ITI0211- Loogiline programmeerimine

Loeng 3: Prologi andmestruktuurid

J.Vain

14.09.2021

3.1 Listid

- Listid esitavad ***korteeže*** ehk järjestatud elementide multihulki (elemendid võivad korduda)

```
[a, d, f, [s, f, [],d]]
```

- List unifitseerub
  - ühe muutujaga

```
List = [a, d, f, [s, f, [],d]]
```

  - listi erinevaid osi adresseerivate muutujatega, kui on mitte-tühi list

```
[Head|Tail]
```

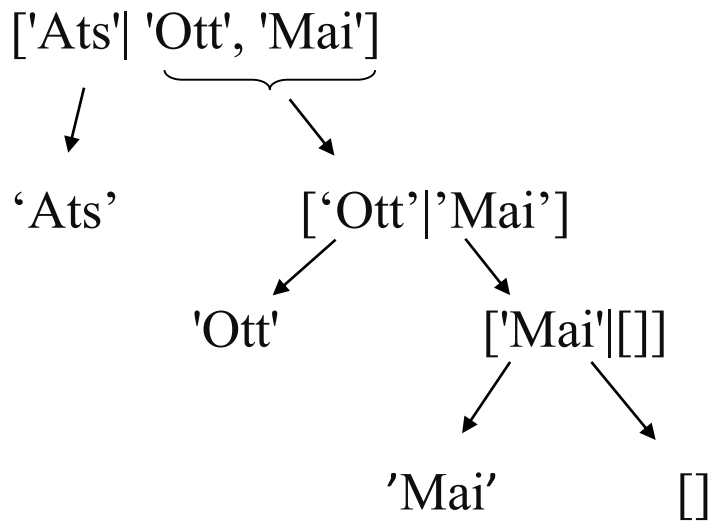
kus

- `Head` - listi pea, mis viitab listi esimestele individuaalsetele elementidele
- `Tail` - listi saba, mis viitab listi kõigi ülejäänud elementide listile
- "`|`" on eraldussümbol pea ja saba vahel.

Kuidas viidata listi elementidele?

- `[ d, f, 4, 5, q]` – otsene viitamine listi elementide väärtustele
- `[ H| T]` – otsene muutujatega viitamine listi peale ja sabale
- `[ _| T]` – kaudne viitamine listi peale ja otsene muutujaga viitamine listi sabale
- `[ H|_]` – otsene muutujaga viitamine listi peale ja kaudne viitamine listi sabale
- `[ El1, El2, El3|Tail]` – otsene muutujatega viitamine listi pea 3-le esimesele elemendile ja sabale

Otsingualgoritmides on list käsitletav kahendpuuna



Otsing kahendpuul on kiire!
$O(\log n)$

Listi tüübikontroll: predikaat `is_list/1`

```
is_list(X) :-                   % Kui argument on väärtustamata muutuja, siis ta ei ole list
      var(X), !,                  % Kas argument on muutuja?
      fail.
is_list([]).                    % kui argument on tühilist
is_list([_|T]) :-               % argument on mittetühi list kui tema saba on ka list
      is_list(T).
```

Märkus:

Konstruktsioon ‘ …`!,fail.`‘ reegli kehas sunnib reegli täitmisel tagurdama ja reeglist väljuma tulemusega `false`

Näiteid listidest ja listipäringutest

```
assert(pere(['Ats', 'Mai', 'Ott'])).
```
% Loome esmalt fakti, mille parameetriks on list

```
?- pere(Liikmed).
```
% Teeme päringu selle fakti poole

```
Liikmed = ['Ats','Mai','Ott']
```
% Prolog tagastab päringu parameetriks oleva
% muutuja väärtuse, milleks on faktis olev list

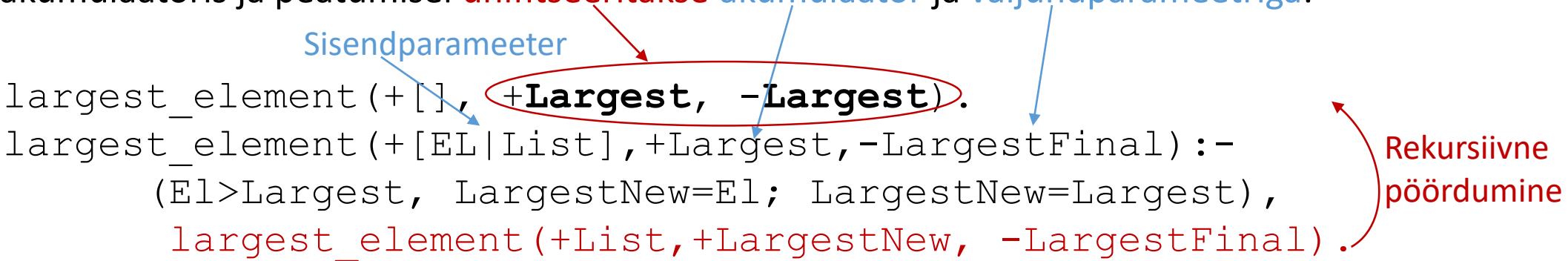
```
?- pere([Pea|Saba]).
Pea = 'Ats',
Saba = ['Mai','Ott']
```

NB! Listi saba on alati list!

```
?- pere([Isa,Ema|Lapsed]).
Isa = 'Ats'
Ema = 'Mai'
Lapsed = ['Ott']
```

Listid ja rekursiivsed reeglid

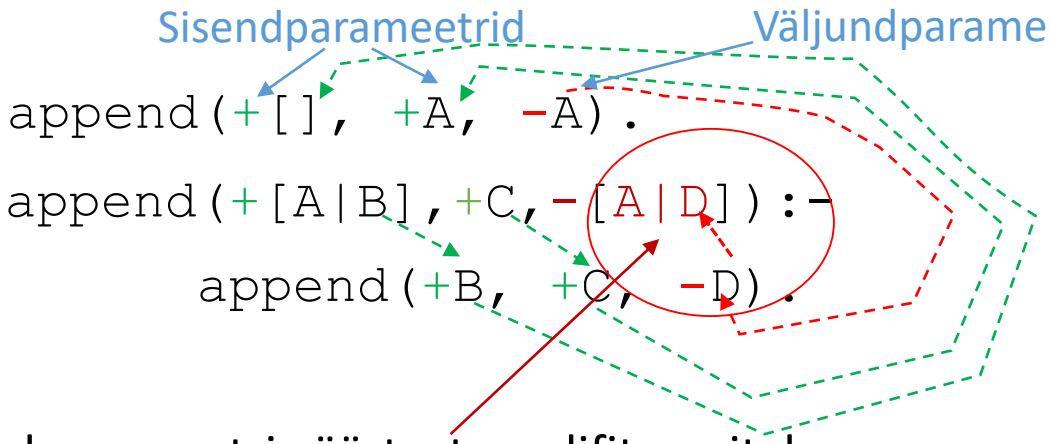
- Listioperatsioonid on realiseeritud enamasti ***rekursiivsete reeglitega***
- ***Pearekursioon*** - tulemus leitakse rekursiivsete päringute ahelas päripidi liikudes, kus vahetulemus säilub akumulaatoris ja peatumisel unifitseeritakse akumulaator ja väljundparameetriga:



NB! Siin väljundparameeter väärtustatakse rekursiooni viimasel sammul, kus kehtib peatumistingimus (vt reegli esimene alternatiiv). Väljundparameetri väärtus säilub tagurdamisel, sest rekursiooni käigus ei ole seda varem väärtustatud.

Listid ja rekursiivsed reeglid

- ***Sabarekursioon*** - tulemus leitakse rekursiivsete päringute ahelas tagurdamisel:
- Näide (listide konkatenatsioon):



Rekursiivne pöördumine

NB! Väljundparameetri väärtust modifitseeritakse reegli alternatiivist tagurdamisega väljudes.

Metasümbolid:
+ – sisendparameeter
– – väljundparameeter
- param. väärtuste ülekandmine pärisuunas
- väärtuste ülekandmine tagurdamisel

Listioperatsioonid: term – list teisendus ”= ..”

Näide 1:

```
?- ?isa(juku,peeter) =.. ?L.
L = [isa, juku, peeter]
```

Näide 2:

```
?- isa(Kes, Kellele) =.. L.
Kes = _G492
Kellele = _G493
L = [isa, _G492, _G493]
```

- Operaatori “=..“ rakendamisel termile on tulemuseks list, mille pea on predikaadi nimi ehk funktor ja saba moodustavad predikaadi argumendid.
- Operaatori “=..“ rakendamisel listile moodustub term, mille funktoriks on listi esimene element ja argumentideks ülejäänud elemendid.

Näide 3: (Kui termiks on list, siis teisendatakse see listiks, mille esimene element on „ .“)

```
?- [peep, ats, ott, mai] =.. L.
L = ['.', peep, [ats, ott, mai]]
```

Listioperatsioonid: konkatenatsioon (sabarekursioon)

% `append(?B, ?C, ?D)` parameetrid võivad olla nii sisendiks kui väljundiks

```
append([], A, A).
append([A|B], C, [A|D]):-
        append(B, C, D).
```

```
?- append (+[s,d,f], +[e,r,t], -X).
 X = [s, d, f, e, r, t]
```

```
?- append(+[s,d,f,g,h],-X,+[s,d,f,g,h,j,k,l]).
X = [j, k, l].
```

```
?- append(-Q,-X,+[s,d,f,g,h,j,k,l]).
Q = [],
X = [s, d, f, g, h, j, k, l] ;
Q = [s],
X = [d, f, g, h, j, k, l] ;
Q = [s, d],
…
```

Listioperatsioonid: elemendi eemaldamine listist (sabarekursioon)

% 1. parameeter on eemaldatav element;

% 2. parameeter on list, millest eemaldatakse elemendi kõik esinemised

% 3. parameeter on tagastatav list

```
remove(+_,+[],-[]).                    % Kui on tühilist
remove(+S,+[S|T],-L):-                 % Kui eemaldatav element on listi peas
        remove(+S,+T,-L),!.
remove(+S,+[U|T],-[U|L]):-             % Kui eemaldatav element on listi sabas, tuleb rekursiivselt
        remove(+S,+T,-L).              % pöörduda ilma listi jooksva välise elemendita

?- remove( ats, [reet, peter, ott, ats], Uus_list).
Uus_list = [reet, peter, ott]
```

Listioperatsioonid: listi pikkuse leidmine (sabarekursioon)

```
?- length(+List, -Length).


length([], 0).
length([S|T], M):-
      length(T, N),
      M is N + 1.
```

Näide:

```
?- length([ e, r, t, w], A).
A = 4
```

Listioperatsioonid: pöördlisti leidmine (sabarekursioon)

```
?- reverse(+List, -R_list).


reverse([],[]).                 % See on saba-rekursiivne reegel
reverse([S|T], L):-
      reverse(T, V),
      append(V, [S], L).  % Konjugeeritakse pööratud listi saba ja jooksev väline element S
```

<u>Näide:</u>

```
?- reverse([1,2,3,4,5],R_list).
R_list = [5,4,3,2,1]
```

Listioperatsioonid: elemendi sisalduvuse kontrollimine listis (pearekursioon)

```
?- member(+Element, -List).
```

```
member(S,[S|T]).
member(S,[V|T]):-
        member(S,T).
```

Näide:

```
?- member([2,3,4],[c,s,[2,3,4],4,s]).
```

true

Listioperatasioonid: listi n-da elemendi leidmine (pearekursioon)

```
?- nth_member(-Element, +N, +List).
```

```
nth_member(S,1,[S|_]).
nth_member(S, N, [_|L]):-
      T is N - 1,
      nth_member(S, T, L).
```

% See on pea-rekursiooni reegel

Näide:

```
?- nth_member(Element,3,[w,5,6,g,h]).
S = 6
```

Listioperatsioonid: permutatsioonid (pearekursioon)

```
?- permutation(+List_1, +List_2).
```

```
permutation(Xs, Ys):-                % Kas list Xs on listi Ys elementide  permutatsioon?
      insort(Xs, Sorted),
      insort(Ys, Sorted).
```

Listioperatsioonid: leksikograafiline sorteerimine (sabarekursioon)

```
?- insort(+List, -SortedList, +Ordering).

insort([], [], _).
insort([X|L], Sorted_list, Ordering):-
        insort(L, SL, Ordering),
        insortx(+X, +SL, -Sorted_list, +Ordering).
```

% `insortx` Asetab X väärtuse sorteeritud listis [A|L] õigele positsioonile

```
insortx(X, [], [X], _):-                          % Kui on tühilist, tagastatakse list elemendiga X
insortx(X, [A|L], [A|M], Ordering):-              % Kui A ≺ X, võrdle listi saba L elementidega
                P =.. [Ordering, A, X], call(P),  %
        insortx(X,L,M,Ordering).
insortx(X, [A|L], [X,A|L], _):-                   % Kui X ≺ A
        A>=X.
```

Listi jooksva elemendi A lisamine tagurdamisel

Sorteerimispredikaadi defineerimine:

`Ordering` := ‘<‘ – aritmeetiline järjestamine;

`Ordering` := ‘aless’ – leksikograafiline järjestamine

Listioperatsioonid: leksikograafiline soreerimine (järg)

```
aless(X,Y):-                                % aatomite leksikograafilise järjestuse kontrollimine
name(X,L), name(Y,M), alessx(L,M).

alessx([],[_|_]).
alessx([X|_], [Y|_]):- X < Y.               % Võrdleme aatomite koodiliste positsioonide kaupa
alessx([H|Q], [H|S]):- alessx(Q,S).
```

- Märkus: `name(?Atom,?List)` teisendab aatomi sümbolite koodide listiks