Status Quo efter lektion 3

Opdatering af lt-course mappen

Vi startede dagen med at rette et problem med lt-course mappen, nemlig at den var kommet ud af takt med det der ligger på Github, så den kunne ikke opdateres via `git pull`

- Slet hele lt-course mappen
```
cd
rm -rfv lt-course
```
- … og så hent en frisk kopi fra Github
```
git clone https://github.com/Oqaasileriffik/lt-course
```

Fordi sådanne ting sker hvor man er nødt til at slette og hente igen, er det smart at have sit arbejde i en mappe der ikke er under versionskontrol. Det var bl.a. derfor i oprettede mappen ~/work i lektion 2 - hvis i havde lavet *-frekvens.txt filerne i mappen ~/lt-course/lecture01/corpus/ så ville `rm` have fjernet dem. Det er også vigtigt at have backups.

Formatet af analyseret tekst

Derefter kiggede i på hvordan analyseret tekst ser ud, her det første af tre corpora som Per Langgård manuelt har sørget for ikke har for mange fejl:
```
cd ~/lt-course/lecture03/corpus
cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | more
```

Analyseret tekst ser ud som:
```
"<Arfinilinnik>"
    "arfinillit" Num Ins Pl @i->N
"<ukioqarlunga>"
    "ukioq" QAR Der/nv Gram/IV V Cont 1Sg @ADVL>
```

De morfologiske tags er dokumenteret under https://giellalt.uit.no/lang/kal/ og https://giellalt.uit.no/lang/kal/root-morphology.html, og de syntaktiske tags er dokumenteret i PDF'en Kalaallisut Syntaktiske Tags.pdf som ligger i mappen ~/lt-course/lecture03/corpus/.

Så vi kan se at fuldformen *Arfinilinnik* er morfologisk analyseret som roden *arfinillit* med ordklassen *Numeral* i

kasus *Instrumentalis* og numerus *Pluralis*, og er tagget med syntaktisk funktion *Adled til inderiveret nomen til højre*.

Tilsvarende, fuldformen *ukioqarlunga* er morfologisk analyseret som roden *ukioq* med derivationsmorfemet *QAR* som ændrer ordklassen fra *nomen til verbum* og er *intransitiv*, med den endelige ordklasse *Verb* i modus *Infinitive* og person-numerus *Subject 1. person singular*, og er tagget med syntaktisk funktion *Adverbial med dependenshoved til højre*.

Terminologi: Fuldform, surface form, word form, overfladeform - alle det samme begreb, nemlig ordet som det forekommer i teksten. Disse er i modsætning til termerne base form, lemma, rod, root, som alle er ordets mindste først bid som der bygges videre på med derivation.

Opbygning af Kalaallisut-analysatoren

Så gennemgik vi hvordan Kalaallisut-analysatoren er opbygget. Altså, hvilke datafiler der bruges til at komme fra fuldformen `Arfinilinnik` til det analyserede `"arfinillit" Num Ins Pl @i->N`.

Et kort rundown, her gengivet som et træ af mapper og filer. Hvert indrykket niveau betyder en undermappe af foregående niveau:

`~/langtech/`	
`giella-core/`	Hjælpefiler fra Giellatekno - må ikke redigeres
`giella-shared/`	… ditto ...
`regression/`	Tests der checker om de ændringer man har lavet i kal gør noget uforventet
`kal/`	Kalaallisut-analysatoren
`src/`	
`morphology/`	Morfologisk analysator, lavet med LEXC-filer
`root.lexc`	Definition af alle tags og andre morfologiske stumper
`stems/`	Mappe med rødder og leksikaliserede rod-lignende former

`abbreviations.lexc`
`nouns.lexc`
`particles.lexc`
`propernouns.lexc`
`acronyms.lexc`
`numerals.lexc`
`pronouns.lexc`
`verbs.lexc`
	LEXC-filer med rødder, opdelt efter ordklasser. Hver rod sender videre til mulige derivationskæder.

`affixes/`	Mappe med derivationer

`derivations-inflections.lexc`
`numerals.lexc`
`propernouns.lexc`
`symbols.lexc`
	LEXC-filer med derivationskæder.

`syntax/`	Morfologisk disambiguator og syntaktisk tagger, lavet med Constraint Grammar-filer

`kal-pre1.cg3`	1 regel, som fjerner paragraf-brud efter kolon
`kal-pre2.cg3`	19 regler, som forbereder håndtering af direkte tale
`disambiguator.cg3`	3390 regler, som udfører både morfologisk disambiguering og syntaktisk tagging

I åbnede filerne ~/langtech/kal/src/morphology/root.lexc og ~/langtech/kal/src/syntax/disambiguator.cg3 i en text editor.

Mht. shortcuts til at åbne filer grafisk:

- Jer med VirtualBox kan åbne filerne grafisk fra terminalen ved at køre e.g.
  ```
  cd ~/langtech/kal/src/syntax/
  gedit disambiguator.cg3 &
  ```
  eller direkte
  ```
  gedit ~/langtech/kal/src/syntax/disambiguator.cg3 &
  ```
  Den sidste `&` betyder at kommandoen skal lægge sig i baggrunden - uden den så kan man ikke bruge den terminal før den åbnede `gedit` lukkes.
- Jer med native macOS kan tilsvarende bruge `open` - https://scriptingosx.com/2017/02/the-macos-open-command/
- Jer med forbindelse til serveren i Canada (juak og doma) har desværre ingen shortcuts. I bliver nødt til hhv. at bruge Notepad++ og Cyberduck til at navigere og åbne filerne grafisk.

Men, det at kunne navigere det grafiske filsystem ved siden af hvordan man navigerer i terminalen er vigtigt. Man bliver ikke så nemt væk hvis man kan lære at visualisere at `cd` betyder det samme som at dobbelt-klikke på en bestemt mappe.

Gennemgang af Kalaallisut-analysatoren

Derefter gik vi gennem hvilke led Kalaallisut-analysatoren har og hvad hvert led gør.

- Vis hvilke funktionaliteter analysatoren har
```
~/lt-course/lecture03/scripts/kal-analyse --help
```

Dette giver output
```
Possible options are:
    --help, -h, ?
    --trace, -t
    --from, -f [breakpoint]
    --regtest
    --cmd
    --raw
Pipe breakpoints:
    --fst
    --pre1
    --hybrids, --hyb
    --pre2
    --morf
    --syntax, --syn
    --all
```

Komma betyder alias, så e.g. `--syntax` og `--syn` er det samme. Det vigtige her er den nederste del, efter `Pipe breakpoints`. Det er analyse-niveauerne som kal-analyse er i stand til at udføre. De er:

`--fst`	Tokenisering og morfologisk analyse. Opdeler inputtet i ord og giver alle mulige analyseformer for ordene, også de usandsynelige. Niveauet hedder `--fst` fordi i daglig tale kalder vi den morfologiske analyse for den teknologi den er lavet med, nemlig FST (Finite State Transducer).
`--pre1`	Fjerner paragraf-brud efter kolon og punktum.
`--hybrids, --hyb`	Deler hybrider. I inputtet står hybrider som ét token, men det er lingvistisk svært at håndtere så de bliver splittet til 2 separate tokens.
`--pre2`	Markerer hvor direkte tale mest sandsyneligt starter og slutter.
`--morf`	Morfologisk disambiguering. Blandt alle de mulige analyser af ordene, forsøger dette niveau at vælge den korrekte analyse ud fra konteksten.
`--syntax, --syn`	Syntaktisk tagging. Sætter den syntaktiske funktion på ordene, igen ud fra konteksten.
`--all`	Rydder en smule op.

Niveauerne `pre1`, `pre2`, `morf`, og `syn` tager optionelt også `--trace` (alias `-t`) for at vise hvilke regler der udfører arbejdet, så når systemet laver en fejl er det ikke svært at finde ud af hvorfor, så man kan rette på det. Hvert niveau's output er det næste niveau's input, så når man beder om et niveau bliver alle foregående niveauer også kørt. Hvis man ikke giver et niveau er det det samme som hvis man havde skrevet `--all`.

Brug af analysatorens niveauer

Så lavede vi hands-on ved at køre en sætning gennem hvert niveau for at se hvad de gør.

- Sætningen *Andap tujuuluk pisiaraa.* gennem den morfologiske analyse

```
cd ~/lt-course/lecture03/scripts/
echo 'Andap tujuuluk pisiaraa.' | ./kal-analyse --fst
```

- ...giver outputtet

```
"<Andap>"
    "Anda" Sem/Mask Sem/Hum Prop Rel Sg
"<tujuuluk>"
    "tujuuluk" N Abs Sg
"<pisiaraa>"
    "piseq" ARAQ Der/nn N Abs Sg 3SgPoss
    "pisi" ARAQ Der/nn N Abs Sg 3SgPoss
    "pisiaq" GE Der/nv Gram/TV Gram/Refl Gram/IV V Int 3Sg
    "pisiaq" GE Der/nv Gram/TV V Ind 3Sg 3SgO
    "pisiaq" N Abs Sg AA
"<.>"
    "." CLB
```

```
"<¶>"
    "¶" CLB
```

Ordene *Andap* og *tujuuluk* er der ikke tvivl om - de er morfologisk entydige. Men *pisiaraa* kan læses på 5 forskellige måder. Når man som menneske læser en tekst har man allerede konteksten i hovedet, så man fravælger fra starten mange muligheder. Det kan en computer ikke gøre - vi bliver nødt til at få alle mulighederne, og skal så vælge en af dem senere.

- Videre til `pre1`-niveauet

```
   echo 'Andap tujuuluk pisiaraa.' | ./kal-analyse --pre1
```

- … outputtet er næsten ikke forskelligt, der fjernes kun det sidste paragraf-tegn.

Niveauerne `hybrids` og `pre2` gør heller ingen forskel på denne simple sætning, så skipper dem.

- Sætningen gennem morfologisk disambiguering

```
   echo 'Andap tujuuluk pisiaraa.' | ./kal-analyse --morf
```

- … giver output

```
"<Andap>"
    "Anda" Sem/Mask Sem/Hum Prop Rel Sg
"<tujuuluk>"
    "tujuuluk" N Abs Sg
"<pisiaraa>"
    "pisiaq" GE Der/nv Gram/TV V Ind 3Sg 3SgO
"<.>"
    "." CLB
```

Altså, 4 af de 5 mulige analyser for *pisiaraa* er blevet fjernet, og den overlevende skulle gerne være den korrekte. For at se hvilke regler der udførte dette arbejde kan der tilføjes `-t` til niveauet.

- Sætningen gennem morfologisk disambiguering, med debugging-information

```
   echo 'Andap tujuuluk pisiaraa.' | ./kal-analyse --morf -t
```

- … giver output

```
"<Andap>"
    "Anda" Sem/Mask Sem/Hum Prop Rel Sg
"<tujuuluk>"
    "tujuuluk" N Abs Sg
"<pisiaraa>"
    "pisiaq" GE Der/nv Gram/TV V Ind 3Sg 3SgO
;   "piseq" ARAQ Der/nn N Abs Sg 3SgPoss SELECT:2583:0003
;   "pisi" ARAQ Der/nn N Abs Sg 3SgPoss SELECT:2583:0003
;   "pisiaq" GE Der/nv Gram/TV Gram/Refl Gram/IV V Int 3Sg SELECT:2583:0003
;   "pisiaq" N Abs Sg AA REMOVE:1858:0001AK
"<.>"
    "." CLB
```

Linier der starter med `;` er blevet fjernet. Information om hvilken regel der har gjort det står til sidst på linien, e.g. den sidste linie er fjernet af `REMOVE` reglen på linie `1858` med navnet `0001AK`. Diverse text editors har mulighed for at hoppe direkte til et linienummer, enten fra menuen eller med keyboard shortcut.

- Sætningen gennem syntaktisk tagging

```
   echo 'Andap tujuuluk pisiaraa.' | ./kal-analyse --syn
```

- … giver output

```
"<Andap>"
    "Anda" Sem/Mask Sem/Hum Prop Rel Sg @SUBJ>
"<tujuuluk>"
    "tujuuluk" N Abs Sg @OBJ>
"<pisiaraa>"
    "pisiaq" GE Der/nv Gram/TV V Ind 3Sg 3SgO @PRED
"<.>"
    "." CLB
```

Søgning i analysen

Derefter kiggede vi på hvad man kan bruge sådan analyserede data til. Specifikt, at finde eksempler på hvordan den syntaktiske funktion @MIK-OBJ (*Objekt i instrumentalis*) bliver brugt i de 3 corpora.

- Søg på @MIK-OBJ der lægger sig til højre

```
   cd ~/lt-course/lecture03/corpus/
   cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | grep '@MIK-OBJ>' | more
```

- … giver output

```
    "assiliaq" N Ins Sg @MIK-OBJ>
    "silassorip" Gram/IV PASIP Der/vv GE Der/vv Gram/TV TAQ Der/vn N Ins Sg 1SgPoss @MIK-OBJ>
    "sava" N Ins Sg @MIK-OBJ>
    "sava" N Ins Sg @MIK-OBJ>
    "sava" N Ins Sg @MIK-OBJ>
```

- … optæl hvor mange af dem der er

```
   cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | grep '@MIK-OBJ>' | wc -l
```

- … giver output

```
   46
```

- Søg på @MIK-OBJ der lægger sig til højre, men vis mere kontekst

```
   cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | grep -C7 '@MIK-OBJ>' | more
```

`grep -C7` betyder at for hver linie der matcher skal der også vises 7 linier før og efter. Man kan bede om så mange eller så få linier man vil, og man kan bede om forskelligt antal før og efter. `-C7` er det samme som `-B7 -A7` - hvis man i stedet vil have 9 linier før og 3 linier efter kan man sige `-B9 -A3`.

Men, @MIK-OBJ kan både lægge sig til højre og venstre, og en søgning på `@MIK-OBJ>` finder kun til højre. Vi vil også gerne se på de andre.

- Søg på @MIK-OBJ der lægger sig til venstre, med kontekst

```
   cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | grep -C7 '@<MIK-OBJ' | more
```

- … optæl hvor mange af dem der er (uden kontekst)

```
   cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | grep '@<MIK-OBJ' | wc -l
```

- … giver output

```
   2
```

Så @MIK-OBJ bliver i dette corpus mest brug i formen der lægger sig til højre.

- Tilsvarende for de 2 andre corpora:

```
   cat AviscorpusDisambigueretOgLettereJusteret.txt | grep '@MIK-OBJ>' | wc -l
   cat AviscorpusDisambigueretOgLettereJusteret.txt | grep '@<MIK-OBJ' | wc -l
   cat UkiutTrettenitDisambigueretOgLettereJusteret.txt | grep '@MIK-OBJ>' | wc -l
   cat UkiutTrettenitDisambigueretOgLettereJusteret.txt | grep '@<MIK-OBJ' | wc -l
```

- … giver hhv. outputs

```
   17
   2
   51
   3
```

Vi kan altså se at for alle 3 teksttyper bliver @MIK-OBJ brugt på samme måde, nemlig langt mest til højre.

Hvis man gerne vil have alle former for @MIK-OBJ, er det besværligt at skulle lave 2 eller flere søgninger, så man kan lægge dem sammen til én søgning ved hjælp af Regular Expressions (regex). Se også https://krijnhoetmer.nl/stuff/regex/cheat-sheet/

- Søg på begge former for @MIK-OBJ, her med en regex `|` alternative

```
   cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | egrep -C7 '@<MIK-OBJ|@MIK-OBJ>' | more
```

- … og her med regex `?` zero-or-one wildcard

```
   cat AtaqqinartuaraqDisambigueretOgLettereJusteret.txt | egrep -C7 '@<?MIK-OBJ>?' | more
```

Opgave til mandag d. 10

At sammenligne hvordan oratio obliqua bliver brugt i de 3 corpora. Dvs. de syntaktiske tags @CL-CIT> og @CL-<CIT. Både sprogligt ved at kigge på konteksterne af matches, og numerisk ved at sammenligne antallet af forekomster med størrelsen af corpuset man arbejder med.

Husk at man kan skrive resultater til en fil med `>` og så åbne disse filer i en text editor hvor man nemmere kan danne sig et overblik.

De 3 corpora er:

- Ataqqinartuaraq — Den Lille Prins, oversat fra fransk til godt grønlandsk
- Aviscorpus — NRK og Sermitiaq AG, oversat fra dansk til dårligt grønlandsk
- UkiutTrettenit — Modersmålsnovelle

Og i kan også inkludere Facebook-corpuset hvis i først kører det gennem `kal-analyse`.

Noter & Hints

I første lektion blev der spurgt om hvordan man kunne finde de længste ord i et corpus. Det kan gøres således:

```
   cat Ataqqinartuaraq.txt | tr '., !"”:?-' '\n' | sort | uniq | awk '{print length, $0}' | sort -n -r | more
```

Dette finder ordet *ataqqinartorujussuanngussutiginavianngilara* på 43 bogstaver. Det fungerer ved at først finde alle de unikke ord, og så bede `awk` om at præfixe længden af linien, og så sortere på længden. `awk` er et helt programmeringsprog som i ikke behøver at kunne, men det vil ofte være det man finder når man søger på Google. E.g., hvis jeg søger på http://google.com/search?q=Linux+sort+line+length så er første resultat https://stackoverflow.com/a/5917762/145919 som benytter `awk`.

I anden lektion blev der spurgt om hvordan man kunne finde alle ord der indeholder *neqar*, men dette var svært at gøre på fuldformsniveau. Nu hvor vi har en morfologisk analyse kan dette nemt lade sig gøre, ved at søge på derivationsmorfemet NIQAR i analysen, her med `-B1` for at få fuldformen med:

```
cat AviscorpusDisambigueretOgLettereJusteret.txt | grep -B1 'NIQAR' | more
```