Longdale workshop januari 2014

1 Queries that have been suggested:

#	Query	Find	Count	Categories	Database
1	Find passives	x	x		
2	Find *it*-extraposition	x	x		
3	Find relative clauses ←	x	x		
4	Find embedded *–ed* clauses	x	x		
5	Find embedded *–ing* clauses	x	x		
6	Find periphrastic tenses	x	x	tense type	
7	Count tense forms	x	x	tense type	
8	Count aspect use	x	x	aspect type	
9	Where are adverbials used?	x	x	Adv[P] location	
10	NP: relative clause types	x	x	RC type	x
11	NP: modification types	x	x	simple, pre, post	x
12	Structure/complexity of NPs	x	x	NP type	x
13	Structure/complexity of VPs	x	x	VP type	x

2 General guidelines

The general procedure to any of the problems above:

1) Find
   a) Identify an example of the form or construction you are looking for
   b) Note the essential constituents
   c) Note the hierarchical relationships between these constituents
   d) If possible: prepare a sample in Cesax for the Query Wizard in CorpusStudio
   e) Start a new project in CorpusStudio
   f) Create your query by making use of the Query Wizard
      i) Make sure to set the output type to "Database" if that is what you intend
   g) Set the "Files" correctly
      i) Make sure to set the input file directory to a small section (testset) of your data
   h) Put the query in the execution pipeline (Constructor Editor)
   i) Execute the query
   j) Look at the results
      i) What is superfluous?
      ii) What is missing?
   k) Adapt your query and repeat steps (i)-(j)
2) Categories
   a) What are the subcategories you want your output to be in?
      i) Adapt the calculation of the $cat variable accordingly
   b) Repeat steps (1i)-(1j)
3) Database
   a) Adapt/extend the features to be passed on to the database
      i) Use the feature-preparation function that has been created in the "Definitions"
   b) Adapt the feature pass-on list in "Constructor Editor"
   c) Create your database
   d) Import it into Cesax
   e) Check it, and if necessary adapt and repeat steps (3a-d)

3 Suggestions for some queries

3.1 Passive voice

What are the main features of the passive?

1) Look for verb forms in the correct VPs
   a) Only look at VPs whose parent is "S"
2) The VP needs to contain a from of "be"
   a) This is a descendant of the VP
   b) It needs to come *before* any "S" or "SBAR" item
      i) Its first "S" ancestor must be equal to the "S" in (1a)
3) The VP should *end* with a past participle
   a) This is the last descendant before any "S" or "SBAR" item → see (2bi)

3.2 Extraposition with *it*

Follow the general guidelines

3.3 Relative clauses

This is treated in the main sessions

3.4 Embedded *–ed* clauses and *–ing* clauses

Follow the general guidelines
Be sure to restrict the hierarchical relations to "child" and "parent".
Try to prevent using "descendant" relations, and if you do, build in tests to see you are on the right 'level' of clause complexity.

3.5 Count tense forms

Follow the general guidelines

3.6 Periphrastic tenses

Follow the general guidelines
The VP must have at least two verb forms *before another embedding starts*
(See notes under VP complexity)

3.7 Count aspect use

Follow the general guidelines

3.8 Where are adverbials used

Follow the general guidelines
Restrict yourself to adverbials (ADVP or PP) that are child of S.
Get the position of the adverbials with respect to clause-begin, clause-end, pre-VP or post-VP.

3.9 NP: relative clause type

Here are some relative clause types I found:

	NP consists of:	**Example**
Relative clause with *that*	NP + SBAR[WHNP S] WHNP = WDT_{that}	another thing **that** might persuade me to get a room
Relative clause with *which*	NP + SBAR[WHNP S] WHNP = WDT_{which}	a statement **which** finally brings me to my point
Relative clause with P + *which/who*	NP + SBAR[WHPP S]	the language **in which** I can express my feelings in a more beautiful way
Relative clause with *who*	NP + SBAR[WHNP S] WHNP = WP_{who}	people **who** interest me
Relative clause with *why*	NP + SBAR[WHADVP S] WHADVP = WRB_{why}	a reason **why** I'm good at English
Relative clause with *where*	NP + SBAR[WHADVP S] WHADVP = WRB_{where}	a website **where** student rooms are put up
Relative clause without C	NP + SBAR[S] S = NP + VP	the English accent I find very interesting

You can basically follow the main session lessons, and adapt the `$cat` variable calculation to your desires. Be sure to make this into a *separate corpus research project*!!!

3.10 NP: modification types

Post-modifiers can be relative clauses; see the previous section.
There can be other post-modifiers of NPs too, but I have only been able to find one, so far

	NP consists of:	**Example**
Comparative clause	NP + SBAR[IN_{than} + S]	a much prettier language **than** Dutch
Other possibilities??		

There may also be occasions where the parser has wrongfully assigned a postmodification to an NP, such as this:

```
(S
  (PP (IN For) (NP (NN instance)))
  (: ;)
  (NP (PRP I))
  (ADVP (RB just) (RB recently))
  (VP (VBD learned)
   (SBAR
    (IN that)
    (S
     (NP (DT the) (NNP Irish))
     (VP (VBP are)
         (NP
            (NP (NNS fishermen)) (, ,)
            (SBAR
              (WHNP (WDT which))
              (S (VP (VBZ is) (NP (NP (NN something))
                 (SBAR
                 (S (NP (PRP I)) (VP (VBD came)
                   (PP (IN across) (PP (IN in)
                   (NP (DT a) (NN pancake) (NN restaurant))))))))))))))
  (. .))
```

The NP "fisherman" does not really have a post-modifier. The *which* clause should have been attached as an adverbial clause to the whole sentence.

3.11 NP complexity

There are a number of different measures that can give an indication of the NP complexity.

3.11.1 Number of words

A rough indication of complexity is the average number of words per NP. The number of words for a constituent can be counted by using the built-in Xquery function `ru:words()`. In order to get the program to calculate averages, this number needs to be presented to a function that calculates the averages: `ru:avg()`. You need to take up the result of the `ru:avg()` in the end-evaluation, in the "where" condition (otherwise it will not get evaluated).

```
(: Look for NPs that are *not* under another NP :)
for $search in //eTree[ru:matches(@Label, 'NP|NP-*')]
let $par := $search/parent::eTree[@Label = 'NP']

(: Get the size of the NP :)
let $wrds := ru:words($search)

(: Make sure averages are calculated :)
let $avg := ru:avg($wrds, 'AnyNP')

(: make a message per hit :)
let $msg := concat('NP [', ru:NodeText($search), '] = ', $wrds)

(: Make sure this clause has a subject :)
where (  not(exists($par))
         and $avg
      )
(: Return the main clause :)
return ru:back($search, $msg, $wrds)
```

3.11.2 NP-hood testing

It is sometimes difficult to find out whether an NP really is an NP or not, but here is one test:

> Any NP must ultimately be part of a sentence (`S`), so it must have an ancestor `eTree` with `@Label` feature "`S`". Instances where the parser has not been able to identify a sentence have "`FRAG`" (fragment) as the top node.

3.11.3 Other NP complexity measures

You can test the "depth" in terms of number of embedded clauses of an NP by counting the number of constituents with label "S" (or "SBAR"?) descending from the NP you are looking at. Make use of the Xquery function `count()`.

The number of "phrases" inside an NP can also be counted. If you take a phrase to be any non-endnode, then you can use:

```
let $feat_PhrNum := count($np/descendant::eTree[not(exists(child::eLeaf))])
```

Note: instead of `$np` you will need to use your own variable that points to the NP node.

3.12 VP: complexity

You can use a lot of the meeasures listed for the NP complexity.
Just be aware that VPs in particular may be nested quite deep.

If you want to count, for instance, the number of phrases inside a VP, excluding the end-nodes, and restricting yourself to the 'current' VP, without any deeper nested clauses, this is what it would look like:

```
(: Suppose you have a clause called "$basicS" :)
let $vp := $basicS/child::eTree[@Label = 'VP']

(: Get the number of phrases :)
let $feat_PhrNum := count($vp/descendant::eTree[
                           (: Exclude end-nodes :)
                           not(exists(child::eLeaf))
                           (: First S-ancestor must be $basicS :)
                           and (ancestor::eTree[@Label='S'][1] is $basicS)
                         ])
```