Teema 2: LOOGILISE PROGRAMMEERIMISE KEEL PROLOG

J.Vain

Loengu kava

- Prolog keele põhimõisted (süntaks ja semantika)
- Tutvume praktiliste programmeerimisvõtetega
- Näited

Praktilisi võtteid tööks programmidega: töötamine käsurealt (1)

- Programmi faili loomine ja laadimine mällu:

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

või

```
? - [faili_nimi].
```

Praktilisi võtteid töös programmidega: töötamine käsurealt (2)

- Lausete sisestamine käsurealt:

```
consult(user).

lause1.
..........
lausen.
end_of_file.
```

Praktilisi võtteid tööks programmidega: kommentaarid

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


% Kommentaar ühel real
```

Põhimõisted: aatom

Aatomid on andmete, programmide, failide jne. nimed:

- Alfanumbrilised aatomid

  Näide:

  `seeOn_aatom9`

- Kvoteeritud aatom

  Näide:

  `'Aatom'`

Põhimõisted: sümbol

- Sümbolid

  Näide:

  `#, $ &. ...`

- Prologis on reserveeritud sümbolid, mida ei ole soovitav kasutada aatomites

  Näide:

  `! ; [] ,...`

Põhimõisted: term

- Muutujad
- Konstandid
  - täisarvud
  - reaalarvud
  - aatomid
  - listid

Põhimõisted: list

- Listid esitavad loendeid
  - Näited:

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

- Listi elementide adresseerimine

```
[Head|Tail]
```

  - Näide:

```
?- [ELEMENT1, ELEMENT2 | TAIL].
   ELEMENT1 = 'Ago'
   ELEMENT2 = 'Peeter'
   TAIL = ['Mai', 'Kadi', 'Rein']
```

Põhimõisted: Horni lause (Horn clause)

- Lause esineb päringu, fakti või reegli kujul.
- Iga lause algab funktoriga (predikaadi nimega) ja lõpeb punktiga.
- Mitu sama funktori ja aarsusega lauset defineerivad Horni lause alternatiivid.

- Harjutus:
  - Esitada Horni lausetega:
    - Prolog on programmeerimiskeel
    - Ma õpin Prologi
    - Nokia varustab soomlasi kummikutega

Põhimõisted: fakt

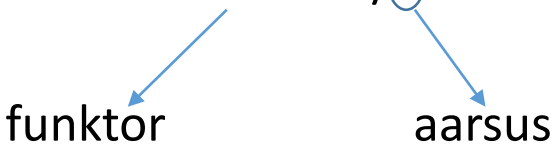
- Fakt esitab tingimusteta kehtivat teadmist (loogikas predikaadi ekstensiooni)
- Faktid jagunevad:
  - kasutaja poolt defineeritavad
  - sisemised e. Prologi sisseehitatud

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

- Näited:

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

Tähistus: teekond/1



Põhimõisted: muutuja

- Tähistus
  - Suure algustähega:
    - Näide

      `A, Inimesed, ...`

  - Allkriipsuga:
    - Näide

      `_loomastik, _c`

  - Interpreteerimata muutuja

    `_` - allkriips ilma nimeta

Põhimõisted: loogikatehe

`,` - konjunktsioon

`;` - disjunktsioon

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

`b:- a.` – reegel esitab implikatsiooni

`s:- a -> b.` - implikatsioon reegli kehas

`s:- not(a) ; b.` - loogiliselt samaväärne implikatsiooniga $a \rightarrow b$

Näide: loogikatehe päringus

Faktibaas:

```
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)).
```

Põhimõisted: reegel

- Reegel ehk *tingimuslik Horni lause*.

```
Järeldus:- Eeldus1,…, EeldusN.
```

- Näide: Kui iga päev sajab vihma, siis sajab vihma ka täna.

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

Põhimõisted: päring

- Päring defineerib missugust lahendit otsitakse (otsingu siht, ingl. *goal*).
- Päring: `call(Goal)` on semantiliselt samaväärne päringuga `?- Goal`
- Päringu muutujad väärtustatakse päringu täitmisel, kui leidub sobiv reegel (unifitseerimine)
- - ”;” kasutamine päringus sunnib *tagasivõtul* otsima uut lahendit.

- Näide:

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

?- isa(karl, martin).

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

Prologi sisesed predikaadid (1)

- 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
```

Prologi sisesed predikaadid (2)

- 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
```

Prologi sisesed predikaadid (3)

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).
```

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

genereerib järgmised 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] ;
   Fail
```

Prologi operaatorid

- Aitavad parandada lähtekoodi loetavust
  - Näiteks

    `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 defineerimine

- 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, module_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 defineerimine

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

Näited:

```
:- op(600, fx, [- , +]).
:- 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.
true
?- inimene likes ratas.
false
```

Võrduspredikaat

`arg1 = arg2` või `=(arg1, arg2)`

Võrdus kehtib, kui võrdusega seotud muutujat omavahel unifitseeruvad st väärtused on võrdsed või kui üks muutja on väärtustamata, siis omandab ta teise väärtuse.

Näited (muutujateta võrdus):

```
?- a = a.
true
?- a = b.
false
?- location(apple, kitchen) = location(apple, kitchen).
true
?- location(apple, kitchen) = location(pear, kitchen).
false
?- a(b,c(d,e(f,g))) = a(b,c(d,e(f,g))).
true
?- a(b,c(d,e(f,g))) = a(b,c(d,e(g,f))).
false
```

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
```

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

```
?- 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)
```

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
   reverse1(A,[],B).

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

? -reverse([2,4,f,g,h,j],Vastus).
```

Dünaamiline programm

- Dünaamilise predikaadi defineerimine

`:- dynamic` *Nimi*1/n_1, ..., *Nimi*N/n_n.

- 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.