Loeng 6: Loogiline programmeerimine ja teadmusstruktuurid

Jüri Vain

ITI0211

Sügis 2021

Ontoloogiad

- Tehisintellekti (AI) ja agentsüsteemides toimub ***teadmiste*** kodeerimine, töötlemine, taaskasutus ja kommunikatsioon ontoloogiate abil.
- **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 lausendite vahetamisega
- Agentide suhtlussõnavara saab põhineda **ühistel** ontoloogiatel.

Ontoloogiate esitusvormid: semantiline võrk

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

Semantiline võrk

Leksikaalsed seosed mõistete vahel:

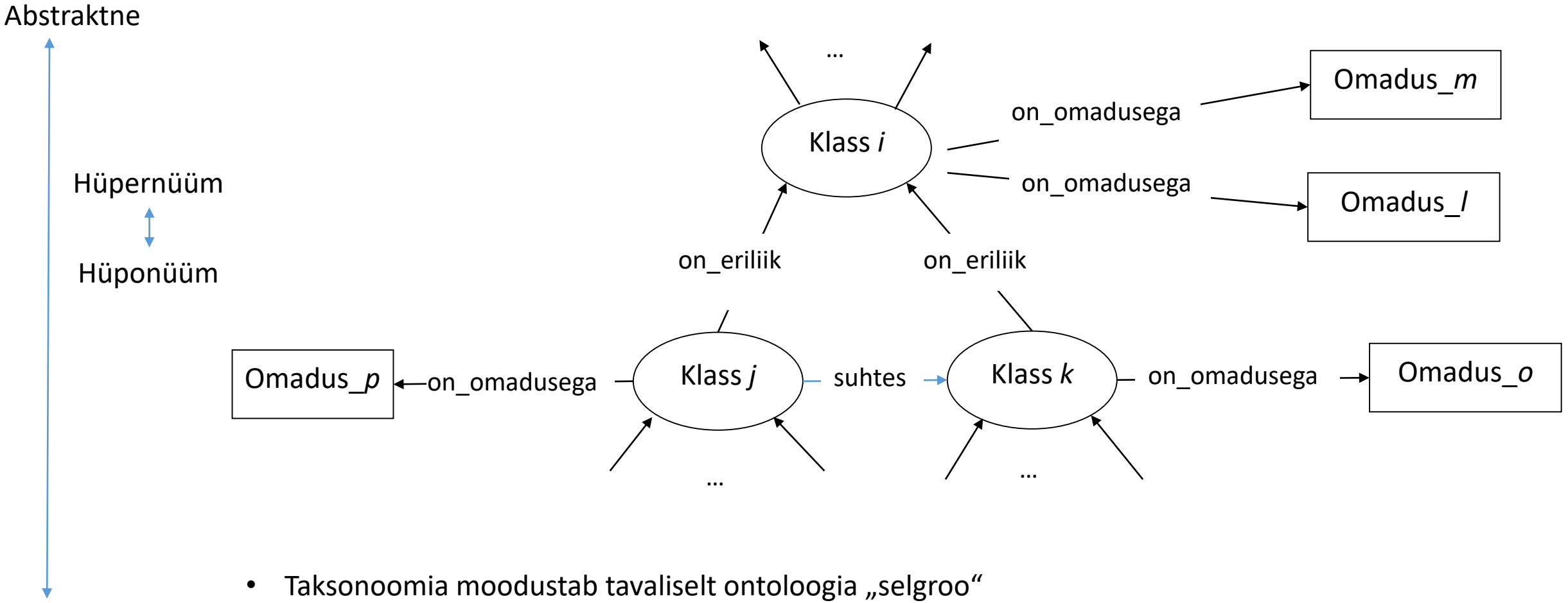
- ***sünonüümia*** – mõiste A väljendab sama mis mõiste B
- ***antonüümia*** – mõiste A väljendab vastupidist mõistele B
- ***meronüümia*** – ***osa-tervik***-seos ehk ***koosneb***-seos (***part_of***)

  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 – meeskonna valimine.

Taksonoomia (mõistete klassifikaator)

Mõistete hierarhia ja omaduste pärimine taksonoomias.

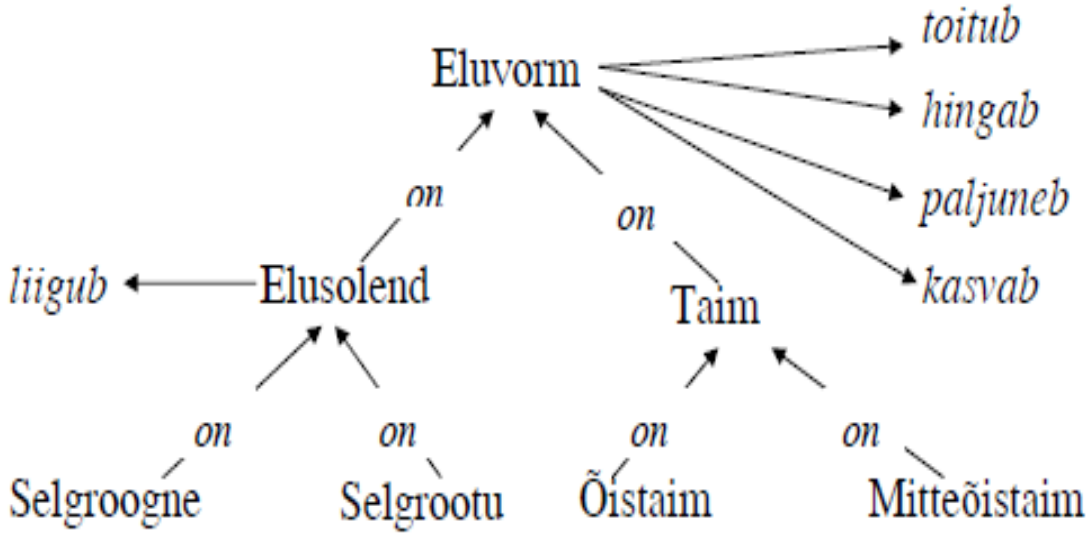


- Taksonoomia moodustab tavaliselt ontoloogia „selgroo“
- Taksonoomias võib olla ka mitmene pärimine e. polümorfism

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*”:

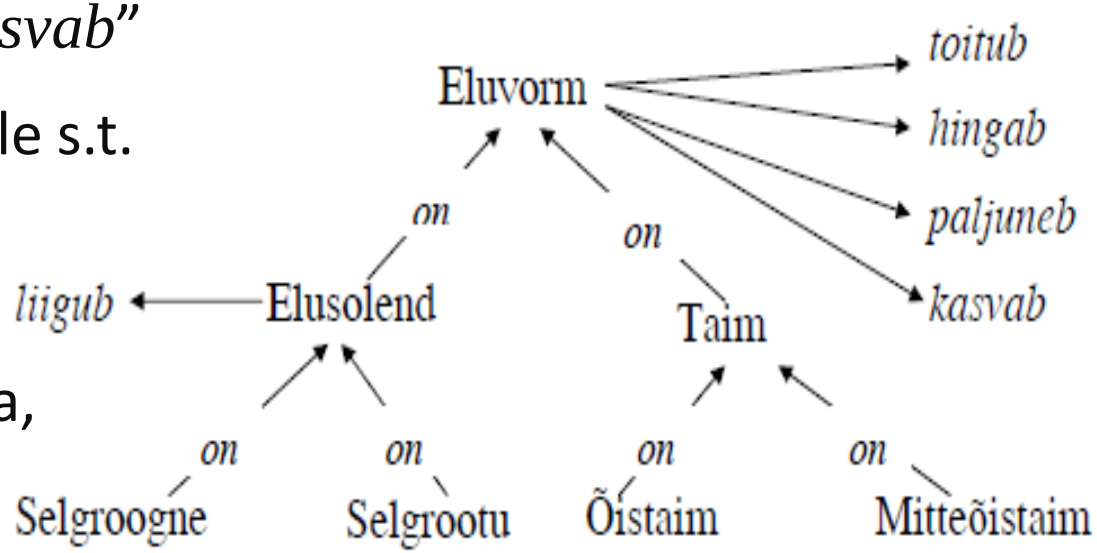
$$on\ (taim, eluvorm) \qquad \% \text{ taim on eluvorm}$$

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

$$toitub(X), \ldots, liigub(X)$$

- Reegel: omadused, mis kehtivad klasside kohta, kehtivad ka kõigi tema alamklasside kohta:

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



Näide: Elusorganismide taksonoomia 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

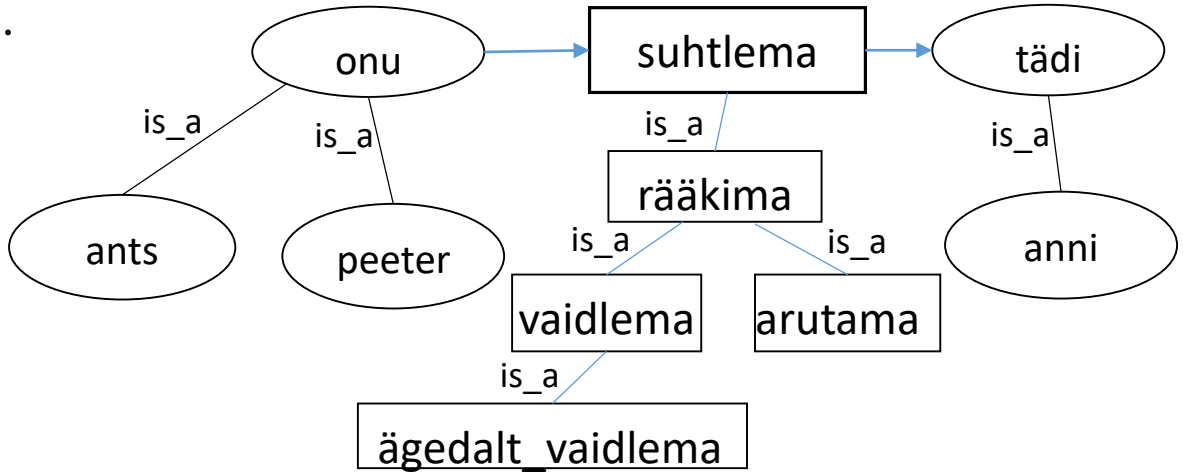
Meta-reegel taksonoomia binaarseoste kasutamiseks tuletuses:

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

Predicate – (kõrgemat järku) predikaatmuutuja

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).
```



Päring: kas peeter ja anni vaidlevad?

```
?- metapredicate(vaidlema, peeter, anni).
true
```

Freimid

- Freime kasutatakse andmestruktuuride üldistusena, kus struktuur on paljudel andmetel ühine, kuid andmed ise (väärtused) on defineerimata, 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: `kujund(atribuut, väärtus)`

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

% Rombile iseloomulikud atribuudid ja nende väärtused
  romb(kyljed, vordsed).
  romb(nurgad, ebavordsed).

% Ruudule iseloomulikud atribuudid ja nende 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:

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

?- diagnoos(aku,Rike).
Rike = tyhi
```

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, näiteks:

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

- Näide: Täiendame reeglit `aku/1` tõepärasuse hinnangu arvutusega:

Mittetäieliku teadmusbaasi ekspertreeglid

Igale reeglile lisandub täiendus (punase kirjaga)

Algne reegel

```
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.
```

Lisareeglid

```
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

Päring

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

Operatsioonid ontoloogiatega: kujutamine (*mapping*)

- Agentide suhtluseks vajalik 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).
- Infokadudeta kujutamine on võimalik siis, kui ontoloogiad on loomulikult ühendatavad (esineb mõistete ühene vastavus)
- Kujutamine teise ontoloogia üldisematesse mõistetesse on ***abstraheerimine*** (*abstraction*) ja ***konkreetsematesse*** mõistetesse on ***täpsustamine*** (*refinement*).
- Kui see ei ole võimalik, toimub osaline kujutamine, kus vasturääkivuste vältimiseks jäetakse osa mõisteid kujutamata.

Operatsioonid ontoloogiatega: **ühendamine** (*merge*)

- Ühendamine - uue ontoloogia moodustamine kahest olemasolevast.
  1. *Loomulik ühendamine* on võimalik, kui ontoloogiates leiduvad mõisted nii, et
     - iga kontsept unifitseerub teise ontoloogia mõne mõistega (on sünonüümid)

     või
     - ühe ontoloogia mõiste on teise ontoloogia mõiste täpsustus (is_a relatsioon, võimalik ka polümorfism!)
  2. Kui loomulik ühendamine ei ole võimalik, siis saab moodustada uue ühise esivanemmõiste, mille omadused on mõlema ühendatava mõiste omaduste ühisosa.
- NB! Et vältida vasturääkivusi liitmõistetes, ei tohi tekkida *omaduste pärimisel konflikti* s.t. pärimisahelas ei tohi esineda vastandlikke omadusi.
- See võib nõuda ontoloogiate laiendamist või kärpimist.

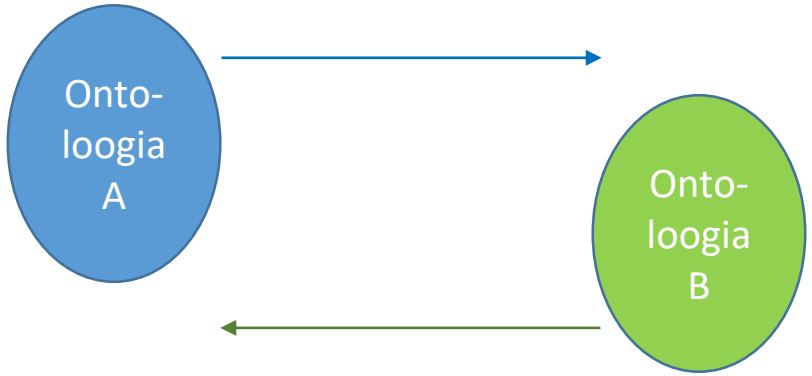
Operatsioonid ontoloogiatega: *laiendamine* ja *kärpimine*

- **Laiendamine** - luuakse uus abstraktsem mõiste ilma alammõistete konfliktse omaduseta, konfliktsed omadused esinevad ainult loodava tipu mittelõikuvates alampuudes

- **Kärpimine** - ühendamisel jäetakse välja konfliktsed mõisted (tekivad osalised ontoloogiad nn *ontoloogia vaated*)

Operatsioonid ontoloogiatega: ***joondamine*** (*alignment*)

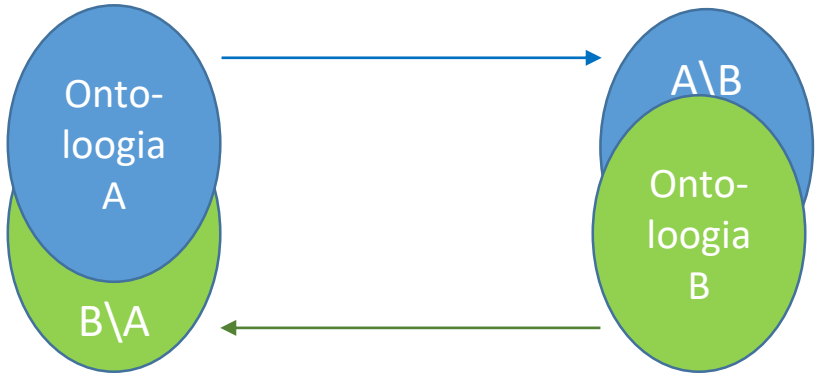
- *Joondamine* on kahe ontoloogia teisendamine üksteise sarnaseks.
- Olgu joondatavad ontoloogiad A ja B



- Et luua vastavus teise ontoloogiaga, saab teisendada lähteontoloogia info kadudeta teiseks ontoloogiaks uute mõistete ja seoste lisamisega st lisame B-le A\B ja A-le lisame B\A.

Operatsioonid ontoloogiatega: ***joondamine*** (*alignment*)

- *Joondamine* on kahe ontoloogia teisendamine üksteise sarnaseks.
- Olgu joondatavad ontoloogiad A ja B



- Et luua vastavus teise ontoloogiaga saab teisendada lähteontoloogia info kadudeta teiseks ontoloogiaks uute mõistete ja seoste lisamisega st lisame B-le A\B ja A-le lisame B\A.
- Joondamise spetsifikatsiooni nimetatakse *artikulatsiooniks*.

Joondamise realisatsioon Prologis

Kasutame süsteemipredikaati **`forall`**(*`:Cond, :Action`*)

```
alignment(A, B):-
      forall((ontology(A,Concept,…),not ontology(B,Concept,…)),
            assert(ontology(B,Concept,…))),
      forall((ontology(B,Concept,…),not ontology(A,Concept,…)),
            assert(ontology(A,Concept,…))).
```

Operatsioonid ontoloogiatega: *unifitseerimine* (*unification*)

- *Unifitseerimine* – joondatakse erinevate ontoloogiate kõik mõisted ja nendevahelised seosed nii, et iga tuletus ühe ontoloogiaga on teisendatav tuletuseks teisel ontoloogial.
- Unifitseerimisel kasutatakse sageli täpsustamist (refinement)

Operatsioonid ontoloogiatega: ***pärimine*** (***inheritance***)

- Kui ontoloogia B on üldisem kui ontoloogia A, siis A pärib kõik ontoloogia B mõisted, seosed ja kitsendused (aksioomid).
- Eeldatakse, et ontoloogia A ei sisalda lisaks midagi, mis põhjustaks vastuolu ontoloogiaga B.
- See tagab ontoloogiate modulaarsuse, kus esivanem ontoloogia kirjeldab üldisemat teadmist ja järeltulija ontoloogia lisab mingi konkreetsema valdkonnaga seotud teadmise.

Lisalugemsit ja Prologi teegid tööks ontoloogiatega

- Conceptual Graph (CG) Libraries in Prolog

  https://github.com/logicmoo/logicmoo_cg
- Access to Sindice semantic web search engine

  https://www.swi-prolog.org/pack/list?p=sindice
- Semantic web library ClioPatria

  https://cliopatria.swi-prolog.org/tutorial/
- Ontology tutorial

  https://www.obitko.com/tutorials/ontologies-semantic-web/ontologies.html
- Murat S¸ensoy et al. Position Paper: Ontological Logic Programming

  http://ceur-ws.org/Vol-668/paper4.pdf

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”.