LOGISK PROGRAMMERING

Prolog (kapittel 8):

- Fakta
- Regler
- Spørsmål
  - Variable
  - Hvordan finne svar?
- Unifikasjon
- Lister

Hoved-prinsipp:

Hva istedenfor Hvordan!

Logisk programmering

Programmerer ved å lage en (formell) verden som vi undersøker. To faser:

1. Beskrive den formelle verden.
2. Stille spørsmål om den som maskinen svarer på.

I Prolog snakker vi om:

- Fakta: Basale sannheter ("database").
- Regler: Hvordan splitte et problem i delproblemer.
- Spørsmål: Prolog skal svare ved å bruke reglene og fakta.

Interaktiv kjøring

```
1  > sicstus          --- starter "sicstus prolog"
2  | ?- [filnavn].    --- innlesning fra fil
3
4  yes
5  | ?- <spørsmål>.
6
7  ...
8  | ?- halt.         --- avslutt
```

Tips:

- Fakta og regler bør legges på en fil.
- Suffikset ".pl" er default.
- Mer info på sicstusintro.txt

Prolog

Eksempel: Familie-forhold.

Fakta

Vi lar $person(a, b, c, d)$ angi en person med navn a, med b som mor, c som far og d som fødselsår.

```
1  person(anne, aase, aale, 1960).
2  person(knut, aase, aamund, 1965).
3  person(lars, aase, aale, 1962).
4  person(beate, anne, arne, 1989).
```

Vi har altså:

- Konstanter: ord som starter med liten bokstav, samt tall
- Relasjoner: ord som starter med liten bokstav

Spørsmål

```
1  | ?- person(anne, aase, aale, 1960).
2
3  yes
4  | ?- person(anne, aase, aale, 1962).
5
6  no
```

Spørsmål med variable

Variable: ord som starter med stor bokstav.

Bruk av variable i spørsmål:

- Leter gjennom kunnskapsbasen til det er noe som passer ved innsetting for variablene.
- Som svar returneres det som blir satt inn.

```
1  | ?- person(anne, aase, aale, Aar).
2
3  Aar = 1960 ? ;
4
5  no
6  | ?- person(Barn, aase, aale, Aar).
7
8  Barn = anne,
9  Aar = 1960 ? ;
10
11 Barn = lars,
12 Aar = 1962 ? ;
13
14 no
```

Regler

$barn(X, Y)$ skal bety at X er barn av Y:

```
1  barn(X,Y) :- person(X,Y,Z,U).
2  barn(X,Y) :- person(X,Z,Y,U).
```

Vi kan så stille spørsmål som involverer $barn$-relasjonen:

```
1  | ?- barn(lars,aale).
2
3  yes
4  | ?- barn(lars,Forelder).
5
6  Forelder = aase ? ;
7
8  Forelder = aale ? ;
9
10 no
```

Hvordan kommer systemet frem til disse svarene?

Å finne svar på spørsmål

```
| ?- barn(lars,aale).
```

For relasjonen *barn* har vi to mulige regler å bruke:

1. person(lars,aale,Z,U).
   Passer ikke med noe fakta.
2. person(lars,Z,aale,U).
   Passer med fakta
   person(lars,aase,aale,1962).

```
| ?- barn(lars,Forelder).
```

To muligheter:

1. person(lars,Forelder,Z,U).
   Passer med person(lars,aase,aale,1962), det vil si Forelder = aase.
2. person(lars,Z,Forelder,U).
   Passer med person(lars,aase,aale,1962), det vil si Forelder = aale.

Vi har altså to løsninger her.

Regler med flere betingelser

sosken(X, Y) skal angi at X og Y er søsken:

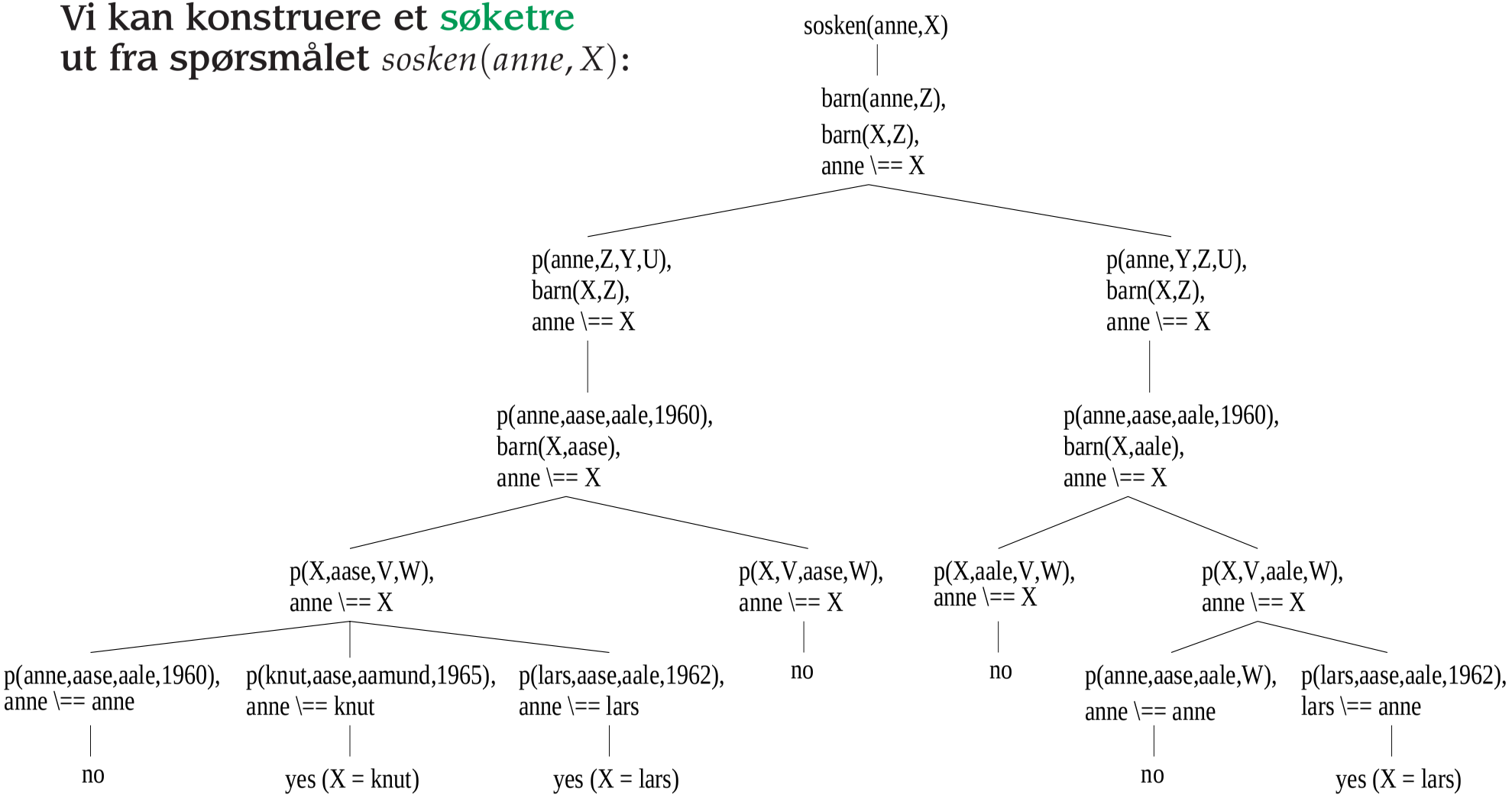
```
1  sosken(X,Y) :- barn(X,Z), barn(Y,Z), X \== Y.
```

- Komma separerer betingelser som alle må være oppfylt.
- Betingelsen X \== Y angir at X og Y må være tekstlig ulike. Dette kravet gjør at *sosken*$(anne, anne)$ gir "no".

```
1  | ?- sosken(anne,X).
2
3  X = knut ? ;
4
5  X = lars ? ;
6
7  X = lars ? ;
8
9  no
```

Hvorfor får vi *lars* til svar to ganger?

Vi kan konstruere et søketre
ut fra spørsmålet $sosken(anne, X)$:



Prolog søker dybde først, venstre mot høyre.

Flere regler

$esosken(X, Y)$ angir at X og Y er "ekte" søsken.

$halvsosken(X, Y)$ angir at X og Y er halvsøsken.

```
1  esosken(X,Y) :- barn(X,Forelder1),
2                  barn(Y,Forelder1),
3                  X \== Y,
4                  barn(X,Forelder2),
5                  barn(Y,Forelder2),
6                  Forelder1 \== Forelder2.
7
8  halvsosken(X,Y) :- barn(X,Forelder),
9                     barn(Y,Forelder),
10                    X \== Y,
11                    barn(X,Forelder1),
12                    barn(Y,Forelder2),
13                    Forelder \== Forelder1,
14                    Forelder \== Forelder2,
15                    Forelder1 \== Forelder2.
```

Avskjæring (cut)

```
1  esosken(X,Y) :- barn(X,Forelder1),
2                  !,
3                  barn(Y,Forelder1),
4                  X \== Y,
5                  barn(X,Forelder2),
6                  barn(Y,Forelder2),
7                  Forelder1 \== Forelder2.
```

Tegnet ! angir "cut": Backtraking over ! er ulovlig.

Dette gir bedre effektivitet, og kan brukes til å unngå repetisjon av samme løsning.

Hva skjer ved $esosken(anne, X)$ og $esosken(X, anne)$ nå?

Rekursive regler

etterkommer(X, Y) skal angi at X er etterkommer til Y:

```
1  etterkommer(X,Y) :- barn(X,Y).
2  etterkommer(X,Y) :- barn(X,Z), etterkommer(Z,Y).
```

Pass på rekkefølgen:

- Ikke-rekursiv regel først!
- Rekursivt delmål til slutt!

```
1  | ?- etterkommer(anne, X).
2
3  X = aase? ;
4
5  X = aale? ;
6
7  no
8  | ?- etterkommer(X, aase).
9
10 X = anne? ;
11
12 X = knut? ;
13
14 X = lars? ;
15
16 X = beate? ;
17
18 no
```

Unifikasjon

Prosessen med å matche (del-)spørsmål med fakta/regler kalles *unifikasjon*.

For at vi skal få en match må vi ha:

- samme relasjon ytterst
- samme antall argumenter
- for hvert argument:
  1. begge er konstater: ok hvis samme konstant
  2. en (ubundet) variabel X og en konstant c: X må bindes til c.
  3. to variable X og Y: Y erstattes med X

Eksempel:

Spørsmål: $etterkommer(anne, Y)$
Fakta: $etterkommer(X, X).$
Unifikasjon: $X := anne$ og $Y := anne$

Ved unifikasjon tilpasses variable, både i arbeidsuttrykk og fakta/regel!

Unifikasjons-operatorer

=	unifiserbart like
==	syntaktisk like
=:=	verdi-like

Eksempler:

1 + 1 = 2	gir no
1 + 1 == 2	gir no
1 + 1 =:= 2	gir yes
X = 1	gir X = 1
X == x	gir no

Negasjon

Operatoren not:

- "yes" blir til "no"
- "no" blir til "yes"

```
1  | ?- not sosken(anne,X).
2
3  no
4  | ?- not sosken(anne,aase).
5
6  yes
```

Lister i Prolog

Eksempler:

- [] - den tomme listen
- [a,b,c] - en liste med tre elementer
- [a|[b,c]] - samme som [a,b,c]

Generelt: [X|Y] er en liste med X som første element og Y som rest-liste.

Funksjoner

Prolog har ikke funksjoner! Funksjonen $f : A \rightarrow B$ kan erstattes med relasjonen $erf(a,b)$ slik at $erf(a,b)$ svarer til $f(a) = b$.

Liste-eksempler:

```
1 lengde([],0).
2 lengde([X|Y],N) :- lengde(Y,M), N is M + 1.
```

I $settsammen(Liste1, Liste2, Liste3)$ fås $Liste3$ ved å sette sammen $Liste1$ og $Liste2$.

```
1 settsammen([],X,X).
2 settsammen([X|Y],Z,[X|U]) :- settsammen(Y,Z,U).
```

Her kan vi tenke på $Liste1$ og $Liste2$ som "inn-parametre", og $Liste3$ som "ut-parameter".

Avsluttende eksempel: tabell

Fakta

$t(I, X)$ angir at verdien X ligger på indeks I i tabellen t:

```
1  t(1,a).
2  t(2,b).
3  t(3,c).
4  t(4,d).
5  t(5,e).
```

Regel

$has(X)$ sier om det finnes en I slik at $t(I, X)$.

```
1  has(X) :- t(I,X).
```

Spørsmål

```
1  | ?- has(c).     % er c med i tabellen?
2
3  yes
4  | ?- t(I,b).     % Hva er indeksen til b?
5
6  I = 2 ;
7
8  no
9  | ?- t(2,X).     % Hva ligger på plass 2?
10
11 X = b ;
12
13 no
```