

DD1361 Programmeringsparadigm HT15

LOGIKPROGRAMMERING 3

Dilian Gurov, TCS

Induktiva datatyper: Träd (inte inbyggd)

- ▶ Binära träd utan data
- ▶ Binära träd med data

PROLOG-specifika konstruktioner

- ▶ Negation, snitt
- ▶ Aritmetik, I/O
- ▶ Kontrollpredikat, metapredikat

Läsmaterial

- ▶ Boken: Brna, kap. 7, 9
- ▶ PROLOG-fil: tree.pl, misc.pl (se kurswebbsida)
- ▶ Handouts: Föreläsningsanteckningar (se kurswebbsida)

Binära träd utan data

Induktiv definition

Binära träd utan data utgör en oändlig mängd av PROLOG-termer:

- ▶ Ett binärt träd utan data är antingen ett **löv**, `leaf`, eller ett **sammansatt träd**, `branch(t1, t2)`, som består av ett vänster delträd `t1` och ett höger delträd `t2`.
- ▶ BNF-definition:

$$T ::= \text{leaf} \mid \text{branch}(T, T)$$

Därmed matchar varje binärt träd utan data `t` antingen `leaf` eller `branch(TL, TR)`.

Exempel på (stängda) träd-termer: `leaf`, `branch(leaf, leaf)`, `branch(leaf, branch(leaf, leaf))`.

Träd-termer kan också innehålla variabler: `branch(X, leaf)`.

Strukturell induktion

För att definiera ett predikat över binära träd utan data med strukturell induktion:

- ▶ för löv `leaf`, definiera predikatet explicit;
- ▶ för sammansatta trädet `branch(t1, t2)`, definiera predikatet med användning av samma predikat beräknat över delträden `t1` och `t2`.

Då blir predikatet väldefinierad för alla binära träd utan data!

Höjden på ett träd

Höjden på ett träd t är längden (antalet bogar) av längsta stigen från roten till något löv.

Definition med strukturell induktion?

- ▶ höjden på ett löv `leaf` är 0;
- ▶ höjden på ett sammansatta träd `branch(t1, t2)` är maximum av höjderna på delträden $t1$ och $t2$ plus 1.

Höjden på ett träd: height(T, N)

```
max(X, Y, X) :- Y<X.
```

```
max(X, Y, Y).
```

```
height(leaf, 0).
```

```
height(branch(TL, TR), N) :-
```

```
    height(TL, NL),
```

```
    height(TR, NR),
```

```
    max(NL, NR, M),
```

```
    N is M+1.
```

Kompletta träd: complete(T)

Ett träd är komplett om det har alla sina löv på samma höjd.

Definition med strukturell induktion?

```
complete(leaf).  
complete(branch(TL, TR)) :-  
    complete(TL),  
    complete(TR),  
    height(TL, N),  
    height(TR, N).
```

Induktiv definition

- ▶ Ett binärt träd med data är antingen ett löv $\text{leaf}(d)$ med en data-term d , eller ett sammansatt träd $\text{branch}(d, t_1, t_2)$ med en data-term d , ett vänster delträd t_1 , och ett höger delträd t_2 .
- ▶ BNF-definition:
$$T ::= \text{leaf}(D) \mid \text{branch}(D, T, T)$$

Därmed matchar varje binärt träd med data t antingen $\text{leaf}(D)$ eller $\text{branch}(D, TL, TR)$.

Exempel: $\text{branch}(-3, \text{branch}(4, \text{leaf}(8), \text{leaf}(-2)), \text{leaf}(7))$.

Medlemskap i ett träd: lookup(D, T)

Som `member(X, L)`, fast för träd med data.

Definition med strukturell induktion?

```
lookup(D, leaf(D)).
```

```
lookup(D, branch(D, _, _)).
```

```
lookup(D, branch(_, TL, _)) :- lookup(D, TL).
```

```
lookup(D, branch(_, _, TR)) :- lookup(D, TR).
```

Summering i ett träd: `treesum(T)`

Beräknar summan av alla tal i trädets (antar att allt data är tal).

Definition med strukturell induktion?

```
treesum(leaf(N), N).  
treesum(branch(N, TL, TR), N1) :-  
    treesum(TL, NL),  
    treesum(TR, NR),  
    N1 is NL+NR+N.
```

Negation i PROLOG

Kom ihåg den “closed world assumption” som PROLOG utgår ifrån: allt som PROLOG inte lyckas bevisa betraktas som falskt!

Negation i PROLOG betraktas som bevis-teoretisk negation, och inte som logisk negation:

```
snygg(kia).  
osnygg(X) :- \+ snygg(X).
```

Vad blir svaret på:

```
?- osnygg(nisse).
```

Snitt

Snitt (eng: cut) skär bort backtrackingen för predikatet.

Exempel:

```
max(X, Y, X) :- Y<X.  
max(X, Y, Y).
```

Vad blir första och andra svaret på frågan:

```
?- max(5, 3, X).
```

För att eliminera flera än ett svar:

```
maxCut(X, Y, X) :- Y<X, !.  
maxCut(X, Y, Y).
```

Vi skiljer mellan två typer av snitt:

- ▶ **grönt snitt**: ändrar inte programmets logiska läsning, utan bara undviker onödiga sökningar;
- ▶ **rött snitt**: påverkar utdata!

Exempel på grönt snitt: lookup(D, T)

```
lookupCut(D, leaf(D)) :- !.  
lookupCut(D, branch(D, _, _)) :- !.  
lookupCut(D, branch(_, TL, _)) :- lookupCut(D, TL), !.  
lookupCut(D, branch(_, _, TR)) :- lookupCut(D, TR).
```

Exempel på rött snitt

Vad åstadkommer predikatet `isnygg`?

```
snygg(kia).  
isnygg(X) :- snygg(X), !, fail.  
isnygg(X).
```

Vad blir svaret på:

```
?- isnygg(nisse).  
?- isnygg(kia).
```

Vanlig likhet = betyder syntaktiskt likhet, inte "samma värde"!

Speciella predikatet `is` utvärderar andra termen och unifierar med första. Obs: andra termen måste vara tillräcklig instansierad!

Dubbelsidig utvärdering och jämförelse: `==`.

Aritmetiska operatörer: `+`, `-`, `*`, `/`, `//`, `mod`, etc.

Olikheter: `<`, `>`, `=<`, `>=`, etc.

Två nödvändiga predikat:

- ▶ `read(t)`: läs en term fram till nästa punkt och unifiera termen `X` med den;
- ▶ `write(t)`: skriv ut termen `X` till terminalen.

Kontrollpredikat

Vi betraktade redan `fail`, ett predikat som alltid misslyckas.

Predikatet `call(X)` betraktar termen som `X` är unifierad med som ett predikat.

```
snygg(kia).  
not(X) :- call(X), !, fail.  
not(X).
```

Vad blir svaret på:

```
?- not(snygg(nisse)).
```

Två predikat:

- ▶ `assert(t)`: lägga till klausulen `t` till databasen/programmet;
- ▶ `retract(t)`: ta bort klausulen `t` från databasen/programmet.

Båda predikat finns i flera varianter.