Logisch programmeren 2012
Week 4

Michael Moortgat

1. Incomplete datastructuren

Een nieuw voorbeeld van werken met incomplete datastructuren.

- Peano rekenen met verschilgetallen
- verschillijsten
- binaire woordenboekbomen

2. Binaire bomen

We kunnen binaire bomen representeren met een predicaat `t/3`.

We schrijven `t(Links,Knoop,Rechts)` voor een boom met

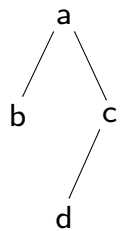
- ▷ `Knoop` als wortel, en
- ▷ `Links` en `Rechts` als linker en rechter dochters, waarbij die dochters zelf binaire bomen zijn.

De lege boom representeren we met de atomaire term `[]`. Een blad van een binaire boom is dus een structuur `t([],Knoop,[])`: een knoop met lege linker en rechter dochter.

3. Voorbeeld

Een boom: `t(t([],b,[]),a,t(t([],d,[]),c,[]))`

Een plaatje:



4. Woordenboekbomen

Een binaire woordenboekboom is een binaire boom met een orderingsrelatie op de knopen.

Een niet-lege boom `t(Links,Knoop,Rechts)` is geordend

- als alle knopen in `Links` kleiner zijn dan `Knoop` en
- alle knopen in `Rechts` groter, en
- als die beide deelbomen zelf ook geordend zijn.

Een binaire boom die aan die voorwaarden voldoet noemen we een (binaire) woordenboekboom.

Vraag Is de binaire boom hierboven een woordenboekboom? Zo niet, maak er dan een woordenboekboom van.

5. Zoeken in een woordenboekboom

Een element opzoeken in een woordenboekboom is efficiënter dan in een lijst, als de woordenboekboom tenminste welgebalanceerd is. Vergelijk `in/2` hieronder met `member/2`.

```
in(X,t(_,X,_)).                 % gevonden!
in(X,t(Links,Knoop,_) )  :-
  Knoop @> X,                   % Knoop groter dan X
  in(X,Links).                  % zoek in de linkerdeelboom
in(X,t(_,Knoop,Rechts) )  :-
  X @> Knoop,                   % X groter dan Knoop
  in(X,Rechts).                 % zoek in de rechterdeelboom
```

6. Incomplete woordenboekbomen

Een incomplete datastructuur voor woordenboekbomen krijg je als je de representatie `[]` voor de lege boom door een variabele vervangt.

De variabele subtermen zijn plaatsen waar een incomplete boom door unificatie kan groeien.

Zo kan je het programma `in/2` hierboven niet alleen gebruiken om te kijken of een element in een woordenboekboom zit, maar ook om een woordenboekboom te bouwen.

Probeer dit uit met de query hieronder. Hoeveel voorkomens van `b` komen er in de woordenboekboom? Waarom?

```
?-in(p,D),in(b,D),in(r,D),in(a,D),in(b,D).
```

7. dic2list/2

We willen nu de geordende lijst van knopen uit een woordenboekboom extraheren. Een naïeve versie zou er zo kunnen uitzien:

```
dic2list([],[]).
dic2list(t(Links,Knoop,Rechts),Lijst):-
    dic2list(Links,L),
    dic2list(Rechts,R),
    append(L,[Knoop|R],Lijst).
```

Laten we kijken hoe dit programma omgeschreven kan worden naar een programma dat van verschillijsten gebruik maakt: `dic2dlist/3`.

8. dic2dlist/3

We schrijven `dic2dlist(Dic,Lijst,Rest)` als het woordenboek `Dic` omgezet kan worden in de verschillijst `Lijst-Rest`. Grensgeval:

```
dic2dlist([],L,L). % L-L is de verschillijst voor de lege lijst.
```

Nu het recursieve geval, eerst met een aanroep van `appendDiff`.

```
dic2dlist(t(Links,Knoop,Rechts),Lijst,Rest):-
    dic2dlist(Links,L,RestL),
    dic2dlist(Rechts,R,RestR),
    appendDiff(L,RestL,[Knoop|R],RestR,Lijst,Rest).

appendDiff(A,B,B,C,A,C).
```

9. appendDiff wegwerken

```
dic2dlist(t(Links,Knoop,Rechts),Lijst,Rest):-
    dic2dlist(Links,L,RestL),
    dic2dlist(Rechts,R,RestR),
    appendDiff(L,RestL,[Knoop|R],RestR,Lijst,Rest).

appendDiff(A,B,B,C,A,C).
```

De aanroep van `appendDiff/3` kan je wegcompileren. Vul de unificerende substituties waardoor `appendDiff` slaagt direct in in de definitie voor `dic2dlist/3`. Het resultaat:

```
dic2dlist(t(Links,Knoop,Rechts),Lijst,Rest):-
    dic2dlist(Links,Lijst,[Knoop|RestL]),
    dic2dlist(Rechts,RestL,Rest).
```