2

Operatorii şi aritmetica

2.1. Operatorii în Prolog

Termenii structuraţi în Prolog sunt scrişi, utilizând o notaţie specială: un functor urmat de o listă de argumente între paranteze. La rândul său, argumentele pot fi termeni structuraţi. De exemplu,

cartea(cugetari, autor(blaise, pascale))

Cu toate că această notaţie este, în unele cazuri, bine venită, în alte situaţii, ea poate să nu fie acceptată. De exemplu, în programul de calculare a factorialului, unele predicate ca *+, -, <* etc. s-au scris între cei doi operanzi ai săi, dar nu în formă prefixată. Aceste predicate sunt exemple de operatori.

Limbajul Prolog oferă o mulţime de operatori, unii se găsesc aproape în orice implementare, iar alţii sunt specifici unei versiuni particulare de implementare. Iată o listă de operatori ce pot fi în diverse implementări Prolog: *"+", "-", "*", "/", "<", "=<", ">", ">=", "=", "is", "\+", "->", ":-", "?-",* etc.

Aşa dar, operatorii constituie o facilitate sintactică a limbajului, permiţând o notaţie simplificată a unor termeni structuraţi. Sintaxa exactă depinde de definiţia operatorului. Este important de menţionat, că operatorii ne propun doar o comoditate sintactică pentru scrierea termenilor structuraţi. Realmente în memorie ei se păstrează în formă prefixată. De asemenea menţionăm, că operatorii sunt un mijloc de a construi structuri de obiecte, dar nu fac asupra acestor obiecte careva operaţii.

Limbajul Prolog permite definirea de noi operatori de către programatori prin introducerea în program a unor clauze de formă specială. Descrierea operatorului include: poziţia operatorului, precedenţa (atenţie: aceasta este inversa priorităţii) şi tipul de asociere.

2.1.1. Poziţia operatorilor

Prologul prevede trei categorii de operatori în dependenţă de poziţiile lor: *infix, prefix, postfix*. Operatorii în Prolog pot fi binari şi unari. Operatorii binari sunt folosiţi în poziţia infixă, în timp ce operatorii unari pot avea două poziţii: prefixă şi postfixă.

Un operator infix este utilizat pentru structuri binare şi el apare între cele două argumente. De exemplu, expresiile aritmetice de mai jos utilizează operatori infix:

*3+2 23-2 8*2 30/2 2<7 6>2*

Să menţionăm, că pe aceşti operatori binari sistemul îi transformă în conformitate cu sintaxa sa, respectiv, în:

*+(3,2) -(23,2) *(8,2) /(30,2) <(2,7) >(6,2)*

Iată două exemple de operatori de tip infix folosiţi în entităţile de bază ale Prologului - clauze:

a:-b
rosu(mar), galbena(gutuie)

Aceşti operatori de asemenea sunt binari şi au forma de reprezentare internă:

:-(a,b)
',’(rosu(mar),galbena(gutuie))

Atenţionăm, că aici virgula în functorul *,/2* este protejată de ghilimelele unare.

Operatorul prefix este utilizat pentru o structură unară şi se găseşte în faţa operandului. De exemplu,

+23 -1 rosu mar

care au echivalenţe regulare

+(23) -(1) rosu(mar)

Operatorul postfix formează o structură unară şi apare după unicul său argument. Nu este nici un predicat postfix predefinit, dar aceasta nu înseamnă că nu poate exista. De exemplu,

X este_un_factorial

În forma regulară a sistemului, acest operator are forma

este_un_factorial(X)

2.1.2. Precedenţa

Dacă examinăm structura unei expresii Prolog:

```
copt(mar):-rosu(mar),dulce(mar).
```

ea poate fi reprezentată univoc în formă de arbore. Astfel arborele din Fig.2.1 redă expresia *copt(mar):-(rosu(mar),dulce(mar))*, iar cel din Fig.2.2 reprezintă expresia *(copt(mar):-rosu(mar)), dulce(mar)*.

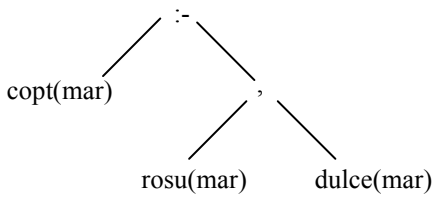

Fig. 2.1. Arborele pentru *copt(mar):-(rosu(mar),dulce(mar))*

Bineînţeles, că prima este varianta acceptată.

Pentru a construi arborele expresiei, mai întâi se găseşte operatorul cu precedenţa cea mai înaltă (care va constitui rădăcina) şi dacă el este infix , expresia se divizează în două: partea stângă şi partea dreaptă. Procesul continuă pentru fiecare din subarbori.

Pentru a descrie astfel de situaţii, fiecărui operator i se asociază o precedenţă (sau prioritate), reprezentată de un număr întreg cuprins între *1* şi *1200* ca, de exemplu, în Prolog/Dec-10 sau CProlog. În orice caz, numerele mai mari corespund precedenţelor mai înalte.

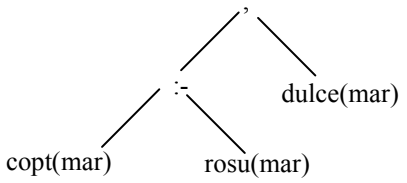

Fig. 2.2. Arborele pentru *(copt(mar):-rosu(mar)),dulce(mar)*

Precedenţa serveşte pentru ridicarea ambiguităţilor în expresiile în care nu sunt utilizate parantezele. Regula generală: operatorul cu precedenţa cea mai înaltă e operatorul principal. În Fig. 2.3 sunt prezentate precedenţele unor operatori. Deoarece operatorul *"+"* are o precedenţă mai joasă decât *"*"*, expresiile *A+B*C* şi *A+(B*C)* sunt echivalente şi ambele reprezintă expresia internă *+(A,*(B,C))*. Forma infixă **(+(A,B),C)* se va scrie cu paranteze *(A+B)*C*.

Parantezelor li se atribuie implicit numărul de precedenţă zero, pentru a preceda toţi operatorii.

Operator	Precedenţă
:-	1200
->	1200
,	100
\+	900
is	700
<	700
=	700
=<	700
>	700
>=	700
+	500
-	500
*	400
/	400

Fig. 2.3. Precedenţele unor operatori

2.1.3. Tipul de asociere

În cazul când operatorii în expresii au precedenţe diferite, construirea arborilor nu prezintă o problemă, dar atunci când în expresii sunt prezenţi operatori cu aceiaşi precedenţă, pot apărea ambiguităţi, necesitând astfel o tratare specială.

Fie că avem doi operatori *"+"* şi *"-"* cu aceeaşi precedenţă. Atunci expresia *A+B-C* poate privită, utilizând parantezele, în două moduri *(A+B)-C* şi *A+(B-C),* reprezentate în Fig. 2.4 a) şi Fig. 2.4 b), corespunzător. De obicei, se acceptă prima variantă şi se spune că ea este asociată în stânga.

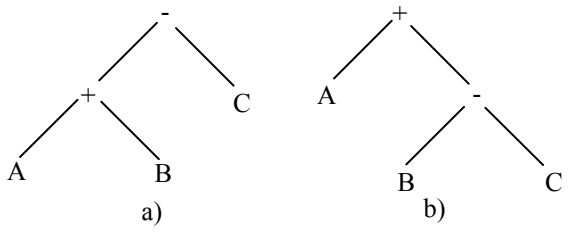

Fig. 2.4. Arborii pentru *(A+B)-C* şi *A+(B-C)*

Dar sunt situaţii când nu este permisă asocierea operatorilor nici în stânga, nici în dreapta. De exemplu, expresia de forma *a:-b:-c* nu poate fi acceptată în Prolog.

Pentru a ridica ambiguităţile de acest tip şi pentru a arăta ordinea executării operatorilor cu aceeaşi precedenţă într-o expresie, se introduce noţiunea de tip de asociere (Fig. 2.5). Aici *f* denotă însăşi operatorul, *x* indică că subarborele respectiv are în calitate de rădăcină un operator cu o precedenţă mai mică decât operatorul *f*, iar *y* indică că rădăcina subarborelui respectiv poate avea o precedenţă mai mică sau egală cu cea a operatorului *f*.

Poziţie operator	Tip de asociere
prefix	*fy* asociere în dreapta *fx* nu este asociere
infix	*yfx* asociere în stânga *xfy* asociere în dreapta *xfx* nu este asociere
prefix	*yf* asociere în stânga *xf* nu este asociere

Fig. 2.5. Tipuri de asociere a operatorilor

Să examinăm cazul operatorului infix. Pentru un operator de tipul *xfx,* functorii principali ai ambelor subexpresii, ce reprezintă argumentele operatorului, vor avea precedenţe mai mici decât a operatorului (în cazul când lipsesc parantezele). Operatorul de tipul *xfx* nu este asociativ. Un operator de tipul *xfy* permite o precedenţă mai mică în stânga, iar în dreapta o precedenţă mai mică sau egală. Adică este asociativ în dreapta. Dimpotrivă, operatorul *yfx* este asociativ în stânga.

Este evident că operatorii *"+"* şi *"-"* ambii sunt de tipul *yfx* şi, având aceeaşi precedenţă, expresia *A+B-C* va avea o singură interpretare *(A+B)-C,* adică regular *–(+(A,B),C)* şi nu *A+(B-C)*.

Pentru formele prefix şi infix pot fi admise prin analogie tipurile respective. De exemplu, operatorul prefix *"+"* are tipul de asociere *fx,* deoarece *++A* nu este o expresie corectă.

2.1.4. Declararea operatorilor

În Prolog (Turbo Prolog şi Visual Prolog fac excepţie) este posibilă declararea operatorilor ca un functor ce are în calitate de argumente precedenţa, *P*, tipul de asociere, *T*, şi numele operatorului, *O*, apelând la predicatul predefinit *op/3*. Atenţionăm, că directiva de declarare, de obicei, se scrie cu *":-"* în faţă:

```
:-op(P,T,O).
```

De exemplu, pentru a defini operatorul "->", se declară:

```
:-op(1200,xfy,"->").
```

după care, operatorul se poate utiliza în program cu aceste caracteristici.

Argumentul *O* poate fi şi o listă de nume (eventual unu singur) declarate cu acelaşi tip de asociere şi precedenţă.

```
:-op(1200,xfy,["->"]).
:-op(1200,xfx,["=","<>"]).
```

Pentru operatorii cu acelaşi nume, dar de diferită natură, de exemplu, infix şi prefix, predicatul *op/3* se scrie de mai multe ori:

```
:-op(500,yfx,["+","-"]).
:-op(500,fx, ["+","-"]).
```

Mulţimea de operatori a sistemului şi, respectiv, numerele de precedenţă variază la diverse implementări Prolog. Dar se poate de formulat o secvenţă de operatori principali cu precedenţe relative, ca în Fig. 3.3.

Este posibilă găsirea tipurilor de asociere şi precedenţelor operatorilor, independent de faptul dacă sunt predefinite sau declarate de utilizator. Cu acest ţel, în unele variante Prolog există operatorul *current_op/3*. De exemplu, se poate afla despre operatorul *"+"* următoarele, formulând întrebarea

```
?-current_op(X,Y,"+")
X=500, Y=fx;
X=500, Y=yfx
```

Prima soluţie este precedenţa şi tipul de asociere ale operatorului unar *"+"*, iar a doua – ale operatorului binar *"+"*. Menţionăm, că se pot obţine toţi operatorii şi caracteristicile lor, aplicând predicatul *fail/0*.

```
?-current_op(X,Y,Z),afisare_op(X,Y,Z),fail.
```

unde

```
afisare_op(Precedenta,Asocierea,Operator):-
       write("Operatorul ",Operator),
       write("        are        precedenta
",Precedenta),
       write(" si tipul ",Asocierea).
```

Este posibilă modificarea precedenţei şi tipului operatorului, inclusiv celui predefinit. Dar o astfel de practică nu este recomandată, deoarece aceşti operatori au un sens specific, iar modificarea lor poate afecta posibilităţile Prologului.

2.1.5. Un exemplu de aplicare

Presupunem, că se doreşte manipularea (fie pentru un sistem expert) regulilor într-un limbaj pseudonatural de forma

```
un animal care "are pene" e pasare
```

În *"are pene"* s-au utilizat ghilimelele pentru ai da o proprietate atomică.

În Prolog e posibilă definirea operatorilor *un*, *care* şi *e*. Operatorul *un* va fi un operator prefix, de tipul *fx* (el nu poate avea doi de *un* în şir). Operatorii *care* şi *e* vor fi infixi de tipul *xfx* (nu pot fi mulţi operatori *care* şi *e* consecutiv). Ordinea relativă de precedenţă (descrescătoare) va fi *e*, *care*, *un*. Argumentul din stânga al lui *e* este *un animal care "are pene"*, cel din dreapta - *pasare*. Argumentul din stânga al lui *care* este *un animal*, iar cel din dreapta este *"are pene"*. În notaţie prefixată propoziţia de mai sus are forma

```
e(care(un(animal),"are pene"),pasare)
```

În consecinţă, vom putea declara

```
:-op(104,xfx,e).
:-op(103,fx,un).
:-op(102,xfx,care).
```

Întrucât aceşti operatori nu vor interacţiona cu alţii, precedenţa este doar relativă. Acum, pot fi puse unele întrebări.

```
?-un X care P e Y
X=animal
P="are pene"
Y=pasare

?-X e Y
X=un animal care "are pene"
Y=pasare

?-un X care Y
no
```

Astfel, declaraţiile operatorilor pot servi pentru analiza sintactică a expresiilor.

2.2. Aritmetica

Aritmetica şi posibilităţile de evaluare a expresiilor aritmetice în Prolog sunt similare celor prestate de limbajele procedurale Basic, Pascal şi C. În toate sistemele Prolog, efectuarea operatorilor aritmetici este asigurată de o mulţime de predicate predefinite, predicate speciale, care acceptă în calitate de argumente expresii aritmetice, evaluarea cărora furnizează rezultate în formă de variabile instanţiate. În Prolog, de asemenea, poate fi definită aritmetica logică.

2.2.1. Expresii aritmetice

O expresie aritmetică este sau un singur număr, sau un termen construit, pornind de la numere, variabile, operatori şi paranteze. Numerele pot fi întregi (integer) sau reale (real), de exemplu, *10, -3, 2.01*. Forma de reprezentare a numerelor depinde de varianta de implementare a sistemului. Parantezele se utilizează pentru a schimba precedenţa aplicării operatorilor. Principalii operatori sunt cei descrişi în secţiunea respectivă:

```
:-op(500,yfx,[+,-]).
:-op(500, fx,[+,-]).
:-op(400, yfx,[*,/,div]).
:-op(350, xfx,[mod]).
```

Valoarea unei expresii poate fi calculată, dacă toate variabilele sunt instanţiate la momentul evaluării. Calculul se face în ordinea determinată de precedenţa operatorilor aritmetici. Primii se evaluează operatorii cu cea mai înaltă precedenţă.

Aşa dar, Prologul poate face toate patru operaţii aritmetice (adunarea: +, scăderea: -, înmulţirea: * şi împărţirea: /) cu numere întregi şi reale. Tipul rezultatului este determinat în concordanţă cu tabelul din Fig. 2.6.

Operandul 1	Operatorul	Operandul 2	Rezultatul
întreg	+, -, *	întreg	Întreg
real	+, -, *	întreg	Real
întreg	+, -, *	real	Real
real	+, -, *	real	Real
întreg sau real	/	întreg sau real	Real
întreg	div	întreg	Întreg
întreg	mod	întreg	Întreg

Fig. 2.6. Tipul rezultatului operațiilor aritmetice

În cazul aritmeticii mixte, când operanzii pot avea semne, rezultatul va avea întotdeauna semn.

Să menționăm, că o expresie aritmetică poate fi scrisă şi în sintaxa specială a termenilor structurați. Astfel expresia

*-X*Y+2*Z*

poate fi scrisă ca un termen standard Prolog (aşa este reprezentată în calculator această expresie) de forma

+(-((X,Y)),*(2,Z)).*

Expresiile aritmetice sunt evaluate în următoarea ordine:

- dacă expresia conține subexpresii (în paranteze), ele sunt evaluate primele
- dacă expresia conține înmulțirea (*) sau împărțirea (/, div, mod), aceste operații se execută următoarele, de la stânga la dreapta
- în sfârşit, se execută de la stânga la dreapta operațiile adunarea (+) şi scăderea (-)

Nume	Descriere
XmodY	Restul împărțirii lui *X* la *Y*
XdivY	Partea întreagă de la împărțirea lui *X* la *Y*
abs(X)	Valoarea absolută a lui *X*
cos(X)	Cosinus de *X*, unde *X* e în radiani
sin(X)	Sinus de *X*
tan(X)	Tangenta de *X*
arctan(X)	Arctangenta de *X*
exp(X)	*e* la puterea *X*
ln(X)	Logaritmul lui *X*, baza *e*
log(X)	Logaritmul lui *X*, baza *10*
sqrt(X)	Rădăcina patrată din *X*

Fig. 2.7. Predicate matematice predefinite

În diverse versiuni Prolog se pot găsi, în afară de precedentele, şi alte predicate predefinite matematice, unele scrise în formă prefixată. În Fig. 2.7 sunt prezentate unele predicate din sistemul Turbo Prolog.

2.2.2. Evaluarea expresiilor

Procesul de unificare nu evaluează o expresie, ci suprapune succesiv componentele simbolice. Astfel, dacă variabila *X* este instanțiată cu *+(Y,1)*, *Y* cu *+(Z,2)* şi *Z* cu *3*, instanțierea finală este *+(+(3,2),1)* şi nu *6*.

Pentru a asigura sensul evaluării unei expresii, Prologul utilizează predicatul *is*. În limbajele Turbo Prolog şi Visual Prolog predicatul *is* lipseşte. Aceste versiuni Prolog utilizează *"="* în loc de *is*.

Predicatul *is* este predefinit şi este un operator infix. Primul său argument e o variabilă noninstanțiată sau un întreg (integer), iar al doilea argument este o expresie aritmetică ce nu conține variabile neinstanțiate. Predicatul *is* unifică primul său argument cu rezultatul evaluării argumentului secund.

Să considerăm câteva exemple.

```
?-Y is 10*4
Y=40
?-40 is 10*4
yes
?-X is 50, Y is X/2
X=50, Y=25
```

Iar întrebările

```
?-a is 4-3
?-2-1+1 is 4-2
?-5 is 4+a
?-X is X+1
```

eşuează.

Acum pot fi aduse exemple de utilizare a unor predicate matematice predefinite.

Predicatul *XmodY* găseşte restul împărțirii lui *X* la *Y*, unde *X* şi *Y* sunt numere întregi.

```
?-Z is 7mod4
Z=3
?-Z is 4mod7
Z=4
```

Predicatul *XdivY* găseşte partea întreagă de la împărțirea *X/Y*, unde *X* şi *Y* sunt numere întregi

```
?-Z is 7div4
Z=1
?-Z is 4div7
Z=0
```

În cazul predicatului *abs(X),* dacă *X* este instanțiată cu o valoare pozitivă *val*, atunci *abs(X)* este această valoare; în caz contrar calculează *-1*val*.

```
?-Z=abs(-7)
Z=7
```

Predicatul *cos(X)* găseşte cosinusul argumentului său

```
?-Pi=3.141592653, Z is cos(Pi)
Z=-1
```

iar predicatul *exp(X)* găseşte *e* ridicat la puterea numărului cu care e instanțiată variabila *X*.

```
?-Z is exp(2.5)
Z=12.182493961
```

Menționăm, că condiția, care impune variabilele argumentului din dreapta lui *is* să fie instanțiate, face predicatul ireversibil. În general, predicatele aritmetice în care este forțată evaluarea expresiei sunt o abatere de la funcționarea normală a unificării, adică comit o îndepărtare de la semnificația declarativă uzuală.

2.2.3. Confruntarea expresiilor aritmetice

Prologul poate compara expresii aritmetice la fel ca şi caractere, simboluri şi secvențe de caractere. Predicatele reprezentate în Fig. 2.8 pot fi utilizate pentru compararea

aritmetică în Turbo Prolog. Fiecare din ele au nevoie de două expresii aritmetice în calitate de argumente. Argumentele sunt evaluate înainte ca compararea să se producă.

Predicate	Relație
<	mai mic
<=	mai mic sau egal
=	egal
>	mai mare
>=	mai mare sau egal
<>,><	diferit

Fig. 2.8. Predicate de comparare

În unele implementări Prolog, de exemplu Prolog^{++}, predicatul =:= denotă egal, iar =\= denotă diferit. În limbajul Turbo Prolog şi Visual Prolog predicatul = joacă şi rolul predicatului *is*.

Aceste predicate, de exemplu, pot fi utilizate astfel:

```
?-4=2+2
yes
?-4=3+2
no
?-4<>2+2
no
?-4<>3+2
yes
?-11+12+13=3*12
yes
?-2-3<0
yes
```

2.2.4. Aritmetica logică

Vom folosi o altă reprezentare a numerelor naturale şi vom scrie programe logice ce utilizează aritmetica acestor numere. Definiția numerelor naturale este una recursivă şi se bazează pe două obiecte: simbolul constantă *0* (zero) şi o funcție unară *s*. Argumentul acestei funcții este un număr natural şi *s(X)* denotă numărul natural ce urmează după *X*. Astfel, numerele naturale pot fi reprezentate sub forma *0, s(0), s(s(0)), s(s(s(0))),...*Numerele naturale pot fi definite printr-un program logic:

```
numar_natural(0).
numar_natural(s(X)):-numar_natural(X).
```

Predicatul *numar_natural(X)* afirmă că *X* este un număr natural. Deci, programul constă dintr-o clauză - condiție limită şi o clauză recursivă. Adică *0* e reprezentat de *0*, iar *N+1* este reprezentat de *s(X),* dacă *X* e reprezentat de *N*.

Acum vom defini predicatul *plus1(X,Y,Z),* ce reprezintă suma a două numere naturale. Aici *Z=X+Y*.

```
plus1(0,X,X).
plus1(s(X),Y,Z):-
       plus1(X,s(Y),Z).
```

Predicatul *plus1/3* utilizează acumulatorul pentru păstrarea rezultatelor intermediare în procesul calculului recursiv. E posibilă şi definirea unui predicat ce modelează operația adunării fără acumulator, utilizând compoziția substituțiilor.

```
plus2(0,X,X).
plus2(s(X),Y,s(Z)):-
       plus2(X,Y,Z).
```

În acest caz particular complexitățile acestor două metode sunt aceleaşi. Ambele predicate *plus1/3* şi *plus2/3* sunt declarative şi pot fi utilizate pentru a calcula atât suma a două numere naturale, cât şi deferența lor, iar cu careva restricții, şi pentru generarea tuturor perechilor de numere naturale suma cărora este dată. De exemplu, poate fi pusă întrebarea

```
?-plus1(X,Y,s(s(s(0)))).
```

Următoarele patru clauze demonstrează diferite definiții ale predicatului *minus(X,Y,Z)*, utilizând *plus1/3* şi *plus2/3*, respectiv.

```
minus1a(X,Y,Z):-plus(Y,Z,X).
minus1b(X,Y,Z):-plus(Z,Y,X).
minus2a(X,Y,Z):-plus2(Y,Z,X).
minus2b(X,Y,Z):-plus2(Z,Y,X).
```

Bineînțeles, că diferența a două numere naturale poate fi definită şi recursiv.

```
minus1(X,0,X).
minus1(s(X),s(Y),Z):-
       minus1(X,Y,Z).
```

Să se compare cu predicatul *minus2/3,* care se bazează pe o altă caracteristică a scăderii. Remarcăm, că *minus2/3* presupune existența soluției. A se încerca întrebarea:

```
?-minus2(0,s(0),Z)
```

Ce s-a întâmplat?

```
minus2(X,X,0).
minus2(X,Y,s(Z)):-
       minus2(X,s(Y),Z).
```

Programele de mai sus pot fi utilizate pentru descrierea unor relații mai complexe dintre numere. Un exemplu tipic este înmulțirea, reprezentată de predicatul *inmultirea(X,Y,Z)*, unde *X* înmulțit cu *Y* produce *Z*.

```
inmultirea(0,X,0).
inmultirea(s(X),Y,Z):-
       inmultirea(X,Y,W),
       plus1(W,Y,Z).
```

La rândul său, ridicarea la putere se definește sub forma unei înmulțiri repetate. În predicatul *exp(N,X,Y)* numerele *N*, *X* şi *Y* sunt naturale, *Y* este *X* ridicat la puterea *N*.

```
exp(_,0,0).
exp(0,_,s(0)).
exp(s(N),X,Y):-
       exp(N,X,Z),
       inmultirea(Z,X,Y).
```