Loeng 4: Prologi andmestruktuurid

Loengu kava

- Listid ja listioperatsioonid
- Semantilised võrgud
- Freimid
- IF-THEN – ekspertreeglid
- Ekspertreeglid osalise teadmise korral

- **Listid**

Esitavad järjestatud elementide korteeže

```
[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

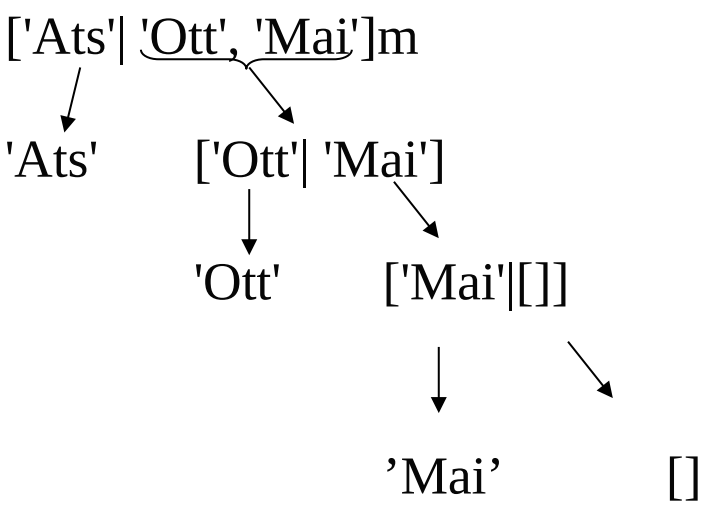
`Head` (listi pea) - listi ilmutatult viidatavad esimesed elemendid

`Tail` (listi saba) – ülejäänud listi elemendid

| - eraldussümbol.

Näiteid: `[H|T], [_|T], [H|_], [El1, El2,El3|Tail]` jne.

Listi esitus kahendpuuna:



Näited:

```
assert(pere(['Ats', 'Mai', 'Ott'])).
?- pere(Liikmed).
Liikmed = ['Ats','Mai','Ott']

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

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

- Listioperatsioonid

1. Termi konverteerimine listiks ja vastupidi ”= ..”.

Tulemuseks on list, mille pea on predikaadi nimi ja saba predikaadi argumendid.

Näide 1:

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

Näide 2:

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

Näide 3:

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

2. Listide konkatenatsioon (*append*):

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

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

3. Elemendi eemaldamine listist:

```
remove(_,[],[]).
remove(S, [S|T], L):-
       remove(S,T,L),!.
remove(S, [U|T], [U|L]):-
       remove(S,T,L).
```

Näide:

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

4. Listi pikkuse määramine

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

Näide:

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

5. Listi pööramine

```
reverse([],[]).
reverse([S|T], L):-
   reverse(T, V),
   append(V, [S], L).
```

Näide:

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

6. Elemendi sisalduvuse kontroll listis:

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

7. Listi n-da elemendi leidmine:

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

• Listide rakendusi: üldistatud sorteerimine (*Clocksin&Mellish insertion sort*, 1987)

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

insortx(X, [A|L], [A|L], Ordering):-
   P =.. [Ordering, A, X], call(P), !.
   insortx(X, L, [X|L], Ordering).
```

Sorteerimispredikaadi defineerimine:

- Ordering := < , kui aritmeetiline järjestamine
- Ordering := aless , kui alfabeetiline järjestamine, kus

```
aless(X,Y):-
   name(X,L), name(Y,M), alessx(L,M).

alessx([],[_|_]).
alessx([X|_], [Y|_]):- X < Y.
alessx([H|Q], [H|S]):- alessx(Q,S).
```

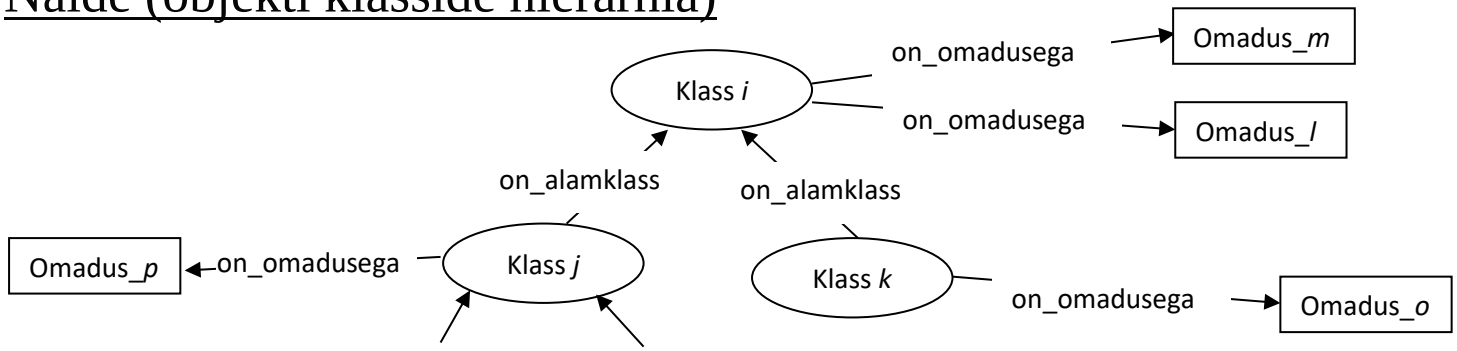
2 Semantilised võrgud

Semantiline võrk (SV) on diagramm, mis esitab objekte, nende omadusi ja objektide vahelisi seoseid.
SV annab mõistete konteksti ja aitab selgitada valdkonna mõistete tähendust.

SV graafiline esitus:

- tipud tähistavad objekte ja nende omadusi
- suunatud kaared näitavad objektide vahelisi seoseid ja objektide seost omadustega

Näide (objekti klasside hierarhia)



Näide:

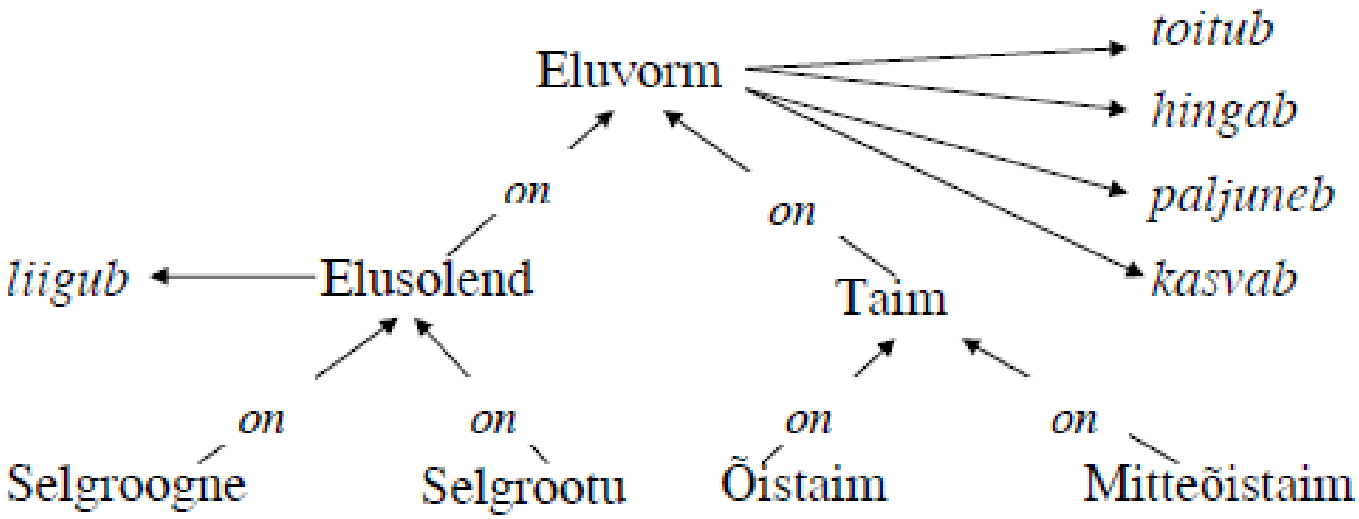
Elusorganismide klassifitseerimine

Klassid:

- elusolend/taim;
- selgroogne/ selgrootu;
- õistaim/mitteõistaim.

Reeglid:

- eluvorm on kas *elusolend* või *taim*;
- eluvorm *toitub*, *hingab*, *paljuneb* ja *kasvab*;
- elusolend on *selgroogne* või *selgrootu*;
- elusolend *liigub*;
- taimed on kas õistaimed või mitteõistaimed.



- Predikaat ”*on*” tähistab seost ”on alamklass”.
- Predikaadid ”*toitub*”, ”*hingab*”, ”*paljuneb*”, ”*kasvab*” on rakendatavad klassidele, millest vastavad nooled lähtuvad ja kõigile nende alamklassidele (omaduste pärimine).

NB! Omadused, mis kehtivad klasside kohta, peavad kehtima ka kõigi tema alam-klasside kohta.

Näide: (on/2 asemel kasutame predikaati is_a/2)

```
% Elusloodus klassifikaator
   is_a(elusolend, eluvorm).
   is_a(taim, eluvorm).
   is_a(selgroogne, elusolend).
   is_a(lehm,selgroogne).
   is_a(selgrootu, elusolend).
   is_a(õistaim, taim).
   is_a(mitteõistaim, taim).
   is_a(kapsas,õistaim).
   is_a(maasi,lehm).
   toitub(eluvorm).
   hingab(eluvorm).
   paljuneb(eluvorm).
   kasvab(eluvorm).
   liigub(elusolend).
   eats(lehm,kapsas).
   eats(selgrootu,taim).
```

Abireeglid:

```
instance_of(X,Y):- is_a(X,Y),!.
instance_of(X,Y):- is_a(X,W), instance_of(W,Y).
```

Reeglid:

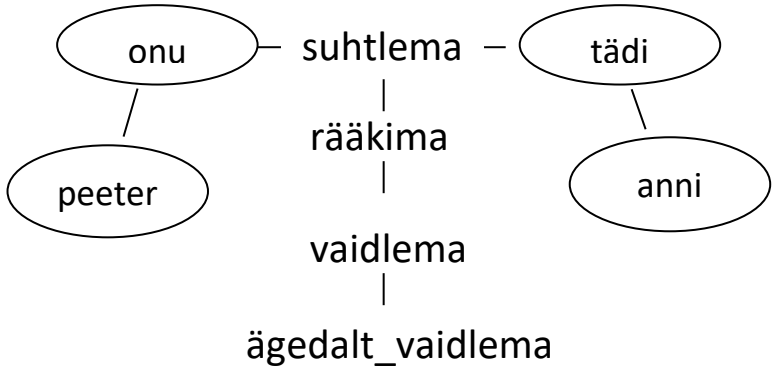
```
hingab(M):- instance_of(M,N), hingab(N).
liigub(P):- instance_of(P,Q), liigub(Q).
toitub(R):- instance_of(R,S), toitub(S).
kasvab(V):- instance_of(V,W), kasvab(W).
paljuneb(X):- instance_of(X,Y), paljuneb(Y).


eats(X,Y):- instance_of(X,V), instance_of(Y,W), eats(V,W).
```

Predikaatmuutujate kasutamine Horni lausetes

(Meta-)reegel:

```
metapredicate(Predicate, Var1, Var2):-
   (Predicate=P ; instance_of(Predicate,P)),
   instance_of(Var1,V1),
   instance_of(Var2,V2),
   Term =..[P,V1,V2],
   call(Term).
```



(Meta-)reegli rakendusnäide:

```
   is_a(rääkima, suhtlema).
   is_a(vaidlema, rääkima).
   is_a(ägedalt_vaidlema, vaidlema).
   is_a(peeter, onu).
   is_a(anni, tädi).
   suhtlema(onu, tädi).

   ?- metapredicate(rääkima, peeter, tädi).
```

3 Freimid

Freime kasutatakse andmestruktuuride üldistusena, kus struktuur on paljudel andmetel ühine, kuid elemendid on erinevat tüüpi või tüüp on täpsustamata.

Freimi iga lahter võib olla omakorda freim, kusjuures kõik lahtrite omadused päritakse tema alamfreimide poolt.

Slot	Value	Type
ALEX	_	(This Frame)
NAME	Alex	(key value)
IS_A	Boy	(parent frame)
SEX	Male	(inheritance value)
AGE	IF-NEEDED: Subtract(current,BIRTHDATE);	(procedural attachment)
HOME	100 Main St.	(instance value)
BIRTHDATE	8/4/2000	(instance value)
FAVORITE_FOOD	Spaghetti	(instance value)
CLIMBS	Trees	(instance value)
BODY_TYPE	Wiry	(instance value)
NUM_LEGS	1	(exception)

Näide1:

```
% on_eriliik(See, Selle)
    on_eriliik(romb, nelinurk).
    on_eriliik(ruut, romb).
% Rombile iseloomulikud lahtri väärtused
    romb(kyljed, vordsed).
    romb(nurgad, ebavordsed).
% Ruudule iseloomulikud lahtri väärtused
    ruut(symmeetriline, 4).

% Pärimisreegel freimi lahtrite väärtustamiseks
    ruut(+Atribuut, -Väärtus):-
        on_eriliik(ruut, Freim),      % Leiab esivanemklassi nime
                                      % millel on viidatud atribuut
        Subgoal =.. [Freim,Atribuut,Vaartus],
        Subgoal.                  % Saab atribuudi päritud väärtuse

?- ruut(kyljed, Missugused).
    Missugused = vordsed
```

Näide 2: Dünaamilised faktid (otsing sügavate pärimispuude korral)

```
on_eriliik_reegel(Esivanem):-
    jooksev(Klass),          % dünaamiline fakt jooksva klassi meeles pidamiseks
    on_eriliik(Klass,Esivanem),
    retract(jooksev(Klass)),
    assertz(jooksev(Esivanem)).

% on_eriliik(See, Selle)
on_eriliik(romb, nelinurk).
on_eriliik(ruut, romb).
on_eriliik(nelinurk,hulknurk).

hulknurk(koosneb,[lõigud,nurgad]).

% Rombile iseloomulikud lahtri väärtused
romb(kyljed, võrdsed).
romb(nurgad, ebavõrdsed).

% Ruudule iseloomulikud lahtri väärtused
    ruut(symmeetriline, 4).
```

% Pärimisreegel objekti omaduste väärtustamiseks

```
ruut(+Atribuut, -Väärtus):-
    assert(jooksev(ruut)),     % dün. fakt jooksev/1 salvestab
                               jooksva klassi nime
    on_eriliik_reegel(Freim), % Leiab esivanemklassi nime
    Subgoal =.. [Freim, Atribuut,Vaartus],
                            % Saab atribuudi päritud väärtuse
    Subgoal,
    retractall(jooksev(_)).

?- ruut(kyljed, Missugused).
Missugused = vordsed
```

4 *”If ... then ...”* reeglid ekspertsüsteemides

Võimaldavad spetsifitseerida põhjus-tagajärg seoseid ning (läbi transitiivsuse) pikemaid põhjuslikkuse ahelaid.

Näide:
Valdkonnas ”Auto diagnostika” kirjeldame auto seisundit faktidega:

```
mootor(ei_käivitu).
...
tuled_on(tuhmid).
...
```

ja järelduste tegemist rikke põhjustest reeglitega:

```
aku(tyhi):-
   mootor(ei_käivitu),
   tuled(tuhmid).
aku(katki):-
   laadimisvool(alla_normi).


?- aku(Seisund).
```

Ekspertreeglid osalise teadmise korral

Olgu otsustuse tingimused võrdse kaaluga:

Näide (teadmusbaas auto seisundist):

```
tuled(tuhmid).
co(üle_normi).
tuuleklaas(mõraga).
mootor(käivitub_raskelt).
```

Reegel

```
aku(tyhi):-
   mootor(ei_käivitu),
   tuled(tuhmid).
```

Päringule

```
?- aku(X).
```

Tagastab Prolog ’false’, sest puudub fakt `mootor(ei_käivitu).`

Täiendame aku/1 reeglit tõepärasuse hinnanguga kujul: $\frac{\textit{Tõeste eeltingimuste arv}}{\textit{Kõigi eeltingimuste arv}}$

```
aku(tyhi, Tõepärasus):-
        reset,
        (mootor(ei_käivitu), incr_valid ;true), incr_all_possible,
        (tuled(tuhmid)),     incr_valid ;true), incr_all_possible,
        valid(Available), all_possible(AllPossible),
        Tõepärasus is Available/AllPossible.

incr_valid:-
    retract(valid(Current)),
    NewCurrent is Current + 1,
    assert(valid(NewCurrent)).

incr_all_possible:-
    retract(all_possible(Current)),
    NewCurrent is Current + 1,
    assert(all_possible(NewCurrent)).

reset:-
    retractall(valid(_)),retractall(all_possible(_)),
    assert(valid(0)), assert(all_possible(0)).

?- aku(Seisund, Tõepärasus).
    Seisund = tyhi
    Tõepärasus = 0.5
    true
```

Harjutus:

Kirjutada Prologis ekspertreeglid:
- ”Kui raadiojaam ei ole korralikult kuuldav või häälestusindikaator vilgub, jätka häälestusnupu keeramist”.
- ”Kui raadiojaam on hästi kuuldav ja häälestusindikaator põleb pidevalt, lõpeta nupu keeramine”.