PROIEL

Querying PROIEL with INESS

Dag Haug

10 February 2016

The PROIEL treebank

- PROIEL is a treebank covering ancient Indo-European languages

Classical Armenian	22773
Gothic	56315
Greek	233974
Latin	173958
Old Church Slavonic	71531

- The core is a parallel corpus of the ancient Indo-European languages
- We are currently planning pilot annotation of Vedic, Hittite and Lithuanian
- We have expanded the coverage of Greek and Latin beyond the NT

Greek and Latin texts

- Herodotus: book 1, last part of book 4, books 5-7
- Sphrantzes: Chronicon Minus (15th century Chronicle), complete
- Caesar, The Gallic War: books 1-4, 6
- Cicero, Letters to Atticus: most of book 1, 3, 4; sporadic coverage of 2 and 5
- Cicero, De Officiis: half of book 1
- Peregrinatio Aetheriae, complete

Annotations

- Morphology: very similar to AGDT
- Syntax: similar but different in details from AGDT
- Information structure: unique, but only available in the NT + Caesar
- Semantics (animacy): unique

Availability of data

`http://foni.uio.no:3000` Browsing, simple single-token querying of all annotations (click "search")

`http://iness.uib.no` Deep querying of morphology and syntax

`http://proiel.github.io/` Versioned source data available for download (under CC-BY-NC-SA) with all annotations and full flexibility

- Today's topic is the INESS web page
- This query language is based on TigerSearch, which is the basis for many other search engines

Basics of INESS

- INESS is a portal offering access to many treebanks of different languages
- It works best in Chrome
- To use it you need to know about two things
  - The INESS query language (based on TigerSearch)
  - The particulars of the corpus you are looking at, in our case PROIEL
- First, we need to choose the right corpus:
  - Click “Treebank selection”
  - Click e.g. “Latin”
  - You get a list of all available Latin texts
  - There are 34! But most contain little data beyong the text itself
  - By default, you query all but you can toggle this on the left side
- “Documentation” will take you to documentation of the query language

Words and links, nodes and edges

- The data are represented as words (aka **nodes**) and links (aka **edges**) between them
- Nodes have **attributes**

word	is the form that occurs in the text
lemma	is the dictionary entry
pos	is the part of speech
morph	is the morphological tag

- Edges come in two types and have **labels**
  - Primary edges are labelled with syntactic functions such as **sub**, **obj**
  - Secondary edges are labelled with ordinary syntactic functions or special relations that we will ignore here

Structure of an attribute query

- Constraints on a single node go inside square brackets and are of the form `atr="val"`, e.g. `[word="sum"]`
- Before the equals sign is an **attribute**, after it comes a **regular expression** enclosed in quotes
- We get a long way with just a little bit of regular expressions:
  - A dot `.` matches any character
  - A character group in square brackets matches any of the characters, e.g. `[ab]` matches `a` or `b`
  - "Quantifiers" count occurrences of the preceding expression
    - `*` zero or more
    - `?` zero or one
    - `+` one or more

Sample regular expressions

- Here are some examples of how regexes can match different forms in a paradigm

`"amat"`:	*amat*
`"ama[st]"`:	*amas* or *amat*
`"amat?"`:	*ama* or *amat*
`"ama[st]?"`:	*ama*, *amas* or *amat*

- But regexes are most useful in dealing with the **morph** and **pos** attributes

Structure of the **pos** attribute

- **pos** is a string with two characters
  - The first character gives the major part of speech, e.g. pronoun
  - The second gives the subtype, e.g. interrogative, relative etc.
  - "-" in the second field means there is no subtype
  - A complete list is found here:
    `http://folk.uio.no/daghaug/part_of_speech.yml`
- Examples

pos="Ne"	finds all proper nouns
pos="Nb"	finds all common nouns
pos="N[be]"	finds all proper and common nouns
pos="N."	finds all types of nouns (= in this case, proper and common)

Structure of the **morph** attribute

- **morph** works the same way as **pos** but there are ten fields!
- person, number, tense, mood, voice, gender, case, degree, strength, inflection
- this makes for very complicated strings!

Feature values

feature	possible values
person	1, 2, 3
number	s (singular), d (dual), p (plural)
tense	l (pluperfect), a (aorist), p (present), f (future), r (perfect), s (resultative), i (imperfect), t (future perfect), u (past)
mood	m (imperative), n (infintive), o (optative), d (gerund), p (participle), g (gerundive), s (subjunctive), i (indicative), u (supine)
voice	a (active), m (middle), e (middle or passive), p (passive)
gender	m (masculine), f (feminine), n (neuter), o (m. or n.), p (m. or f.), q (m., f. or n.), r (f. or n.)
case	n (nominative), a (accusative), g (genitive), d (dative), c (genitive or dative), i (instrumental), b (ablative), l (locative)
degree	p (positive), c (comparative), s (superlative)
strength	w (weak), s (strong), t (weak or strong)
inflection	i (inflecting), n (non-inflecting)

Examples

`"3sria----i"` Third person, singular, perfect, indicative, active, inflecting, e.g. *vidit*

`"---------n"` Non-inflecting, e.g. *ab*

`"-p---mb--i"` Plural, masculine, ablative, inflecting, e.g. *hominibus*

`"-s---qbc-i"` Singular, m./f./n., ablative, comparative, inflecting, e.g. *meliore*

Regular expressions over **morph**

- We are typically not interested in all the fields at once, so . is our friend
- For example, let us say we are interested in perfect tense verbs:
  - `[morph="..r.*"]`
- Or superlatives
  - `[morph=".......s.*"]`
- We can combine several constraints. Let us say we are interested in the distribution ablative endings in *-i* or *-e* in third declension adjectives
  - `[morph="......b.*" & pos="A." & word=".*[ie]"]`

Naming nodes

- In some contexts we may want to name the nodes
- A node name is an ASCII string preceded by #, e.g. #x
- They are attached to the node constraints with a colon
  - `#x:[morph=".....b.*" & pos="A." & word=".*[ie]"]`
- This allows us to give several constraints on a single node
  - `#x:[morph=".....b.*"] & #x:[pos="A."] & #x:[word=".*[ie]"]`
- In this case, the two queries are equivalent, but when we deal with syntax, we often **have to** use node names

Tables of results

- A nice effect of named nodes is that they allow us to tabulate the results by running the same query in the "query" view rather than the "sentence" view
- By default a table is constructed based on all node constraints
- However, if a node name ends with an underscore (_), its constraints are not tabulated
- So if we wanted to table only the word form and not the morph- or postag:
  - `#x_:[morph=".....b.*"] & #x_:[pos="A."] & #x:[word=".*[ie]"] & #x_=#x`

Querying syntax

- The syntactic structure is modelled as a tree, which is a 2-dimensional object and there are two corresponding operators that we need to know about
  - `>` The vertical dimension, dominance/government
  - `.` The horizonal dimension, precedence
- Examples
  - `#x >#y` Node *x* directly dominates node *y*
  - `#x . #y` Node *x* directly precedes node *y*
- * gives us the generalization of these operators
  - `#x >* #y` Node *x* dominates node *y*
  - `#x .* #y` Node *x* precedes node *y*

Labelled dominance

- In most cases we are not interested in pure dominance but in specific syntactic **functions**, which are captured in **labels**
- To use them properly you need to carefully study the syntactic guidelines of the corpus you are using, in this case `http://folk.uio.no/daghaug/syntactic_guidelines.pdf`
- For now we will stay with the following labels familiar from traditional grammar

sub	subject
obj	object
adv	adverbial
atr	attribute
comp	complement clause

Deictics inside complement clauses

- When e.g. *nunc* occurs within reported speech there is ambiguity whether it refers to the speaker's now or the narrator's?

Embedded *nunc*

#x >comp #c & #c >* [word="nunc"]

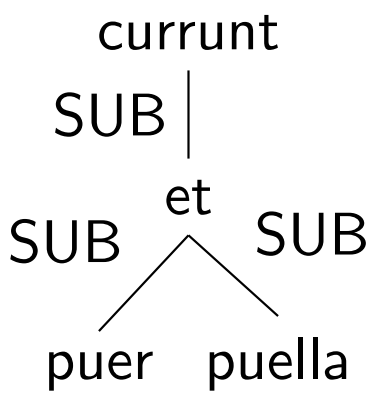
- The first part ensures that #c has the COMP function
- The second part ensures that *nunc* is dominated by #c

Predicate-subject pairs

`#x >sub #y` finds all subjects

`#x:[lemma=".*"] >sub #y:[lemma=".*"]` tabulates per lemma

- False positives because of the analysis of coordination



- Fixed with `#x:[lemma=".*"] >((sub)+) #y:[lemma=".*"] & #x_:[pos="V-"] & #y_:[pos!="C-"] & #x = #x_ & & #y = #y_`

Word order - Kirk's criteria

- Finding a main clause
  - Unbroken sequence of PRED relations: `#r >((pred)*) #v`
  - Under the root: `!(#x > #r)`
  - Not an elided verb: `#v:[pos="V-"]`
- With a subject and an object: `#v >sub #s & #v >((obj)|(obl)) #o`
- ...which are nominals : `#s:[pos="[AN].*"] & #o:[pos="[AN].*"]`
- ...and not discontinuous: should have been `_cont(#s) & _cont(#o)` (does not currently work!)
- No auxiliares: `!(#v >aux #aux:[pos=".+"] )`
- Build your query step by step!

The result

```
#r >((pred)*) #v & !(#x > #r) & #v:[pos="V-"] & #v >sub #s
& #v >((obj)|(obl)) #o & #s:[pos="[AN].*"] &
#o:[pos="[AN].*"] & !(#v >aux #aux:[pos=".+"] )
```

- Pretty daunting, but after all we found the data material for a whole dissertation in a matter of minutes
- Moreover, it is flexible - we can tweak the query and look at various other definitions of the source material
- That is for the exercises!

Exercises

- Change the query so that we allow pronominal subjects and objects
- Change the query so that we only look at complement clauses
- Change the query so that we only look at adverbial clauses
- There are six possible permutations of S, V and O. Express these permutations as INESS queries that can be added to our base query
- Change the query so that you find orders of subject, verb and complement clause (instead of object) and specify different types of complement clauses
  - accusative with infinitive
  - finite complement clause