

## Kort introduktion til eksempler og opgaver om

- **Prolog som simpel vidensbase**
- **Definite Clause Grammars for sprogbeskrivelse og -analyse**

### Indledning

Notatet beskriver de eksempler vi benytter i undervisningen og introducerer opgaver som bygger videre på disse eksempler. Der gives kun begrænset indføring i Prologs begreber og der henvises til anden litteratur, som er omtalt på kursets hjemmeside.

Formålet med øvelsen her er dels at introducere til de centrale værktøjer, Prolog og DCG, og at give de studerende en mulighed for, selv med en begrænset introduktion, at opbygge et meget enkelt analyse- og evt. dialogsystem.

Det meste af den viste kode er tilgængelig på kursets hjemmeside.

### En vidensbase i Prolog

En af de ting man kan skrive i et Prologprogram er en sekvens af fakta, der fungerer som en simpel videns- eller database. Vi benytter følgende program som er vist i den korrekte syntaks — bemærk dog at »...« ikke er en del af programmet men blot antyder »flere af samme slags«.

```
foraelder(margrethe, frederik).
foraelder(margrethe, joachim).
foraelder(henrik, frederik).
foraelder(henrik, joachim).
...
elsker(margrethe, henrik).
elsker(henrik, margrethe).
elsker(joachim, alexandra).
...

kvinde(ingrid).
...
mand(frederikIX).
...
```

Hvis nu dette program ligger på en fil med navn familie, og man har fået startet Prolog-systemet op i det rette arbejdskatalog (=folder=mappe), kan det indlæses som følger:

```
| ?- [familie].
```

(Det er Prolog som skriver »| ?-« og brugeren som skriver resten til og med punktum og et efterfølgende liniskift.

Nu kan der stilles forespørgsler til systemet, hvor det så svarer ja eller nej til om den genkender det forespurgte.

```
| ?- foraelder(alexandra, felix).
yes
| ?- foraelder(mary, felix).
no
| ?- quaximox(alexandra).
! Existence error in user:quaximox/1
! procedure user:quaximox/1 does not exist
! goal: user:quaximox(alexandra)
```

Bemærk alle symbolske konstanter, her personnavne, er stavet med småt. Det er konventionen i Prolog, og ting stavet med stort, f.eks. »X« repræsenterer variable.

```
| ?- foraelder(alexandra,X).
X = nikolai ? ;
X = felix ? ;
no
| ?- foraelder(X,felix).
X = joachim ? ;
X = alexandra ? ;
no
```

Her spurgte vi »Give mig et X så ...«. Det er brugeren som har tastet semikolon for at bede om flere svar, og »no« skal her forstås som »ikke flere svar«.

Vi kan bruge variable til at udtrykke komplekse sammenhænge i sammensatte forespørgsler.

```
| ?- foraelder(frederikIX,X), foraelder(X,Y).
X = margrethe,
Y = frederik ? ;
X = margrethe,
Y = joachim ? ;
no
```

Her var det tydeligvis et bedsteforælder/barnebarns-forhold der blev spurgt om.

Hvad med følgende forespørgsel:

```
| ?- foraelder(X,margrethe), foraelder(X,Y), kvinde(Y).
```

Her spørges om et X som er fælles forælder til Margrethe og Y. Vi må formode, at vi får Margrethes søstre som svar for Y.

Men lad os se:

```
| ?- foraelder(X,margrethe), foraelder(X,Y), kvinde(Y).
X = frederikIX,
Y = margrethe ? ;
X = frederikIX,
Y = benedikte ? ;
X = frederikIX,
Y = anne_marie ? ;
X = ingrid,
Y = margrethe ? ;
X = ingrid,
Y = benedikte ? ;
X = ingrid,
Y = anne_marie ? ;
no
```

Kun næsten det vi troede, vi får søstre men også Margrethe selv — vel i overensstemmelse med logikken, men ikke det vi ønskede, for man kan da ikke være søster til sig selv. Det, at systemet kunne identificere søster- (m.v.) forholdet på to måder er ikke noget logisk problem, vi får blot svarene for Y gentaget.

Der findes en indbygget betingelse til at teste (og garantere i sidste ende!) at to størrelser er forskellige; den hedder »dif« og benyttes som følger.

```
| ?- foraelder(X,margrethe), foraelder(X,Y), kvinde(Y),
dif(margrethe,Y).
X = frederikIX,
Y = benedikte ? ;
X = frederikIX,
Y = anne_marie ? ;
X = ingrid,
Y = benedikte ? ;
X = ingrid,
Y = anne_marie ? ;
no
```

Nu får vi netop søstre, »erkendt« korrekt på to forskellige måde.

Prolog indeholder selvfølgelig også *regler*, så vi kan udtrykke en gang for alle, hvad der forstås ved en søster. Det kan vi gøre ved at tilføje følgende til `familie`-filen, som viser det generelle format for regler:

```
soester(X, Y):-  
    foraelder(Z, X),  
    foraelder(Z, Y),  
    kvinde(X),  
    dif(X, Y).
```

Indprent dig detaljerne, kolon-streg mellem hovede og krop, komma imellem de enkelte betingelser i kroppen og punktum til sidst.

Vi kan spørge nu sådan her:

```
| ?- soester(margrethe,A).  
A = benedikte ? ;  
A = anne_marie ? ;  
A = benedikte ? ;  
A = anne_marie ? ;  
no
```

Resultaterne er korrekte, og da vi kender den interne mekanik fra tidligere, skal vi ikke gå i panik over at svarene kommer to gange.

### Opgaver til vidensbaser

- Leg med vidensbasen, stil alle former for mulige og umulige forespørgsler til den.
- Udvid den med flere medlemmer af den kongelige familie.
- Tilføj og afprøv regler som definerer følgende relationer:  
    `bedsteforaelder(X, Y)`  
    `mor(X, Y)`  
    `far(X, Y)`  
    `bror(X, Y)`

## Definite clause grammars

Denne grammatiknotation som vi vi viser nedenfor, er indbygget i Prologsystemet, så man kan altså skrive dem direkte som en del af sine Prologprogrammer. Men lad os allerførst illustrere den lille delmængde af dansk vi vil arbejde med, og antyde en grammatik uformelt. (Tilgiv forfatteren, hvis de grammatiske betegnelser ikke benyttes helt korrekt).

Eksempler:

Margrethe er mor til Frederik.  
Hvem er mor til Frederik.  
Frederik IX er bedstefar til hvem?  
Felix sover.  
Hvem elsker Mary?

Det generelle mønster er noget i stil med:

En *sætning* er en *nominalfrase* efterfulgt af en *verbsfrase*.

En *nominalfrase* er her begrænset til navne på kongefamilien og spørgeordet »hvem«

En *verbsfrase* er enten et *intransitivt verbum* (f.eks. »sover«), et *transitivt verbum* (f.eks. »elsker«) efterfulgt af nok en *nominalfrase*, eller hjælpeverbet »er« efterfulgt af en *prædikation*.

En *prædikation* er enten angivelse af en *kategori* (f.eks.»en mand«) eller en *relation* efterfulgt af en *nominalfrase*.

En *relation* er begrænset til ting vi kender fra vidensbasen, f.eks. »søster til«.

### Første grammatikopgave uden DCG

- Forklar hvordan eksempelsætningerne ovenfor er opbygget i forhold til den uformelt skitserede grammatik.

Vi kan illustrere DCG-notationen med en grammatik som lige akkurat kan klare sætningen »Felix sover«.

```
s --> np, vp.
```

```
vp --> iv.
```

```
iv --> [sover].
```

```
np --> navn.
```

```
navn --> [felix].
```

Konkrete sproglige symboler, ofte kaldet *terminalsymboler* eller *leksikalske symboler*, er angivet i kantede parenteser, de øvrige symboler som »s« (for sætning) osv. kaldes *nonterminaler* eller *syntaktiske kategorier*. Disse regler skal forstås som genskrivningsregler, så man starter med et »s« som man så ved gentagen brug af regler får lavet om til en streng af terminalsymboler.

```
s → np, vp → navn, vp → [felix], vp → [felix], iv → [felix, sover]
```

Notationen med kantede parenteser er Prologs (og DCGs) notation for generelle lister, og eksemplerne ovenfor viser at man i DCG repræsenterer en frase som en liste af symboler.

Vi kan ikke umiddelbart tale til et Prolog-system eller bede det om at kigge på et pdf-dokument, men er begrænset til at stille forespørgsler i stil med det vi har set tidligere.

Antag nu, vi har en fil med navnet `gram` som indeholder de fem DCG-regler ovenfor. Vi kan læse den ind i systemer som enhvert Prolog-program.

```
| ?- [gram].
```

Analyse af en tekst foregår på følgende måde:

```
| ?- phrase(s, [felix, sover]).
```

```
yes
```

```
| ?- phrase(s, [margrethe, cykler]).
```

```
no
```

Som forventet accepterer den kun de sætninger som eksplicit kan genereres af grammatikken. Vi spurgte altså systemet, om den givne liste kunne genereres som en »s« og systemet kombinerer så reglerne for at afgøre spørgsmålet.

En interessant og måske lidt overraskende egenskab er at vi også kan generere sætninger vha. grammatikken:

```
| ?- phrase(s,X).  
X = [felix,sover] ? ;  
no
```

Vi spørger altså systemer, hvilke s-er der findes, og med den simple grammatik er der kun den ene.

Når vi engang får udvidet grammatikken med flere sætningstyper vil der komme flere svar.

### Opgave med simple DCGer

- Udvid felix-sover-grammatikken så den dækker alle de sætningstyper vi har betragtet som mulige ovenfor. Hver gang du har skrevet en ny eller ændret en regel, så læs grammatikken ind i systemet og aftest med det samme.  
NB: Det anbefales at se bort fra »?« i spørgsmål!

## Udvidelse af DCGerne med semantisk repræsentationer

Nonterminalerne i en DCG kan udstyres med argumenter ligesom de prædikater vi så i første afsnit.

Lad os tage et minimalistisk eksempel, hvor vi har nonterminalen *k* med et argument som er enten 1 eller 2 (tallene fungerer her som symboler akkurat som *margrethe* og *mary*).

```
k(1) --> [a].  
k(2) --> [b].
```

Følgende forespørgsel kan forstås som »hvad slags *k* er [*a*]?«.

```
| ? phrase(k(X),[a]).  
X = 1
```

Hvis vi omvendt vil generere fraser, kan vi angive hvilken slags:

```
| ? phrase(k(2),Fraser).  
Fraser = [b]
```

Nu tager vi et relativt stort spring ved at indføre generelle termer i Prolog. Udover lister og symboler har vi en slags dataværdier som kaldes *strukturer* — og tilsammen kaldes alle disse for *termer*. I et Prolog-program kan man f.eks. skrive:

$p(f(a,b))$ .

Det læses som at strukturen  $f(a,b)$  opfylder betingelsen  $p$ . Vi kan spørge:

| ?-  $p(X)$ .  
 $X = f(a,b)$

Det smarte ligger i at vi kan flytte variable ind i strukturerne:

| ?-  $p(f(X,Y))$ .  
 $X = a$   
 $Y = b$

Sådanne strukturer er praktiske i forhold til sprog. Vi kunne, hvis det er det vi ønsker, udstyre vores grammatik så den bygger såkaldte syntakstræer, som repræsenterer strukturen i en sætning, dvs. som modsvarer de regler som er blevet kaldt. Med en grammatik, som vi ikke har vist, kan vi få genereret følgende repræsentation ud fra `[felix,sover]`:

| ? - `phrase(s(S),[felix,sover])`.  
 $S = s(\text{np}(\text{pn}(\text{felix})), \text{vp}(\text{iv}(\text{sover})))$

Det angiver, som vi kan se, en detaljeret beskrivelse af, hvordan den omtalte sætning er konstrueret. Det kræver at man sætter lidt flere ting ind i grammatikreglerne, men det er ikke det vi er interesseret i her.

I stedet vil vi finde en mere direkte repræsentation, som i en eller anden forstand beskriver det udsagn, som en sætning indeholder uden diverse syntaktisk detaljer.

Med den grammatik — som vi viser om et øjeblik — vil vi få følgende svar:

| ? - `phrase(s(S),[felix,sover])`.  
 $S = \text{egsk}(\text{felix}, \text{sovende})$

Strukturbyggeren »egsk« er tænkt at udtrykke der forhold at én ting (her »felix«) har en bestemt egenskab (her »sovende«). Her følger en udvidet grammatik for felix-sover-sproget, som producerer dette svar. Bemærk, at vi benytter symbolet »?« på en særlig måde til at repræsentere »et-eller-andet-ukendt«. Rent teknisk er »?« her bare et symbol helt på linje med »margrethe« og »2«, men det er forfatteren af grammatikken som benytter det på en særlig måde.

$s(\text{egsk}(X, \text{Hvad})) \rightarrow np(X), vp(\text{egsk}(?, \text{Hvad}))$ .

$vp(X) \rightarrow iv(X)$ .

$iv(\text{egsk}(?, \text{sovende})) \rightarrow [\text{sover}]$ .

$np(X) \rightarrow \text{navn}(X)$ .

$\text{navn}(\text{felix}) \rightarrow [\text{felix}]$ .

Vi kan læse reglerne højt således (vi tager dem her i omvendt rækkefølge):

Teksten svarende til navnet `[felix]` repræsenterer symbolet `felix`.

En `np` som består af et navn repræsenterer det samme som navnet gør.

Det intransitive verbum »`sover`« repræsenterer egenskaben at nogen sover, og når vi ser på dette verbum alene, ved vi jo ikke hvem der sover – og derfor spørgsmålstegnet, og altså i alt termen `egsk(?, sovende)`.

Når vi danner en sætning ved at kombinere en `np` (som repræsenterer et individ eller en entitet `X`) med en `vp` (som repræsenterer en egenskab) betyder det, at dette `X` har bemeldte egenskab.

I eksemplet er individet angivet ved `felix` og egenskaben ved `egsk(?, sovende)`, og `s`-reglen beskriver hvordan `felix` indsættes i stedet for spørgsmålstegnet.

Vi vil omtale sådanne termer som *semantiske termer*. Fra et filosofisk synspunkt er det en smule kritisabelt at kalde disse termer semantiske, men det vi blot mener er at de kan fungere som grænseflade til vor vidensbase; mere om dette senere.

I en opgave nedenfor skal du sætte generering af semantiske termer på hele den grammatik du selv konstruerede i et tidligere opgave; nu forklares hvordan disse termer skal se ud.

I `felix-sover`-eksemplet er det ét enkelt individ, som besidder én egenskab, men andre sætninger beskriver relationer mellem to individer. For eksempel kan vi forestille os den semantiske term

`rel(frederik, elsker, mary)`

som »indholdet« af sætningen `[frederik, elsker, mary]`. Verbet »`elsker`« set ude af sin sammenhæng repræsenterer en relation som kan gælde mellem to individer ad gangen, men ikke hvilke. Vi kan altså vælge som repræsentation af »`elsker`« den semantiske term

`rel(?, elsker, ?)`

og når verbet sammen med et efterfølgende nominalled danner en vp, får vi nu følgende repræsentation, hvor så at sige den ene ubekendte er gjort kendt:

```
rel(? , elsker , mary)
```

Dvs. en symbolsk angivelse af »det at elske Mary«, og det kunne der jo godt være flere der gjorde.

Vort sprog kan også udtrykke andre relationer, f.eks. kan vi opfatte [ soester , til ] set isoleret som repræsenterende indholdet `rel(? , soester_til , ?)`.

Spørgeorder »hvem« er lidt problematisk. Vi kan ganske enkelt vælge at »hvem« har betydningen svarende til symbolet »?«, dvs. den samme form for ubekendt som vi brugte for ukendte led. Rent intuitivt kan vi argumentere således:

```
vpen [ elsker , mary ] udtrykker det at elske Mary:  
rel(? , elsker , mary)
```

```
sætningen [ hvem , elsker , mary ] udtrykker det samme, bortset fra at ordet  
»hvem« rent syntaktisk er klistret på for at angive at det er altså er et spørgsmål.
```

Altså: Meningen med sætningen [ hvem , elsker , mary ] angives ved `rel(? , elsker , mary)`. Og vi kan gå videre: meningen med [ hvem , elsker , hvem ] er `rel(? , elsker , ?)`.

### DCG-opgave med semantiske termer

- Tag den grammatik du har konstrueret tidligere og udvid den, så den genererer semantiske termer. Start med de simpleste konstruktioner først, og afprøve hver gang du har lavet en ændring/tilføjelse.
- Når det hele virker, overbevis dig om at du også kan generere sætninger ud fra angivelse af semantisk indhold.

```
| ?- phrase(s( rel(? , elsker , ?) ), Tekst).
```

## Til slut: Et dialogsystem

Vi har nu en vidensbase, vi har en grammatik som kan skabe semantiske repræsentationer ud fra sætninger — og sætninger ud fra semantiske repræsentationer.

Vi har altså brikkerne til et enkelt spørgsmål-svar-system, som minder om den type samtaler man fører på lektion nr. 2 i et sprogkursus. Lidt i stil med følgende:

```
| ?- besvar([margrethe,er,mor,til,frederik], Sv).  
Sv = [margrethe,er,mor,til,frederik]
```

```
| ?- besvar([frederik,er,mor,til,margrethe], Sv).  
Sv = nej
```

```
| ?- besvar([margrethe,er,mor,til,hvem], Sv).  
Sv = [margrethe,er,mor,til,frederik];  
Sv = [margrethe,er,mor,til,joachim]
```

Følgende Prolog-regel kunne være et forslag til den del som behandler sætninger af formen »Hvem er mor til ...«

```
besvar(Sp,Sv):-  
    phrase(s(rel(? ,mor_til,Y)), Sp),  
    mor(X,Y),           % dvs. spørg vidensbasen  
    phrase(s(rel(X,mor_til,Y)), Sv).
```

Du kan jo nok regne ud, at med dette princip skal der en helt del besvar-regler til som dækker hvert specialtilfælde, men din lærer kan vise dig nogle Prolog-fiduser, så det kan gøres lidt nemmere.