DD1361 Programmeringsparadigm HT16

Logikprogrammering 3

Dilian Gurov, TCS

Idag

Induktiva datatyper: Träd (inte inbyggd)

- Binära träd utan data
- Binära träd med data
- Problemdomänbeskrivning

Läsmaterial

- PROLOG-fil: tree.pl
- Handouts: Föreläsningsanteckningar

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: (OBS: PROLOG vet ingenting om detta!)

  ```
  <Tree> ::= leaf | branch(<Tree>, <Tree>)
  ```

  där `leaf` och `branch` kallas för **konstruktörer**.

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 direkt;
- för sammansatta trädet `branch(TL, TR)`, definiera predikatet med användning av samma predikat beräknat över delträden `TL` och `TR`.

Då blir predikatet väldefinierad över **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 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`

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 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(TL, TR)`

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(TL, TR)`
  är maximum av höjdena på delträden `TL` och `TR` 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?

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).
```

Binära träd med data

Induktiv definition

- Ett binärt träd med data är antingen ett löv `leaf(d)` med en data-term `d`, eller ett sammansatt träd `branch(d, t1, t2)` med en data-term `d`, ett vänster delträd `t1`, och ett höger delträd `t2`.
- BNF-definition:

```
<Tree> ::= leaf(<Data>) |
           branch ((<Data>, <Tree>, <Tree>)
```

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

Exempel: `branch(-3,branch(4,leaf(8),leaf(-2)),leaf(7))`.

Medlemskap i ett träd: `lookup(D, T)`

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

Definition med strukturell induktion?

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, N)`

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

Definition med strukturell induktion?

Summering i ett träd: `treesum(T, N)`

Beräknar summan av alla tal i trädet (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.
```

Andra operationer över träd

Operationer

- Att lägga till ett element.
- Att ta bort ett element.
  Hur ska man göra med ett löv?

Problemdomänbeskrivning: Hierarkiska mjukvarusystem

Datatypen **hierarkiska mjukvarusystem** kan definieras induktivt med två regler: ett mjukvarusystem är antingen en *funktion* (som i språket C), eller en *modul* bestående av mjukvarusystem.

1. Föreslå ett sätt att representera mjukvarusystem som PROLOG-termer, där funktionerna representeras som atomer med samma namn som funktionen, t.ex. `main`, `min`, `max`.
2. Ge två exempel på termer som representerar mjukvarusystem med flera nivåer av nästlande.
3. För din representation av datatypen mjukvarusystem, skriv ett predikat `funcsMVS(S, FL)` som är sant om och bara om `FL` är en lista som innehåller namnen på alla funktioner i hierarkiska mjukvarusystemet `S` (med möjlig upprepning).

Hierarkiska mjukvarusystem: PROLOG-termer

1. Sammansättningar går enklast att representeras som listor.

En möjlig BNF-grammatik vore:

```
    <MVS> ::= func(<Name>) | mod(<MVSList>)
<MVSList> ::= [] | [<MVS>|<MVSList>]
```

2. Exempel-termer:

- `func(foo)`
- `mod([func(main),`
  `mod([func(min), func(max), func(avg)])])`

Hierarkiska mjukvarusystem: `funcsMVS(S, FL)`

3. Om vi följer strukturella induktionsprincipen för ömsesidigt rekursiva datatyper (se texten *"The Priciple of Structural Induction"*), så får vi:

```
funcsMVS(func(N), [N]).
funcsMVS(mod(SL), FL) :-
   funcsMVSList(SL, FL).

funcsMVSList([], []).
funcsMVSList([S|SL], FL) :-
   funcsMVS(S, FL1),
   funcsMVSList(SL, FL2),
   append(FL1, FL2, FL).
```