

## Praktikum PR09

---

### Selgitusi ja näiteid

#### Dünaamiline predikaat,

predikaadid, mida saab luua ja kustutada programmi töö kaigus.

Defineerimine:

`:- dynamic labitud/1.`

`assert/1` – lisab predikaatide dünaamilise andmebaasi lõppu uusi fakte ja reegleid.

`retract/1` - kustutab päringuga unifitseeruvaid predikaate ühekaupa.

`retractall/1` – kustutab dünaamilisest andmebaasist kõik antud päringuga unifitseeruvad predikaadid.

“is” - parema poole arvutamine (`Z is X+Y`)

#### Lõikepredikaat

Lõikepredikaat (!) elimineerib tagurdamisest (*backtracking*) kõik talle vastavas reeglis või päringus eelnevad predikaadid.

`p(X) :- a(X).`

`p(X) :- b(X),c(X),d(X),e(X).`

`p(X) :- f(X).`

`a(1).`

`b(1).`

`c(1).`

`b(2).`

`c(2).`

`d(2).`

`e(2).`

`f(3).`

`p(X) :- a(X).`

`p(X) :- b(X),c(X),!,d(X),e(X).`

`p(X) :- f(X).`

`a(1).`

`b(1).`

`c(1).`

`b(2).`

`c(2).`

`d(2).`

`e(2).`

`f(3).`

`?- p(X).`

`X = 1 ;`

`X = 2 ;`

`X = 3 ;`

`no`

`?- p(X).`

`X = 1 ;`

`no`

#### Rekursioon

% Seost, kes on kelle ema, esitasime faktiga: `mother(Laps, Ema).`

`mother('Mai', 'Kai').`

`mother('Kai', 'Kadri').`

`mother('Kadri', 'Epp').`

`mother('Epp', 'Leida').`

```
esivanem(C, G) :- mother(C, G).
```

```
esivanem(C, G) :-  
    mother(C, Mother),  
    esivanem(Mother, G).
```

```
?- esivanem('Mai', 'Epp').           % Kas Epp on Mai esiema?  
True.
```

```
?- esivanem('Mai', Vanem).           % Kes on Mai esiemad?  
Vanem = kai ;  
Vanem = kadri ;  
Vanem = epp ;  
Vanem = leida ;  
false.
```

Rekursiooni reegils on vaja vähemalt 2-te alternatiivi:

- rekursiooni baas, mis esitab rekursiooni lõpetamistingimust
- ja rekursiooni samm, mis sisaldab rekursiivset pöördumist

vanem/2 reeglit rakendades saame teada, kas või kes on kellegi vanem.

Kui me sooviksime teada ka järgnevust e. kuidas on Leida Mai vana-vana-vanaema, siis saame kasutada järgnevat reeglit:

```
esivanem(X, Y, tee(X,Y)) :- mother(X, Y).  
esivanem(X, Y, tee(X,Path)) :-  
    mother(X, Z),  
    esivanem(Z, Y, Path).
```

```
?- esivanem(mai, leida, Puu).  
Puu = tee(mai, tee(kai, tee(kadri, tee(epp, leida)))) .
```

## Ülesanne

1. Luua teadmusbaas reisiinfo kohta, kus on kirjas, millise transpordivahendiga saab liikuda kahe linna vahel, palju maksab pilet ning reisi saabumise ja väljumise aeg.

Teadmusbaasi loomiseks kasuta predikaate:

- laevaga/3 - laevaga(Kust, Kuhu, Pileti\_hind, Väljumise\_aeg, Saabumise\_aeg).
- bussiga/3 - bussiga(Kust, Kuhu, Pileti\_hind, Väljumise\_aeg, Saabumise\_aeg).
- rongiga/3 - rongiga(Kust, Kuhu, Pileti\_hind, Väljumise\_aeg, Saabumise\_aeg).
- lennukiga/3 - lennukiga(Kust, Kuhu, Pileti\_hind, Väljumise\_aeg, Saabumise\_aeg).

, kus aeg defineeritakse predikaadiga time(Tunnid, Minutid, Sekundid).

Tunnid on täisarv (0..23), Minutid on täisarv (0..59), Sekundid on murdarv (0.0 .. 60.0).

Näiteks:

laevaga(tallinn, helsinki, 120, time(12, 45, 0.0), time(14, 45, 0.0)).

laevaga(tallinn, stockholm, 480, ..., ...).

bussiga(tallinn, riia, 300, ..., ...).

rongiga(riia, berlin, 680, ..., ...).

lennukiga(tallinn, helsinki, 30, ..., ...).

lennukiga(helsinki, paris, 180, ..., ...).

lennukiga(paris, berlin, 120, ..., ...).

2. Lisa teadmusbaasile reegel reisi/4, mis näitab läbitavaid linnu, millise transpordivahendiga antud vahemaa läbitakse ja reisi maksumust alguspunktist lõpppunkti.

Lisa faktibaasi fakte nii, et tekiks võimalus linnade vahel ringiratast sõita.

Näiteks lisa: lennukiga(paris, tallinn, 12, ..., ...).

Lisa otsingureeglile kitsendus, et juba külastatud linna uuesti ei läbita. Kasuta dünaamilist fakti labitud/1 teekonnal läbitud linnade hoidmiseks nt. listis.

```
?- reisi(tallinn, berlin, Tee, Hind).
Tee = mine(tallinn, helsinki, lennukiga, mine(helsinki, paris, lennukiga,
mine(paris, berlin, lennukiga))),
Hind = 330 ;
Tee = mine(tallinn, helsinki, laevaga, mine(helsinki, paris, lennukiga,
mine(paris, berlin, lennukiga))),
Hind = 420 ;
Tee = mine(tallinn, riia, bussiga, mine(riia, berlin, rongiga)),
Hind = 980 ;
false.
```

**NB!!** Näiteks reisi: tallinn → helsinki → paris → tallinn → helsinki → paris → berlin EI TOHI reegel leida.

3. Lisa teadmusbaasile reegl odavaim\_reis/4, mis leiab odavaima reisi kahe punkti vahel, näitab teel läbitavaid linnu, millise transpordivahendiga antud vahemaa läbitakse ja reisi maksumust.

```
?- odavaim_reis(tallinn, paris, Tee, Hind).
Tee = mine(tallinn, helsinki, lennukiga, mine(helsinki, paris, lennukiga)),
Hind = 210 .
```

Näpunäited: kasuta reeglit reisi/4, hetkel parima reisiinfo hoidmiseks kasuta dünaamilisi fakte.

4. Lisa teadmusbaasile reegel lyhim\_reis/4, mis leiab ajaliselt lühima marsruudi kahe punkti vahel,

näitab teel läbitavaid linnu, millise transpordivahendiga antud vahemaa läbitakse ja reisi maksumust.

```
?- lyhim_reis(tallinn, paris, Tee, Hind).  
Tee = mine(tallinn, helsinki, lennukiga, mine(helsinki, paris, lennukiga)),  
Hind = 210 .
```

Kitsendused: Reisi saab jätkata transiitpunktist ainult siis, kui see on võimalik transpordivahendiga, mis väljub mitte varem kui 1 tund peale sellesse punkti saabumist. Reisi kestuse arvutamisel tuleb arvesse võtta ka mitmepäevaseid marsruute.

Näpunäited:

Soovitav on koostada abireeglid teheteks kellaaegadega: (kellaaegade liitmine ja lahutamine).

Näiteks:

```
aegade_vahe(Aeg1, Aeg2, Vahe):-  
    time(H1,M1,S1) = Aeg1,  
    time(H2,M2,S2) = Aeg2,  
    % TODO: siin tee arvutusi  
    .
```