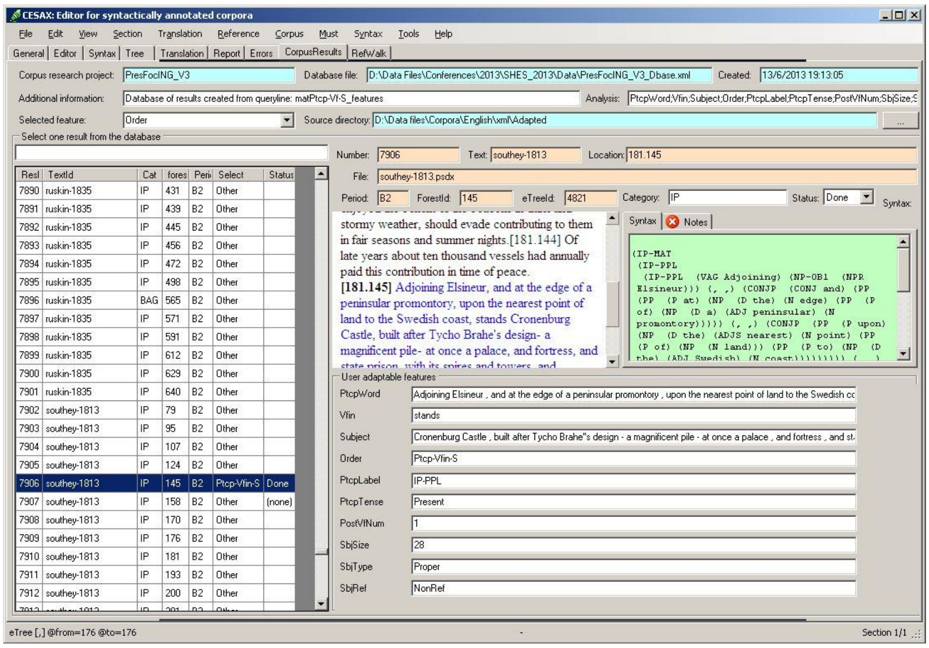
Comments:

```
Main clauses must contain in order:
        1) Participle; in LmodE coded as IP-PPL
        2) Finite verb
        3) Subject (not empty)
Include checks for: (a) clause type, (b) empty subjects
Add features for database processing:
        1) The subject, written out
        2) the finite verb, written out
        3) The order: inverted or other
        4) The participle form, written out
        5) The @Label of the progressive/participle
        6) The type of the participle (present/past)
        7) The number of constituents occurring after the finite verb
        8) Subject: the number of words
        9) Subject: the NPtype of the subject
        10) Subject: its referentiality (estimate, 85% accurate)
```

Figure 1 Creating an *xml* database in the Constructor Editor of CorpusStudio

6 Editing examples in the database

Steps (1) to (3) involved in working through a linguistics example, as described in the introduction, have been taken, and everything is ready for step (4), manually editing and inspecting the database. Loading the database in Cesax results in the following display.


CESAX: Editor for syntactically annotated corpora
File Edit View Section Translation Reference Corpus Must Syntax Tools Help
General | Editor | Syntax | Tree | Translation | Report | Errors | CorpusResults | RefWalk
Syntax | Notes
Syntax:
...

Corpus research project: PresFocING_V3

Database file: D:\Data Files\Conferences\2013\SHES_2013\Data\PresFocING_V3_Dbase.xml

Created: 13/6/2013 19:13:05

Additional information: Database of results created from queryline: matPtcp-Vf-S_features

Analysis: PtcpWord;Vfin;Subject;Order;PtcpLabel;PtcpTense;PostVfNum;SbjSize;S

Selected feature: Order

Source directory: D:\Data files\Corpora\English\xml\Adapted

Select one result from the database

Resl	TextId	Cat	fores	Peri	Select	Status
7890	ruskin-1835	IP	431	B2	Other	
7891	ruskin-1835	IP	439	B2	Other	
7892	ruskin-1835	IP	445	B2	Other	
7893	ruskin-1835	IP	456	B2	Other	
7894	ruskin-1835	IP	472	B2	Other	
7895	ruskin-1835	IP	498	B2	Other	
7896	ruskin-1835	BAG	565	B2	Other	
7897	ruskin-1835	IP	571	B2	Other	
7898	ruskin-1835	IP	591	B2	Other	
7899	ruskin-1835	IP	612	B2	Other	
7900	ruskin-1835	IP	629	B2	Other	
7901	ruskin-1835	IP	640	B2	Other	
7902	southey-1813	IP	79	B2	Other	
7903	southey-1813	IP	95	B2	Other	
7904	southey-1813	IP	107	B2	Other	
7905	southey-1813	IP	124	B2	Other	
7906	southey-1813	IP	145	B2	Ptcp-Vfin-S	Done
7907	southey-1813	IP	158	B2	Other	(none)
7908	southey-1813	IP	170	B2	Other	
7909	southey-1813	IP	176	B2	Other	
7910	southey-1813	IP	181	B2	Other	
7911	southey-1813	IP	193	B2	Other	
7912	southey-1813	IP	200	B2	Other	

Number: 7906 Text: southey-1813 Location: 181.145

File: southey-1813.psdx

Period: B2 ForestId: 145 eTreeId: 4821 Category: IP Status: Done

stormy weather, should evade contributing to them in fair seasons and summer nights.[181.144] Of late years about ten thousand vessels had annually paid this contribution in time of peace.
[181.145] Adjoining Elsineur, and at the edge of a peninsular promontory, upon the nearest point of land to the Swedish coast, stands Cronenburg Castle, built after Tycho Brahe's design- a magnificent pile- at once a palace, and fortress, and state prison, with its spires and towers, and

```
(IP-MAT
 (IP-PPL
  (IP-PPL  (VAG Adjoining) (NP-OB1  (NPR
Elsineur))) (, ,) (CONJP  (CONJ and) (PP
(PP  (P at) (NP  (D the) (N edge) (PP  (P
of) (NP  (D a) (ADJ peninsular) (N
promontory))))) (, ,) (CONJP  (PP  (P upon)
(NP  (D the) (ADJS nearest) (N point) (PP
(P of) (NP  (N land))) (PP  (P to) (NP  (D
the) (ADJ Swedish) (N coast)))))))) (, ,)
```

User adaptable features

PtcpWord	Adjoining Elsineur , and at the edge of a peninsular promontory , upon the nearest point of land to the Swedish cc
Vfin	stands
Subject	Cronenburg Castle , built after Tycho Brahe''s design - a magnificent pile - at once a palace , and fortress , and st.
Order	Ptcp-Vfin-S
PtcpLabel	IP-PPL
PtcpTense	Present
PostVfNum	1
SbjSize	28
SbjType	Proper
SbjRef	NonRef

eTree [,] @from=176 @to=176

Section 1/1

Figure 2 Loading an *xml* database in Cesax

The Cesax program has originally been created to facilitate coreference resolution and referential state processing, but it has been extended with several more functions, one of which is the editing of databases. Once a database has been loaded, editing options become available on the “CorpusResults” tab page [6].

1) **Delete**. Individual records can be deleted, but it is also possible to keep the records that are available, and indicate their status as “Ignore”.
2) **Add**. If important sentences have not been captured by the database construction query, it is better to adapt the query in such a way that all sentences are added.
3) **Editing**. Feature values can be edited in the textboxes available for each record.
4) **Notes**. The “Notes” window allows adding remarks to individual records
5) **Status**. The status of each record can be set in order to keep track of progress.
6) **Bulk-changes**. Two different methods are provided to provide a search and replace feature. The most extensive option uses Xpath to find its way through the results in the database *xml* file, but it uses a user-friendly interface.

The database results can be re-ordered on the basis of any of the columns, and one column can be filled with one of the user-supplied features. It is also possible to *filter* the database without actually changing its content. These kinds of features make life easier for the annotator, especially when databases are large (the databases with results I have encountered typically exceed 10.000 sentences).

The syntax and local context of each record in the database are immediately visible in the “CorpusResults” tab page, but it may, at times, be necessary to look at the sentence that has been found in the larger context of the original text. Cesax allows this: double clicking the entry in the results list opens the corpus file on the corresponding place and shows it in the “Editor” tab page. Should it be necessary to take a different look at the syntax of this particular example, then clicking the “Tree” tab page results in displaying the selected sentence in a syntactic tree.

7 Preparation for statistics

Step (5) in the process described in the introduction involved preparing the database results for statistical processing. Cesax contains several commands to suit the needs of the user. Preparation for SPSS processing, for instance, involves the following steps:

1) Construct a table with the ‘original’ values of the features; the values as they are visible in the CorpusResults tab page;

2) Construct a tab-separated text file where the ‘original’ feature values are replaced by numerical values (an additional table with the ‘index’ to these values is supplied separately);
3) Construct a separate .sps file (an SPSS ‘syntax’ file).

Work with SPSS can be conducted by transferring the second (numerical value) table to SPSS, and processing the .sps file with the feature values. An SPSS user will, in addition to this, also need to specify which features are to be excluded from statistic, which are the independent variables, and which one is the dependent variable.

Work with memory-based language programs like ‘TiMBL’ is also supported [4]. Cesax allows preparing a training and test file with the necessary features for further processing by TiMBL.

Since the purpose of this paper is to show *how* data gained through corpus searches can be prepared for statistical processing, no attempt will be made to figure out which of the independent variables play a role in determining whether progressive inversion occurs or not.

8 Querying a database

Once a database has been manually edited, as described in section 6, a user will probably not want to go back to adjusting the original corpus query (section 3) in order to make a new version of a database (e.g. one that contains a selected subset, or one with adjusted feature values). This may, due to the cyclic process of research in general, not always be circumvented, but the CorpusStudio-Cesax combination does allow for one way out. If a user wants to make an adapted database that (a) uses a subset of the features available in the original one, or (b) that has records filtered out by additional criteria, or (c) that uses additional features that can be calculated on the basis of the existing ones, then this can be achieved by writing a query with the database as input. The CorpusStudio manual contains information on how to do this.

Returning now to the linguistic task that has been undertaken as an example, I would not like to withhold the outcome to the interested reader. The manually inspected corpus database yields a total of twelve examples of the progressive inversion (against a total of 5-6 million words), and the first clear one is found in early Modern English (1500-1700).

(5) a. and vpon the ryght hande goynge from Rama to Jherusalem, about .xx. myle from Rama, <u>is</u> **the castell of Emaus**. [chaplain-e1-p2:289]

The example in (5) has the finite verb *is* preceded by a participle clause that is headed by the present participle *going*. It clearly serves to introduce a new ‘participant’ in the narrative, namely the castle of ‘Emaus’.

9 Discussion

This paper has shown a new, windows-based approach to research into variation and change of syntactic constructions. The new approach is centered around the programs CorpusStudio and Cesax, and makes heavy use of *xml*, *xpath* and *xquery*, which have become standard public-domain conventions.

Just as CorpusSearch [9], tgrep [10], TigerSearch [3] and similar query programs do, CorpusStudio allows for the definition of queries that select sentences from syntactically parsed texts on the basis of user-definable criteria. Just as the Alpino project [2, 13] does, CorpusStudio makes use of the Xquery language with all its advantages in terms of user-extensibility, recursive functions and independent W3C development. Different from its competitors, however, CorpusStudio allows for combining multiple queries into a *corpus research project* that is kept in one place, which facilitates experiment replicability. Essential for the creation of a database with examples is CorpusStudio's capability to provide the examples that are found with pre-calculated feature values. This capability surpasses, for instance, CorpusSearch's "coding" functionality; first in the area of user-friendliness, and second in terms of complexity. Pre-calculating feature values in CorpusStudio is "advanced", since it can make use of the Xquery functionality of user-definable functions, and it can make use of the Xquery functions that have been hard-wired into CorpusStudio.

Since databases that have been made with CorpusStudio contain features that can have text values, editing such databases becomes a doable task. When database entries are also supplied with notes, the data become a valuable treasure, that allow back-tracking annotation choices. The facility to jump to the location in the text associated with a database entry allows for speedy inspection of the larger context, and it opens the way to a tree-view of the selected sentence's syntax.

Cesax allows simple transformation of a database into a format that can be used by statistical programs such as "R" and "SPSS", as well as by memory-based learning programs such as "TiMBL".

I suggest that future developments of Corpus databases based on treebanks involve web interfaces instead of dedicated programs (which tend to be OS-dependant), but I leave that challenge to the experts.

10 References

[1] Boag, Scott, Chamberlin, Don, Fernández, Mary F., Florescu, Daniela, Robie, Jonathan, and Siméon, Jérôme (2010) XQuery 1.0: An XML Query Language (Second Edition) W3C Recommendation.

[2] Bouma, Gosse (2008) XML information extraction with Xquery: processing wikipedia and Alpino trees. In Editor (ed.)^(eds.): 'Book XML information extraction with Xquery: processing wikipedia and Alpino trees' (Information science, university of Groningen, edn.), pp.

[3] Brants, Sabine, Dipper, Stefanie, Eisenberg, Peter, Hansen-Schirra, Silvia, König, Esther, Lezius, Wolfgang, Rohrer, Christian, Smith, George, and Uszkoreit, Hans (2004) TIGER: Linguistic Interpretation of a German Corpus. *Research on Language and Computation* 2, (4), 597-620.

[4] Daelemans, Walter, and Bosch, Antal van den (2005) Memory-based language processing (Cambridge University Press, 2005)

[5] Komen, Erwin R. (2009) Corpus Studio manual. Nijmegen: Radboud University Nijmegen.

[6] Komen, Erwin R. (2011) Cesax: coreference editor for syntactically annotated XML corpora. Reference manual. Nijmegen, Netherlands: Radboud University Nijmegen.

[7] Komen, Erwin R. (2012) Coreferenced corpora for information structure research. In Tyrkkö, Jukka, Kilpiö, Matti, Nevalainen, Terttu, and Rissanen, Matti (eds.) *Outposts of Historical Corpus Linguistics: From the Helsinki Corpus to a Proliferation of Resources. (Studies in Variation, Contacts and Change in English 10)*. Helsinki, Finland: Research Unit for Variation, Contacts, and Change in English.

[8] Komen, Erwin R. (2013) Predicting referential states using enriched texts. In Editor (ed.)^(eds.): 'Book Predicting referential states using enriched texts' (edn.), pp.

[9] Randall, Beth, Taylor, Ann, and Kroch, Anthony (2005) http://corpussearch.sourceforge.net, accessed 2/Jun/2009

[10] Rohde, Douglas L. T. (2005) TGrep2 user manual

[11] Sperberg-McQueen, C.M., and Burnard, Lou (2009) TEI P5: Guidelines for Electronic Text Encoding and Interchange (TEI Consortium, 2009)

[12] Ward, Gregory L., and Birner, Betty J. (1992) VP inversion and aspect in written texts. In Stein, Dieter (ed.) *Co-operating with written texts : the pragmatics and comprehension of written texts*, pp. 575-588. Berlin; New York: Mouton de Gruyter.

[13] Yao, Xuchen, and Bouma, Gosse (2010) Mining Discourse Treebanks with XQuery. In Editor (ed.)^(eds.): 'Book Mining Discourse Treebanks with XQuery' (edn.), pp. 245-256