Loogilise programmeerimise keel Prolog

Loengu eesmärgid:

- Senini oli LP õppimisel rõhk
  - Loengutes LP teoorial
  - Praktikumides intuitsioonil ja näidetest arusaamisel
- Käesolevas loengus:
  - Tutvume Prolog keele põhimõistetega (süntaks ja semantika)
  - Tutvume praktiliste programmeerimisvõtetega
  - Teadmiste kinnistamiseks lahendandame programmeerimisülesandeid

- Põhimõisteid

  - Programmi faili loomine ja laadimine mällu käsurealt:

    ?– consult('c:\\...*faili_nimi* ').

    või

    ?- [*faili_nimi*].

- Lausete sisestamine käsurealt:

```
consult(user).
```

lause$_1$.
.....
lause$_n$.

```
end_of_file.
```

- Kommentaar

```
/*    Kommentaar mitmel real
      Kommentaar mitmel real
      Kommentaar mitmel real
*/
```

```
% Kommentaar ühel real
```

- Aatomid -- andmete, programmide, failide jne. nimed:
  - alfanumbrilised aatomid
    - Näide: `seeOn_aatom9`
  - kvoteeritud aatom
    - Näide: `'Aatom'`)
  - Sümbolid
    - Näide: #, $ &. ...
  - Prologi jaoks reserveeritud sümbolid, mida ei ole soovitav kasutada aatomites
    - Näide: ! ; [] ,...

- Termid:
  - muutujad
  - konstandid
    - täisarvud
    - reaalarvud
    - aatomid
    - listid

- Listid -- esitavad loendeid [Head|Tail]

Näide:

```
['Ago', 'Peeter', 'Mai', 'Kadi', 'Rein'],
[],
[12, [34,2],[peep,[]],89]
```

- Predikaadid (faktid)
  - kasutaja poolt defineeritavad predikaadid
  - sisemised e. sisseehitatud predikaadid

predikaadi_nimi(argument$_1$, ..., argument$_n$).

Näited:

```
onupoeg(X, martin).
algarv(3).
teekond([tallinn, risti, haapsalu, kaerdla]).
```

Predikaadi tähistus: `teekond/1`

`teekond` – predikaadi funktor

1 – predikaadi aarsus.

- Horni lause (*clause*)
  Lause esineb fakti või reegli kujul.
  Iga lause algab predikaadi nimega ja lõpeb punktiga.

  Mitu sama funtori ja aarsusega lauset defineerivad Horni lause alternatiivid.

Harjutus:

Esitada lausetega:

- Prolog on programmeerimiskeel
- Ma õpin Prologi
- Nokia saapavabrik varustab soomlasi kummikutega

- Muutujad

  Tähistus:
  - Suure algustähega:

    `A, Inimesed`, ...

  - Allkriipsuga:

    `_loomastik, _c`

- Loogikatehted

  , - konjunktsioon

  ; - disjunktsioon

  `not` - eitus (eitus kehtib ainult Prologi andmebaasi kontekstis so ”suletud maailma” eeldus)

  `b:-  a` või `s:- a -> b` (reegli kehas) – implikatsioon

  `s:-  not (a) ; b.`

  ; (käsurealt) nõuab otsingumootorilt järgmist lahendit

  Näide:

```
tootja(kalev).
tootja(liviko).
tootja(saku).
vahendaja(abestock).
vahendaja(hulgi).
myyja(stockman).
myyja(spar).
myyja(selver).
myyja(liviko).
```

Päringud

```
? - tootja(Kes), myyja(Kes).
? - tootja(a_le_coq); myyja(saku).
? - not (vahendaja(a_le_coq)).
```

- Reeglid

  Reegel ehk *tingimuslik Horni lause*.

  Järeldus :- *Eeldused.*

  Näide: Kui iga päev sajab vihma, siis sajab vihma ka pühapäeval.

```
sajab('Pyhapaev'):-  sajab(iga_paev).

sajab(iga_paev):-
              sajab('Esmaspäev'),
              sajab('Teisipäev),
              sajab('Kolmapäev'),
              sajab('Neljapäev'),
              sajab('Reede'),
              sajab('Laupäev'),
              sajab('Pühapäev').
```

- Päringud (*queries*)

  - Päring defineerib programmi jaoks sihi (*goal*).
  - Päring: `call(Goal)` on semantiliselt samaväärne päringuga `?- Goal`
  - Päringu muutujad väärtustatakse päringu täitmisel, kui leidub sobiv unifitseering

  - ”;” kasutamine päringus sunnib *tagasivõtul* otsima uut lahendit.

  Näide:

```
callex:-
   call(isa(karl, martin)).

?- isa(karl, martin).

?- isa(karl, martin); isa(karl, peeter).
```

- Sisseehitatud predikaadid

Loogikavälised predikaadid:

- otsingu juhtimise predikaadid (`repeat, !, fail,...`)
- sisend-/väljundpredikaadid (`consult, reconsult, get, put, write`, ...)
- aritmeetika predikaadid
- operaatorid

Predikaadid tööks termidega:

- termiteisendused

Näide 1:

```
term_variables(+Term, -List)   % leiab termis esinevad muutujad
          ?- term_variables(a(X, b(Y, X), Z), L).
           L = [G367, G366, G371]
           X = G367
           Y = G366
           Z = G371
```

Predikaadid tööks stringidega:

- `string_to_atom(?String, ?Atom)`
- `string_to_list(?String, ?List)`
- `string_length(+String, -Length)`
- `string_concat(?String1, ?String2, ?String3)`
- `sub_string(+String, ?Start, ?Length, ?After, ?Sub)`

Näide

```
?- sub_string(seebikivikaupmees, 6, 4, After, Sub).

After = 7
Sub = ivik
```

Predikaadid mitme lahendi leidmiseks:

- `findall(+Template, +Goal, -Bag)`
- `bagof(+Template, +Goal, -Bag)`

Näide

Olgu faktid:

```
foo(a, b, c).
foo(a, b, d).
foo(b, c, e).
foo(b, c, f).
foo(c, c, g).
```

Päring

```
?- bagof(C, foo(A, B, C), Cs).
```

annab järgmised alternatiivsed lahendid:

```
A = a, B = b, C = G308, Cs = [c, d] ;
A = b, B = c, C = G308, Cs = [e, f] ;
A = c, B = c, C = G308, Cs = [g] ;
No
```

- Operaatorid
  - Aitavad parandada lähtekoodi loetavust

    `2*3+4*5` *vs* `+(*(2,3),*(4,5)).`
  - Kõik süsteemioperaatorid v.a. "`,`" on ümberdefineeritavad
  - Omavad kehtivust mooduli piires, kuid saab ka moodulitest välja eksportida

Operaatori deklareerimine

- prioriteet (1, ..., 1500) – väiksem number annab kõrgema prioriteedi.
- tüüp:
  - assotsiatiivsus (näide: `16/2 + 6`)
  - kuju (prefiks, infiks, postfiks)

fx	mitte-assotsiatiivne	prefiks
fy	parem-assotsiatiivne	prefiks
xf	mitte-assotsiatiivne	postfiks
yf	vasak-assotsiatiivne	postfiks
xfx	mitte-assotsiatiivne	infiks
xfy	parem-assotsiatiivne	infiks
yfx	vasak-assotsiatiivne	infiks

Näited:

Kui operaatori # tüüp on `yfx`, siis täidetakse # korduvesinemisi vasakult paremale

```
P#Q#R#S = #(#(#(P,Q),R),S)
```

Kui operaatori # tüüp on `xfy`, siis täidetakse # korduvesinemisi paremalt vasakule

```
P#Q#R#S = #(P,#(Q, #(R,S)))
```

Süsteemi operaatorid:

1200	xfx	-->, :-
1200	fx	:-, ?-
1150	fx	dynamic, discontiguous, initialization, mod- ule_transparent, multifile, thread_local, volatile
1100	xfy	;, |
1050	xfy	->, op*->
1000	xfy	,
900	fy	\+
900	fx	~
700	xfx	<, =, =.., =@=, =:=, =<, ==, =\=, >, >=, @<, @=<, @>, @>=, \=, \==, is
600	xfy	:
500	yfx	+, -, /\, \/, xor
500	fx	?
400	yfx	*, /, //, rdiv, <<, >>, mod, rem
200	xfx	**
200	xfy	^
200	fy	+, -, \

Operaatori deklaratsioon:

```
:- op(Priority, Type, Name).
```

Näited:

```
1) :- op(600, fx, [- , +]).
2) :- op(700, xfy, likes).
   juhan    likes    mari.
   mari     likes    peeter.
   juhan    likes    jane.
   jane     likes    juhan.
   kati     likes    X.
   inimene  likes    Y:-          Y=loom;Y=auto.

?- inimene likes auto.
yes

?- inimene likes ratas.
no
```

Võrdus: arg1 = arg2 või =(arg1, arg2)
Võrdus kehtib, kui võrdusega seotud muutujat omavahel unifitseeruvad (väärtused on võrdsed või kui üks muutja on väärtustamata, siis omandab ta teise väärtuse:

Näited (muutujateta võrdus):

```
?- a = a.
    yes

?- a = b.
    no

?- location(apple, kitchen) = location(apple, kitchen).
    yes

?- location(apple, kitchen) = location(pear, kitchen).
    no

?- a(b,c(d,e(f,g))) = a(b,c(d,e(f,g))).
    yes

?- a(b,c(d,e(f,g))) = a(b,c(d,e(g,f))).
    no
```

Näited (muutujatega võrdus):

```
?- X = a.
X = a

?- 4 = Y.
Y = 4

?- location(apple, kitchen) = location(apple, X).
X = kitchen

?- location(X,Y) = location(apple, kitchen).
X = apple
Y = kitchen
```

Näited (väärtustamata muutujatega võrdus):

```
?- X = Y.
   X = _01
   Y = _01

?- location(X, kitchen) = location(Y, kitchen).
   X = _01
   Y = _01

?- X = Y, Y = hello.
   X = hello
   Y = hello

?- X = Y, a(Z) = a(Y), X = hello.
   X = hello
   Y = hello
   Z = hello
```

Keerulisemaid näiteid unifitseerimisest:

```
?- a(b,X) = a(b,c(Y,e)), Y = hello.
   X = c(hello, e)
   Y = hello

?- food(X,Y) = Z, write(Z), nl, X = broccoli, Y =
apple, write(Z).

   food(_01,_02)
   food(broccoli, apple)
   X = broccoli
   Y = apple
   Z = food(broccoli, apple)
```

\= on eelmise eitus.

- Rekursioon
  Predikaadi poole pöördumine toimub sama predikaadi kehast.

  Näide 1:

```
esivanem(Vanem,Noorem):-
      vanem(Vanem,Noorem).
esivanem(Vanem, Noorem):-
      vanem(Vanem, Vahepealne),
      esivanem(Vahepealne,Noorem).
```

Näide 2: (pearekursioon)

```
reverse(A,B):-          % listi pööramine ? -
reverse([2,4,f,g,h,j],Vastus).
   reverse1(A,[],B).

reverse1([],RL,RL).
reverse1([El|L],L1,L2):-
      reverse1(L,[El|L1],L2).
```

Näide 2 (järg): Kõikide alamlistide pööramine

```
reverseM([],RL,RL).
reverseM([El|Suf],L1,L2):-
   sub_list(El,El1),
   reverseM(Suf,[El1|L1],L2).

sub_list (El, El1):-
   list(El),
   reverseM(El,[],El1).
sub_list(El, El).
```

- Programmi dünaamiline muutmine

- Dünaamilise predikaadi defineerimine

```
:- dynamic Nimi1/n1, ..., NimiN/nn.
```

- Predikaatide dünaamiline loomine:

```
assert(Clause).
asserta(Clause).
assertz(Clause).
```

- Predikaatide dünaamiline kustutamine:

```
retract(Clause).
retractall(Clause).
abolish(Nimi/aarsus)
```

- Päringute dünaamiline loomine (predikaat '*Univ*'):

Esituskuju:

```
?Term =.. ?List
```

Listi esimene element on loodava termi funktor ja ülejäänud elemendid on loodava termi argumendid. Argumendiks võib olla ka mutuja.

Näited:

```
?- foo(hello, X) =.. List.
List = [foo, hello, X]

?- Term =.. [baz, foo(1)]
Term = baz(foo(1))
```

Ettevaatust! Dünaamiliste faktide kasutust piirab tagasivõtuga otsing – vältida dün. faktide loomist/kustutamist nende faktide järgi tehtava otsingu ajal.