Loeng 6: Loogiline programmeerimine ja teadmusstruktuurid

Jüri Vain

ITI0211

Sügis 2020

Ontoloogiad

- Tehisintellekti süsteemides toimub ***teadmiste*** kodeerimine, töötlemine, taaskasutus ja kommunikeerimine.
- **Ontoloogia on mõistete ja nendega seotud teadmiste ilmutatud spetsifikatsioon.**
- **Ontoloogia** esitab teadmusvaldkonna struktuuri **kontseptuaalsel tasemel.**
- Kui ontoloogia kirjeldab valdkonda üldiselt, siis **valdkonna seisundit** kirjeldab ontoloogial põhinev **teadmusbaas** (ontoloogia konkretiseering**)**.
- AI agendid omavad teadmusbaasi ja kommunikeerivad ontoloogiate ja ontoloogial interpreteeritavate teadmiste vahetamisega
- Agentide suhtlussõnavara saab põhineda **ühistel** ontoloogiatel.

Ontoloogiate esitusvormid: semantiline võrk

- Semantilisi võrke (SV) kasutatakse ***ontoloogiate esitamiseks***
- SV ehk mõistete võrk on **graafiline esitusvorm**, kus tipud esitavad mõisteid (kontsepte) ja kaared esitavad seoseid nii mõistete endi, kui mõistete ja nende omadust vahel.
- SV loodi eesmärgiga anda **ühine semantiline baas** (*interlingua*) loomulike keelte mõistete tõlkimiseks ühest keelest teise.
- Seega SV määratleb mõistete **konteksti** ja aitab selgitada valdkonna tähendust.

Semantiline võrk

Levinumad seosed (*leksikaalsed suhted*) mõistete vahel:

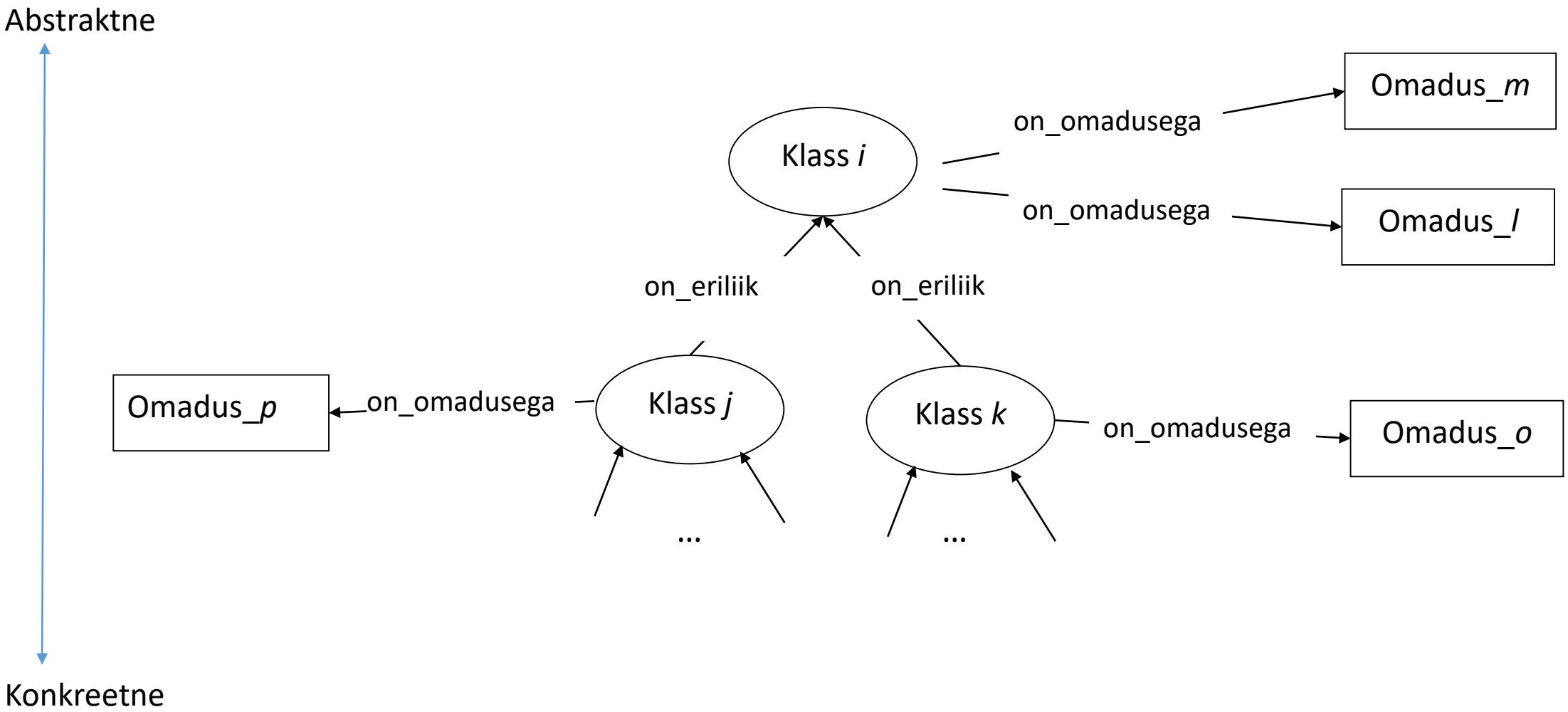
- ***sünonüümia*** – mõiste A väljendab sama mis mõiste B
- ***antonüümia*** – mõiste A väljendab vastupidist mõistele B
- ***meronüümia*** – ***osa-tervik***-suhe ehk ***part_of***-suhe ehk ***koosneb***-suhe

  Mõistete rollid meronüümias:
  - ***holonüüm*** ehk tervik. Näide: nädal
  - ***meronüümid*** ehk osad. Näide: esmaspäev, teisipäev,…, pühapäev
- ***hüponüümia*** – ***üldisem-konkreetsem***-suhe ehk ***is_a*** suhe

  Mõistete rollid hüponüümias:
  - ***hüpernüüm*** ehk üldisem mõiste. Näide: *ülemliik Kala*
  - ***hüponüüm*** ehk alammõiste. Näide: *alamliik Räim*
- ***related_to*** suhe - tähistab ühe mõiste olemist teise mõiste semantilises kontekstis. Näide: inimsuhted – korvpalli meeskond.

Taksonoomia (mõistete klassifikaator)

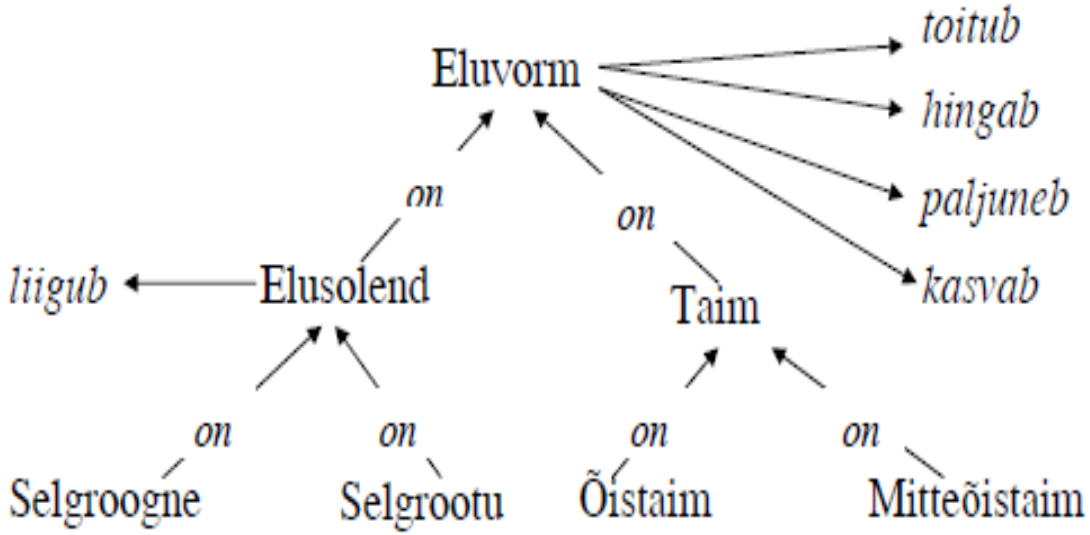
Mõistete hierarhia ja omaduste pärimine taksonoomias.



Näide: Elusorganismide taksonoomia

Klassid:

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



Klass – alamklass seosed:

- eluvorm on kas *elusolend* või *taim*;
- elusolend on *selgroogne* või *selgrootu*;
- taimed on kas õistaimed või mitteõistaimed.

Klasside omadused:

- eluvorm *toitub*, *hingab*, *paljuneb* ja *kasvab*;
- elusolend *liigub*.

Näide: Elusorganismide taksonoomia esitus predikaatloogikas

- Predikaat ”*on*” tähistab seost ”on alamklass”:

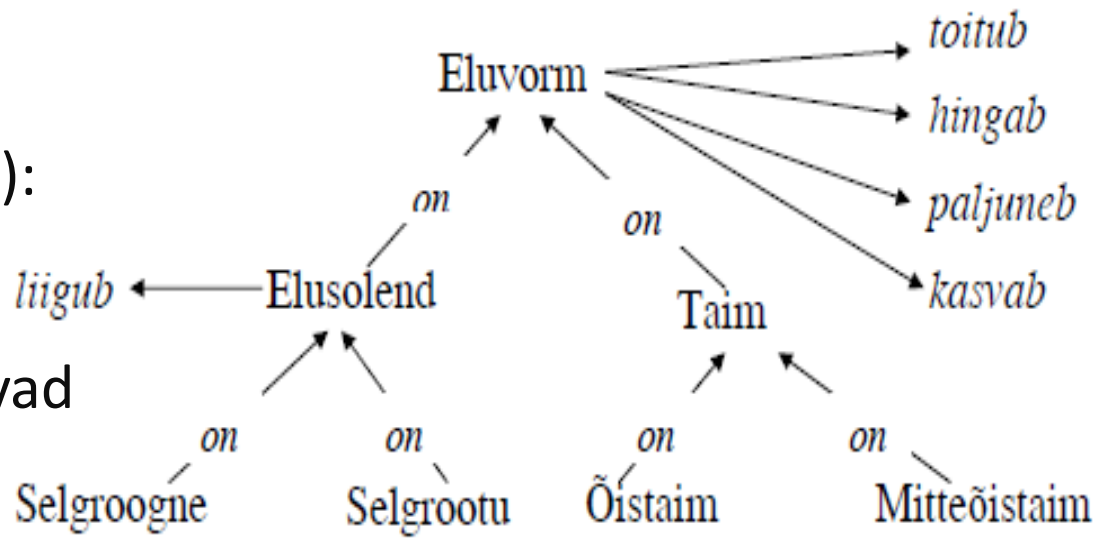
  on (*taim, eluvorm*)

- Predikaadid ”*toitub*”, ”*hingab*”, ”*paljuneb*”, ”*kasvab*” on rakendatavad klassidele, millest vastavad nooled lähtuvad ja kõigile nende alamklassidele s.t. toimub omaduste pärimine):

  toitub(*X*),…, *liigub*(*X*)

- Omadused, mis kehtivad klasside kohta, kehtivad ka kõigi tema alamklasside kohta:

$$\frac{omadus(Y) \wedge on(X,Y)}{omadus(X)}$$



Näide: Elusorganismide klassifikaatori esitus Prologis

```
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).
```

Reeglid omaduste pärimise programmeerimiseks

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

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

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

Meta-reeglid ja predikaatmuutujate kasutamine Horni lausetes

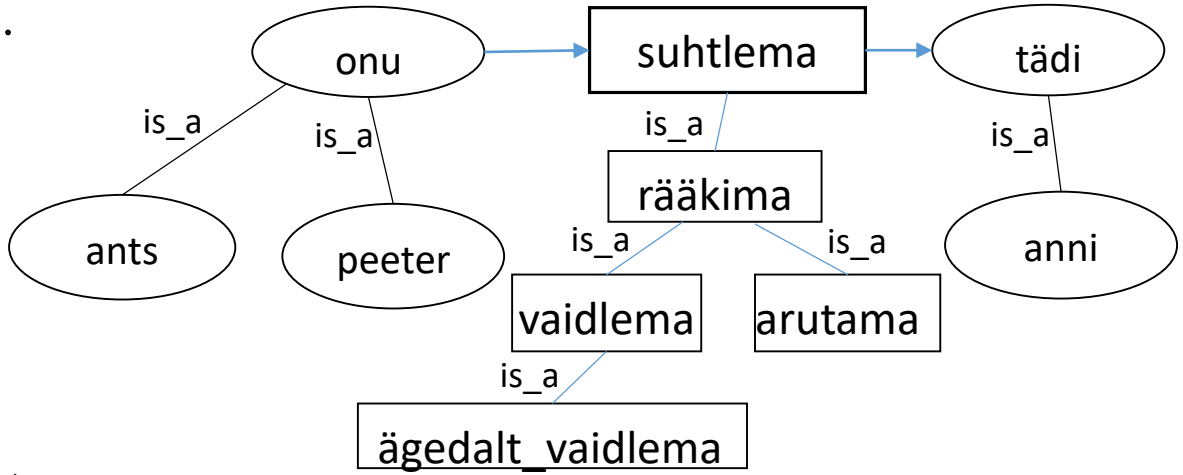
Meta-reegel taksonoomia binaarseoste esitamiseks :

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

Predicate -- predikaatmuutuja

Meta-reegli rakendusnäide: *conceptual graph*

```
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(vaidlema, peeter, anni).
true
```

Freimid

- Freime kasutatakse andmestruktuuride üldistusena, kus struktuur on paljudel andmetel ühine, kuid elemendid on erinevat tüüpi või tüüp on täpsustamata (*uninterpreted*).
- Freimi iga lahter (*slot*) võib olla omakorda freim, kusjuures kõik lahtrite omadused päritakse tema alam-freimide poolt.

Frame Slots	Values	Types
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)

Lahtrid

Lahtrite võimalik sisu

Freimide esitamine Prologis: näide

% on_eriliik(See, Selle)

```
on_eriliik(romb, nelinurk).
on_eriliik(ruut, romb).
```

% Rombile iseloomulikud lahtri (*slot*) 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),                    % Leia esivanemklassi nimi,
    Subgoal =.. [Freim,Atribuut,Vaartus],       % millel on vajalik atribuut
    Subgoal.                                    % Pärib atribuudi väärtuse
```

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

Missugused = vordsed
```

Dünaamiliste faktide kasutamine otsinguks **sügavate** pärimispuude korral

Pärimispuu läbimiseks saab kasutada rekursiooni asemel ka tagurdamisega otsingut (mälu efektiivsem)

```
:- dynamic jooksev/1.
on_eriliik_reegel(Esivanem):-
        jooksev(Klass),                    % dünaamiline fakt jooksva klassi meeles pidamiseks
        on_eriliik(Klass, Esivanem),
        retract(jooksev(Klass)),
        assertz(jooksev(Esivanem)).
```

Pärimisreegli alternatiivne kuju objekti omaduste leidmiseks

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

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

“If … then …” reeglid ekspertsüsteemides

Reeglid esitavad 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).
```

Mittetäieliku teadmusbaasi ekspertreeglid

Olgu otsustustingimused reeglis 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).`

Kuidas tuletada tõepäraseid teadmisi, kui mõni eeldustest on puudu?

- Tuletamine mitte-täieliku teadmusbaasiga
- Püüame reegleid lõdvendada tuues sisse järelduse tõepärasuse hinnangu

$$\frac{\textit{Tõeste eeltingimuste arv või nende kaalutud summa}}{\textit{Kõigi eeltingimuste arv}}$$

- Näide: Täiendame reeglit aku/1 tõepärasuse hinnanguga:

Mittetäieliku teadmusbaasi ekspertreeglid

Igale reeglile lisandub täiendus (punase kirjaga)

```
aku(tyhi, Tõepärasus):-
        reset,
        (mootor(ei_käivitu,T1), incr_valid(T1);true), incr_all_possible,
        (tuled(tuhmid,T2),      incr_valid(T2);true), incr_all_possible,
        valid(Counter), all_possible(AllPossible),
        Tõepärasus is Counter/AllPossible.
```

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

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

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

Abipredikaadid

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

Üldised operatsioonid ontoloogiatega: ***merge*** (kokku sulatamine)

- Sulatamine - uue ontoloogia moodustamine kahest olemasolevast.
- Loomulik sulatamine on võimalik, kui ontoloogiates leiduvad kontseptid nii, et
  - iga kontsept unifitseerub teise ontoloogia mõne kontseptiga (on sünonüümid).
  - ühe ontoloogia kontsept on teise ontoloogia kontsepti täpsustus (is_a relatsioon, võimalik ka polümorfism!)
- Kui loomulik sulatamine ei ole võimalik, siis saab moodustada uue ühise esivanemkontsepti liitkontseptina, mille omadused on mõlema sulatatava kontsepti omaduste ühisosa.
- Et vältida vasturääkivusi liitkontseptides, ei tohi tekkida *omaduste pärimisel konflikti* s.t. pärimisahelas ei tohi esineda vastandlikke omadusi.
- See võib nõuda ontoloogiate laiendamist (***alignment***) või kärpimist.
  - **Laiendamine** --> luuakse uus esivanemtipp ilma konfliktse omaduseta, konfliktsed omadused esinevad ainult loodava tipu mittelõikuvates alampuudes
  - **Kärpimine** --> sulatamisel jäetakse välja konfliktsed kontseptid (tekivad osalised kontekstid - ***views***)

Üldised operatsioonid ontoloogiatega: ***mapping*** *(kujutamine)*

- Lausend moodustatakse mõistetest vastavalt grammatika reeglitele ja lausendi semantika määrab ontoloogia, milles seda interpreteeritakse.
- Lausendi *kujutamine* ühest ontoloogiast teise tähendab selle semantika esitamist algsest erinevas ontoloogias (teises kontekstis).
- Info kadudeta kujutamine on võimalik siis, kui ontoloogiad on loomulikult sulatatavad (esineb kontseptide ühene vastavus)
- Kujutamine teise ontoloogia üldisematesse kontseptidesse on *abstraheerimine* ja *konkreetsematesse* kontseptidesse on täpsustamine (*refinement*).
- Kui see ei ole võimalik, toimub osaline kujutamine, kus vasturääkivust vältimiseks jäetakse osa kontsepte kujutamata.

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”.