1. Loģiskās programmēšanas pamatprincipi

Nodaļas saturs

1. Loģiskās programmēšanas pamatprincipi 1-1
1.1. Programmēšanas paradigmas un Prolog 1-1
1.2. Izcelšanās un darbības pamatprincipi 1-2
1.2.1. Prolog izcelšanās 1-2
1.2.2. Prolog pamatprincipi 1-2
1.3. GNU Prolog interaktīvais interpretators 1-3
1.3.1. GNU Prolog sistēmas palaišana un aizvēršana 1-3
1.3.2. Triviāla Prolog programma 1-3
1.3.3. Programmas ielādēšana sistēmā 1-4
1.3.4. Programmas darbināšana 1-5
1.3.5. Sarakstu apstrāde ar iebūvēto predikātu *append* 1-6
1.4. Valodas Prolog domāšana un konstrukcijas citās sfērās 1-8
1.4.1. Paraugu savietošana un bezkonteksta gramatikas 1-8
1.4.2. Matemātiskā loģika 1-8

Nodaļā ievietotie attēli, programmu piemēri un tabulas

Att. 1-1. GNU Prolog konsoles loga fragments (*Windows*). 1-3
Att. 1-2. Iziešana no GNU Prolog konsoles. 1-3
Att. 1-3. Triviālas Prolog programmas piemērs (*parents.pl*). 1-3
Att. 1-4. Programmas Att. 1-3 grafiskā interpretācija. 1-4
Att. 1-5. Vecāku informācijas ielādēšana interaktīvi – pa vienam teikumam. 1-4
Att. 1-6. Programmas ielādēšana (*parents.pl*). 1-4
Att. 1-7. Prolog programmas ielādēšanas varianti ar dažādu konteksta līmeni. 1-5
Att. 1-8. Vaicājums "Kas ir 'a' bērni". 1-5
Att. 1-9. Rezultāta parādīšanas turpināšana ar semikolu. 1-5
Att. 1-10. Rezultāta parādīšanas turpināšana ar komandu 'a'. 1-5
Att. 1-11. Divu sarakstu konkatenācija. 1-6
Att. 1-12. Konkatenācijas pirmā operanda noskaidrošana. 1-6
Att. 1-13. Konkatenācijas otrā operanda noskaidrošana. 1-6
Att. 1-14. Vaicājums, kas nedod nevienu risinājumu. 1-7
Att. 1-15. Konkatenācijas abu operandu noskaidrošana. 1-7
Att. 1-16. Anonīmais mainīgais _. 1-7
Att. 1-17. Darbināšanas cikla pārtraukšana. 1-8

Tab. 1-1. GNU Prolog darbināšanas cikla kontroles komandas. 1-8
Tab. 1-2. GNU Prolog dažas svarīgākās komandas, predikāti un jēdzieni 1-8

1.1. Programmēšanas paradigmas un Prolog

Prolog ir programmēšanas valoda, kas tika veidota simboliskas (nevis skaitliskas) informācijas apstrādei. Savulaik, blakus valodai Lisp, tā tika uzskatīta par vienu no 2 mākslīgā intelekta programmēšanas valodām, un jebkuram sevi cienošam mākslīgā intelekta speciālistam bija obligāti jāpārzina abas valodas.

Galvenā un acīmredzamā Prolog atšķirība no "pierastajām" programmēšanas valodām (C++, Pascal, Basic) ir programmas pieraksta koncepcija – programma satur nevis izpildāmo instrukciju virknes, bet gan problēmas specifikāciju uzskaitījuma formā, no kuras dators pats nosaka darbību izpildes secību.

Prolog programmēšana nosaka pilnīgi atšķirīgu domāšanas veidu un pieeju programmēšanai, un atsevišķos gadījumos "tradicionālo" programmēšanas valodu zināšana var pat traucēt Prolog apguvi. Pieejas atšķirības ir tik būtiskas, ka Prolog tiek pieskaitīta citai programmēšanas paradigmai.

> **Programmēšanas paradigma** (*programming paradigm*) ir programmēšanas veids, virziens vai pieeja plašākā nozīmē, kas nosaka savu atšķirīgu priekšstatu par to, kas vispār ir programmēšana.

Vairums pazīstamo programmēšanas valodu pieder pie imperatīvās paradigmas. **Imperatīvā** (jeb procedurālā) paradigma reprezentē tradicionālo veidu, kā notiek programmēšanas process. Imperatīvās paradigmas ietvaros programmēšanas process ir komandu virknes pierakstīšana noteiktu operāciju veikšanai. Imperatīvo paradigmu pārstāv mašīnkoda valodas, asambleri, kā arī, piemēram, FORTRAN, COBOL, BASIC, C, Pascal.

Deklaratīvā paradigma, kuru pārstāv arī Prolog, atšķirībā no imperatīvās, nenosaka, kā izpildīt noteiktu uzdevumu, bet gan "kas ir problēma" vai arī "kā problēma varētu tikt reprezentēta". Deklaratīvās programmēšanas paradigmas pārstāvis Prolog no tās realizācijas aspekta tiek saukts arī par **loģisko programmēšanas valodu**.

Citas programmēšanas paradigmas ir funkcionālā, objektorientētā, šablonu.

1.2. Izcelšanās un darbības pamatprincipi

1.2.1. Prolog izcelšanās

Atšķirībā no savulaik konkurējošās valodas Lisp, kas izstrādāta Masačūsetsas tehnoloģiskajā institūtā, Prolog ir "Eiropas produkts". Tā izstrāde sākās 1970. gadā (Alans Kulmeroers un Filips Rusels; *Alan Coulmerauer*, *Philippe Roussel*). Autori vēlējās izveidot valodu, kas uz dotā teksta bāzes spētu veikt **loģiskus** secinājumus. No tā arī veidojās valodas nosaukums: "PROgramming in LOGic". Valoda tika izstrādāta Marseļā (Francijā) 1972. gadā. Lai varētu veidot loģiskus izvedumus, tika izmantots Edinburgas universitātes līdzstrādnieka Kovaļska (*Kowalski*) modelis, un Prolog programmēšanas valoda asociējas ar Edinburgas vārdu.

Prolog pirmā realizācija tapa 1972. gadā, tomēr sakarā ar efektīvu realizāciju trūkumu Prolog izplatība līdz 1980. gadam bija ļoti vāja.

1.2.2. Prolog pamatprincipi

Prolog programma sastāv no divu veidu konstrukcijām – faktiem un noteikumiem, savukārt, lai darbinātu programmu, lietotājam sistēmā ir jāievada vaicājums.

Atbilde uz Prolog sistēmai uzdotu jautājumu ir "jā" vai "nē". Ja atbilde ir "jā", tad papildus tam tiek izdrukātas arī mainīgo vērtības, pie kurām atbilde ir "jā".

Darbošanās ar valodu Prolog prasa rekursīvu domāšanu.

Prolog kā interpretators tradicionālajā variantā ir interaktīva sistēma, ar kuru komunikācija notiek pa atsevišķiem teikumiem, kur katrs teikums beidzas ar punktu (.).

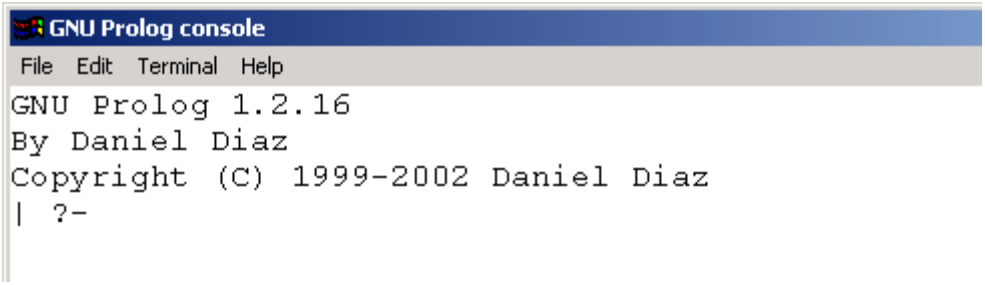
1.3. GNU Prolog interaktīvais interpretators

1.3.1. GNU Prolog sistēmas palaišana un aizvēršana

Mācību materiāli loģiskajā programmēšanā veidoti uz GNU Prolog bāzes. GNU Prolog ir *freeware* programma, kas ir pieejama dažādām platformām, bez tam GNU Prolog ir ļoti tuvs "klasiskajam" prologam un Prolog ISO standartam.

GNU Prolog neatkarīgi no platformas tiek izsaukts ar *gprolog* (Windows sistēmā pieejams GNU Prolog programmu grupā). Pēc programmas izsaukuma parādās interaktīvs konsoles logs:

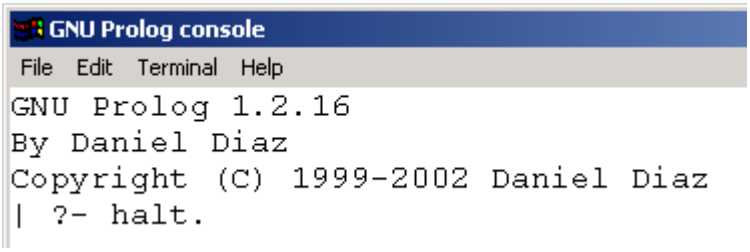
Att. 1-1. GNU Prolog konsoles loga fragments (*Windows*).



Komandu rindā aiz "?-" ir jāraksta komandas, katru komandu noslēdzot ar punktu, komandas nodošanu izsaucot ar ENTER.

Arī, lai aizvērtu konsoles logu jāieraksta speciāla komanda (Windows sistēmā, protams, var arī vienkārši aizvērt konsoles logu). Dažādās Prolog versijās iziešana no programmas ir, izmantojot dažādas komandas, piemēram, *fin* vai *exitsys*, bet GNU Prolog tā ir *halt*:

Att. 1-2. Iziešana no GNU Prolog konsoles.



1.3.2. Triviāla Prolog programma

Klasiskais piemērs Prolog programmas demonstrācijai ir attiecība "vecāks bērns".

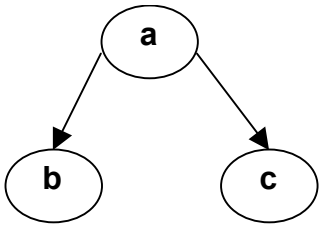
Nākošā Prolog programma demonstrē, ka personai "a" ir divi bērni: "b" un "c":

Att. 1-3. Triviālas Prolog programmas piemērs (*parents.pl*).

```
parent(a,b).
parent(a,c).
```

Šajā programmā ir divi teikumi, kas nosaka divas attiecības "vecāks-bērns":

Att. 1-4. Programmas Att. 1-3 grafiskā interpretācija.



GNU Prolog īpatnība ir tāda, ka kompilatoram daudzos gadījumos "nepatīk" liekie tukšumi programmas tekstā, piemēram, nekorekti būtu likt tukšumu starp *parent* un atverošo iekavu:

```
    parent (a,b).
```

1.3.3. Programmas ielādēšana sistēmā

Šādu programmu var ielādēt sistēmā divos veidos:

- **interaktīvi** pa vienam teikumam, izmantojot predikātu *assertz* katra teikuma ielādēšanai,
- **ielādējot failu** (šeit *parents.pl*), izmantojot predikātu *consult*.

Att. 1-5. Vecāku informācijas ielādēšana interaktīvi – pa vienam teikumam.

```
| ?- assertz(parent(a, b)).

yes
| ?- assertz(parent(a, c)).

yes
| ?-
```

Ievērojiet, ka pēc katras veiksmīgas darbības sistēma pasaka *yes*. Ja komandā tiks ielaista kļūda, sistēma komentēs kļūdu un pateiks *no*.

Šo pašu informāciju var ielādēt arī pa taisno no faila:

Att. 1-6. Programmas ielādēšana (*parents.pl*).

```
| ?- consult(parents).
compiling C:\darbi\acad\courses\elog\src\parents.pl for byte code...
C:\darbi\acad\courses\elog\src\parents.pl compiled, 2 lines read - 403 bytes written, 46 ms

yes
| ?-
```

Iepriekšējā attēlā parādīta programmas ielādēšana vienkāršotā variantā (*consult(parents)*), kas tik vienkārša ir iespējama, pateicoties 3 šādiem apstākļiem: (a) programmas failam "parents.pl" ir GNU Prolog noklusētais paplašinājums ".pl", (b) faila nosaukumā nav speciālo simbolu un (c) faila direktorija ir arī Prolog sistēmas noklusētā direktorija.

Ja kāds no šiem nosacījumiem neizpildās, tad var gadīties, ka šī paša faila (Att. 1-3) ielāde būtu jāveic kādā no sekojošiem variantiem (pieņemot, ka fails atrodas direktorijā "c:\darbi\acad\courses\elog\src\"):

Att. 1-7. Prolog programmas ielādēšanas varianti ar dažādu konteksta līmeni.

```
?- consult('parents').
?- consult('parents.pl').
?- consult('c:\\darbi\\acad\\courses\\elog\\src\\parents.pl').
```

1.3.4. Programmas darbināšana

Programma tiek darbināta ar vaicājumu palīdzību. Vaicājums iedarbina ievades-izpildes-izvades ciklu (*read-execute-write loop*).

Nākošais vaicājums noskaidros, "kas ir personas 'a' bērni":

Att. 1-8. Vaicājums "Kas ir 'a' bērni".

```
?- parent(a,X).

X=b ?
```

Pēc šī vaicājuma vēl netiek pateikts ne *yes*, ne *no* – to vietā ir jautājuma zīme '?', un tiek parādīts tikai pirmais variants: persona b ir personas a bērns (predikātā *parent* pirmais arguments ir vecāks, bet otrais – bērns).

Jautājuma zīme nozīmē, ka darbināšanas cikls, kas saistās ar doto vaicājumu vēl nav beidzies.

Ievērojiet, ka:

- vārds, kas sākas ar mazo burtu (piemēram, a) nozīmē vērtību,
- vārds, kas sākas ar lielo burtu (piemēram, X) nozīmē mainīgo.

Lai turpinātu cikla izpildi un parādītu nākošo rezultātu jāievada semikols:

Att. 1-9. Rezultāta parādīšanas turpināšana ar semikolu.

```
?- parent(a,X).

X=b ? ;

X=c

yes
```

Semikola ';' vietā spiežot 'a' tiek parādīti uzreiz visi atbilžu varianti (nevis tikai viens nākošais variants).

Šajā piemērā gan tam nav nekādas atšķirības, jo kopā ir tikai divi iespējamie atbilžu varianti:

Att. 1-10. Rezultāta parādīšanas turpināšana ar komandu 'a'.

```
?- parent(a,X).

X=b ? a

X=c

yes
```

1.3.5. Sarakstu apstrāde ar iebūvēto predikātu *append*

Saraksts ir galvenā datu struktūra, kas pieejama Prolog. Primitīvajā variantā saraksts ir ar komatiem atdalīts vērtību (t.sk. citu sarakstu) vai mainīgo uzskaitījums, kas ievietots kvadrātiekavās:

```
[a,b,c,d]
```

Tālāk tiks apskatīta iebūvētā predikāta *append* darbība, kas veic divu sarakstu konkatenāciju. Šis piemērs ilustrēs Prolog programmu un vaicājumu deklaratīvo raksturu, kā arī papildus tam – nianses programmas izsaukšanā un darbināšanas cikla kontrolē.

Atcerieties, ka simbolu virkne, kas sākas ar mazo burtu, apzīmē vērtību, bet, kas sākas ar lielo burtu – mainīgo.

Nākošais piemērs parāda, ka, veicot sarakstu [a,b] un [c,d] konkatenāciju, tiek izveidots saraksts [a,b,c,d]:

Att. 1-11. Divu sarakstu konkatenācija.

```
?- append([a,b],[c,d],X).

X = [a,b,c,d]

yes
```

Prolog sistēmai ir iespējams pajautāt arī pretējā virzienā – kādam sarakstam jāpieliek [c,d], lai iegūtu [a,b,c,d]:

Att. 1-12. Konkatenācijas pirmā operanda noskaidrošana.

```
?- append(X,[c,d],[a,b,c,d]).

X = [a,b] ? ;

no
```

Tas, ka aiz rezultāta [a,b] tika parādīta jautājuma zīme '?', nozīmē, k, parādot rezultātu, sistēma vēl nezin, ka cita rezultāta vairāk nebūs.

Var pajautāt arī tā – kas jāpieliek [a,b], lai iegūtu [a,b,c,d]:

Att. 1-13. Konkatenācijas otrā operanda noskaidrošana.

```
?- append([a,b],X,[a,b,c,d]).

X = [c,d]

yes
```

Šoreiz pēc rezultāta parādīšanas sistēma uzreiz jau zina, ka vairāk citu rezultātu nebūs – tā ir tehniska nianse, kas saistīta ar Prolog sistēmas darbību un ar to, kā realizēts konkrētais predikāts (šeit – *append*).

Sistēmai var uzdot jautājumu, uz kuru nav atbildes:

Att. 1-14. Vaicājums, kas nedod nevienu risinājumu.

```
?- append([n,m],X,[a,b,c,d]).

no
```

Predikāta *append* izmantošana nebūt nav izsmelta – izmantojot to pašu programmu, mēs varam pajautāt – kādas virknes konkatenējot, sanāks [a,b,c,d]:

Att. 1-15. Konkatenācijas abu operandu noskaidrošana.

```
?- append(X,Y,[a,b,c,d]).

X = []
Y = [a,b,c,d] ? ;

X = [a]
Y = [b,c,d] ? a

X = [a,b]
Y = [c,d]

X = [a,b,c]
Y = [d]

X = [a,b,c,d]
Y = []

no
```

Var gadīties, ka mani interesē tikai visi saraksti, kurus kaut kam pievienojot, tiek iegūts [a,b,c,d], bet saraksti, kam tik pievienots mani neinteresē. Šim nolūkam noder t.s. anonīmie mainīgie, kurus apzīmē pasvītrojuma zīme ‘_’.

Att. 1-16. Anonīmais mainīgais _.

```
?- append(_,X,[a,b,c,d]).

X = [a,b,c,d] ? a

X = [b,c,d]

X = [c,d]

X = [d]

X = []

no
```

Dažreiz visu iespējamo risinājumu skaits ir tik liels, ka visus tos parādīt nav iespējams vai arī nav vajadzīgs. Pārtraukt darbināšanas ciklu dotajam vaicājumam var, pēc jautājuma zīmes (‘;’ un ‘a’ vietā) spiežot ENTER.

Att. 1-17. Darbināšanas cikla pārtraukšana.

```
?- append(X,Y,[a,b,c,d]).

X = []
Y = [a,b,c,d] ? ;

X = [a]
Y = [b,c,d] ? <ENTER>

yes
```

Tab. 1-1. GNU Prolog darbināšanas cikla kontroles komandas.

`;`	Parādīt nākošo rezultātu
`a`	Parādīt visus nākošos rezultātus
`<ENTER>`	Pārtraukt cikla izpildi

Ja programma ieciklojusies, tad to var pārtraukt, izmantojot komandu *Ctrl + C*.

Tab. 1-2. GNU Prolog dažas svarīgākās komandas, predikāti un jēdzieni.

`halt`	Iziet no sistēmas
`Ctrl + C`	Pārtraukt darbojošas programmas izpildi
`assertz`	pievienot programmai vienu teikumu
`consult`	pievienot programmai faila saturu (ielādēt failu)
`value`	vērtība (sākas ar mazo burtu) (vistuvāk simbolu virknei citās programmēšanas valodās)
`Var`	mainīgais (sākas ar lielo burtu)
`_`	anonīmais mainīgais
`[a,b,c,d]`	saraksts
`append`	divu sarakstu konkatenācija

1.4. Valodas Prolog domāšana un konstrukcijas citās sfērās

1.4.1. Paraugu savietošana un bezkonteksta gramatikas

Viens no svarīgākajiem mehānismiem, kas tiek izmantots Prolog programmas darbībā, ir paraugu savietošana (*pattern matching*). Praktiski tas notiek, savietojot mainīgos vai piešķirot tiem vērtības kādā iepriekš definētā šablonā. Viens no šādas darbības piemēriem ārpus Prolog ir sintaktiskās analīzes koka atpazīšana bezkonteksta gramatikās.

Prolog domāšana ļoti atšķiras no tradicionālās programmēšanas domāšanas, savukārt valodu aprakstīšana, izmantojot bezkonteksta gramatikas prasa domāšanu, kas irt līdzīga Prolog domāšanai. Tādēļ nākošajā nodaļā tiks aprakstīta valodu gramatiku veidošana un šādu valodu analīze, jo vairākas nepieciešamās iemaņas abās sfērās pārklājas.

1.4.2. Matemātiskā loģika

Jau spriežot pēc nosaukuma "loģiskā programmēšana", var nojaust, ka Prologam ar matemātisko loģiku ir daudz kopīga. Tā arī ir, jo Prolog aprēķinu mehānisma pamatā ir

matemātiskā loģika, precīzāk, **predikātu rēķini**. Predikātu rēķinu ietvaros 1965. gadā tika izstrādāts t.s. **rezolūcijas princips**, uz kuru lielā mērā balstās Prolog. Kaut arī dziļas zināšanas matemātiskajā loģikā nav obligāti nepieciešamas, lai programmētu Prolog, tomēr to esamība palīdz izprast veidu, kā Prolog programmas tiek izpildītas. Matemātiskās loģikas elementi tiks apskatīti kādā no nākamajām nodaļām.