DD1361 Programmeringsparadigm HT16

Logikprogrammering 1

Dilian Gurov, TCS

Delkursinnehåll

- Logikprogrammering
- Logisk versus procedurell läsning
  - Kontrollflöde
  - Unifiering, Backtracking, Snitt
  - Negation
- Induktiva datatyper och rekursion
  - Inbyggda datatyper: listor
  - Datatyper med PROLOG-termer: träd
- Programmeringtekniker med PROLOG
  - Generera-och-testa

Lärandemål

- Den **deklarativa** aspekten:
  Prolemdomänbeskrivning i termer av relationer
  - Fakta och regler
  - Sundhet och fullständighet av beskrivningen
  - Antagandet om en sluten värld, PROLOG-negation
  - Induktiva datatyper som termer
    - Inbyggda och implicita i PROLOG
    - Strukturell induktion: en princip att garantera well-definedness
- Den **procedurella** aspekten: Kontrollflödet
  - Unifiering
  - Backtrackning
    - Typisk programstruktur: generera-och-testa
  - Snitt

Delkurslitteratur och -material

- Kursbok: **Learn Prolog Now!**
  P. Blackburn, J. Bos, och K. Striegnitz
  Free online version: `www.learnprolognow.org`
- SWi-Prolog manual
- PROLOG-fil: *.pl
- Föreläsningsanteckningar
- Se kurswebbsidorna: Före kursstart, Kursmaterial

Idag

Ämnen

- Deklarativ- och logik-programmering
- Satser: fakta och regler
- Logisk versus procedurell läsning
- Kontrollflöde, Lådmodellen
- Rekursion

Läsmaterial

- Boken: kap. 1-3
- PROLOG-fil: intro.pl
- Föreläsningsanteckningar

Deklarativ- och logik-programmering

Deklarativ programmering

- Programmet är en beskrivning av problemdomänen
- Exekveringar resulterar från frågor (queries)

Logikprogrammering

- En konkret realisering av deklarativ programmering med hjälp av **predikat**
- Domänen beskrivs som en sammansättning av predikatlogiska formler i så-kallad **Horn-klausul** form
- Frågor besvaras med en sök-algoritm som kallas för **resolution**

Från funktioner till predikat

Om f är en funktion så kan vi skriva:

$$\underset{\textbf{(funktion)}}{f(a) = b} \qquad \underset{\textbf{(relation)}}{\text{eller } (a, b) \in f} \qquad \underset{\textbf{(predikat)}}{\text{eller } f(a, b)}$$

Relationer är dock mer generella: inte varje relation är en funktion.

Exempel: "Fadern till a är b" kan deklareras:

- som funktion: $far(a) = b$
- som predikat: $far(a, b)$

Däremot är "a är dotter till b" en relation men ingen funktion!

Satser: fakta

Relationer kan deklareras explicit, varje tupel för sig själv. Sådana definitioner kallas för **fakta**:

```
far(agnes, petter).
far(per, petter).
far(anton, per).

mor(agnes, annika).
mor(per, annika).
mor(kia, agnes).
mor(anton, agnes).
```

Queries

Nu när vi har skapat en relationell databas, vill vi kunna ställa frågor (**queries**) till den.

```
?- far(agnes, per).
?- far(anton, per).
?- far(monika, per).   % Closed world assumption!
```

Notera att PROLOG inte vet vad vi menar med alla symboler: den resonerar helt symboliskt, **utan interpretation**! Så vet PROLOG inte om `far(a, b)` betyder "fadern till `a` är `b`", eller "`a` är fader till `b`", eller t.o.m. att "himlen har färgen `a` och det är `b` stycken moln på den".

Fler queries

Vi kan också ställa frågor som involverar variabler:

```
?- far(agnes, X).  % Vem är far till agnes?
                   % Eller, för vilka X är det sant
                   % att fadern till agnes är X?

?- far(X, petter). % Vems fader är petter?
                   % Flera svar möjliga!
                   % Fås fram med 'n' eller ";"

?- far(X, Y).
```

Satser: regler

Relationer kan också definieras med **regler**:

```
farmor(X, Y) :- far(X, Z), mor(Z, Y).
```

Läs: farmodern till X är Y
om fadern till X är (någon) Z
och modern till Z är Y.

Vi kan observera att:

- ":-" läses som "om", dvs implikation "$\leftarrow$" i omvänd riktning
- "," läses som "och", dvs konjunktion "$\wedge$"
- variabler läses universellt
- Formler (satser) i en sådan form kallas för **Horn-klausuler**

Satser: regler

Disjunktiva regler kan ges som separata regler:

```
foralder(X, Y) :- far(X, Y).
foralder(X, Y) :- mor(X, Y).
```

Läs: en förälder till X är Y (OBS: relation fast inte funktion!)
om fadern till X är Y
eller modern till X är Y.

Följer logiska ekvivalensen:

$$p \vee q \rightarrow r \Leftrightarrow (p \rightarrow r) \wedge (q \rightarrow r)$$

Logisk versus procedurell läsning

Logisk läsning

- som en predikatlogisk formel, deklarativt

Procedurell läsning

- proceduren som PROLOG följer för att hitta ett bevis
- PROLOG läser fakta och regler uppifrån och ned, och klausuler från vänster till höger
- PROLOG försöker instansiera variablerna med termer så att målet blir sant: **unifiering**
- om detta misslyckas med ett faktum eller en regel, går PROLOG till föregående klausul inom regelns kropp om en sådan finns, och annars till nästa faktum eller regel i programmet: **backtracking**

PROLOG-termer

Predikatlogikens syntax har både predikatsymboler såsom funtionssymboler, tagna från olika syntaktiska mängder. I PROLOG däremot finns det tyvärr ingen syntaktisk skillnad mellan dem, och båda representeras som atomer. Det är kontexten som bestämmer hur symbolerna tolkas. Tex i satsen:

```
a(b, c(X)).
```

är `a` en 2-ställig predikatsymbol, `c` en 1-ställig funktionssymbol, `b` en konstant (dvs en 0-ställig funktionssymbol), och `X` en variabel.

Unifiering

Två termer kan **unifieras** om det finns en substitution av variabler med termer för vilken båda termer blir syntaktiskt identiska.

Exempel:

- `X` och `agnes` unifierar med substitution `X = agnes`
- far(X, petter) och `far(agnes, petter)` unifierar med substitution `X = agnes`
- far(agnes, Y) och `far(agnes, petter)` unifierar med substitution `Y = petter`
- far(X, petter) och `far(agnes, Y)` unifierar med substitution `X = agnes, Y = petter`

Kontrollflödet i Prolog: Exempel 1

Fråga: `farmor(anton, X).`

Kontrollflödet i Prolog: Exempel 1

Fråga: `farmor(anton, X).`

- Skapar regelinstans:
  `farmor(X1, Y1) :- far(X1, Z1), mor(Z1, Y1).`

Kontrollflödet i PROLOG: Exempel 1

Fråga: `farmor(anton, X).`

- Skapar regelinstans:
  `farmor(X1, Y1) :- far(X1, Z1), mor(Z1, Y1).`
  unifierar `X1=anton, Y1=X`

Kontrollflödet i PROLOG: Exempel 1

Fråga: `farmor(anton, X).`

- Skapar regelinstans:
  `farmor(X1, Y1) :- far(X1, Z1), mor(Z1, Y1).`
  unifierar `X1=anton, Y1=X`
- `far(anton, Z1).`

Kontrollflödet i PROLOG: Exempel 1

Fråga: `farmor(anton, X).`

- Skapar regelinstans:
  `farmor(X1, Y1) :- far(X1, Z1), mor(Z1, Y1).`
  unifierar `X1=anton, Y1=X`
- `far(anton, Z1).`
  - vid faktum 3 unifierar `Z1=per`

Kontrollflödet i PROLOG: Exempel 1

Fråga: `farmor(anton, X).`

- Skapar regelinstans:
  `farmor(X1, Y1) :- far(X1, Z1), mor(Z1, Y1).`
  unifierar `X1=anton, Y1=X`
- `far(anton, Z1).`
  - vid faktum 3 unifierar `Z1=per`
- `mor(per, X).`

Kontrollflödet i PROLOG: Exempel 1

Fråga: `farmor(anton, X).`

- Skapar regelinstans:
  `farmor(X1, Y1) :- far(X1, Z1), mor(Z1, Y1).`
  unifierar `X1=anton, Y1=X`
- `far(anton, Z1).`
  - vid faktum 3 unifierar `Z1=per`
- `mor(per, X).`
  - vid faktum 5 unifierar `X=annika`

Kontrollflödet i PROLOG: Exempel 1

Fråga: `farmor(anton, X).`

- Skapar regelinstans:
  `farmor(X1, Y1) :- far(X1, Z1), mor(Z1, Y1).`
  unifierar `X1=anton, Y1=X`
- `far(anton, Z1).`
  - vid faktum 3 unifierar `Z1=per`
- `mor(per, X).`
  - vid faktum 5 unifierar `X=annika`

Svar: `X=annika`

Kontrollflödet i Prolog: Exempel 2

Fråga: `foralder(kia, X).`

Kontrollflödet i Prolog: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).`

Kontrollflödet i PROLOG: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`

Kontrollflödet i PROLOG: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`
- `far(kia, X).`

Kontrollflödet i Prolog: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`
- `far(kia, X).`
  - Lyckas inte: backtrackar till nästa regel.

Kontrollflödet i Prolog: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`
- `far(kia, X).`
  - Lyckas inte: backtrackar till nästa regel.
- Skapar regelinstans: `foralder(X2, Y2) :- mor(X2, Y2).`

Kontrollflödet i PROLOG: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`
- `far(kia, X).`
  - Lyckas inte: backtrackar till nästa regel.
- Skapar regelinstans: `foralder(X2, Y2) :- mor(X2, Y2).` unifierar `X2=kia, Y2=X`

Kontrollflödet i PROLOG: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`
- `far(kia, X).`
  - Lyckas inte: backtrackar till nästa regel.
- Skapar regelinstans: `foralder(X2, Y2) :- mor(X2, Y2).` unifierar `X2=kia, Y2=X`
- `mor(kia, X).`

Kontrollflödet i PROLOG: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`
- `far(kia, X).`
  - Lyckas inte: backtrackar till nästa regel.
- Skapar regelinstans: `foralder(X2, Y2) :- mor(X2, Y2).` unifierar `X2=kia, Y2=X`
- `mor(kia, X).`
  - vid faktum 6 unifierar `X=agnes`

Kontrollflödet i PROLOG: Exempel 2

Fråga: `foralder(kia, X).`

- Skapar regelinstans: `foralder(X1, Y1) :- far(X1, Y1).` unifierar `X1=kia, Y1=X`
- `far(kia, X).`
  - Lyckas inte: backtrackar till nästa regel.
- Skapar regelinstans: `foralder(X2, Y2) :- mor(X2, Y2).` unifierar `X2=kia, Y2=X`
- `mor(kia, X).`
  - vid faktum 6 unifierar `X=agnes`

Svar: `X=agnes`

Rekursion

Regler kan vara **rekursiva**, dvs referera till sig själv:

```
forfader(X, Y) :- foralder(X, Y).
forfader(X, Y) :- foralder(X, Z), forfader(Z, Y).
```

Betrakta frågan:

```
?- forfader(kia, X).
```

Vad blir kontrollflödet tills första svaret?
Och till ansdra svaret? (';' leder till backtrackning!)
Vad skulle hända om vi vänder på ordningen på de två reglerna?
Och på de två konjunkterna i andra regeln?

Kontrollflödet i PROLOG: Lådmodellen

För att få ett bättre förståelse av kontrollflödet i PROLOG och trace-funktionen, läs också om **lådmodellen**: se handouts.