

- **Operatori**

- **Operatori aritmetici:**

- sunt o rescriere infixata a unor structuri si de aceea valoarea expresiei definite cu ei nu este calculata;
 - evaluarea expresiei se face la cerere in cazul in care se foloseste operatorul predefinit infixat *is*, ca in exemplul: X is 1+2 (in acest caz se instantiaza variabila X la valoarea 3).

- **Operatori relationali:**

- sunt predicate predefinite infixate;
 - cel mai important este operatorul de egalitate, care functioneaza ca si cand ar fi definit prin urmatorul fapt:
 $X = X$ (VEZI satisfacerea unui scop de tipul $X = Y$, prin incercarea de a unifica X cu Y).

- Un exemplu:

-

- $?- X = 1+2.$

- $?- X \text{ is } 1+2.$

-

- Prolog raspunde
raspunde

- Prolog

-

- $X = 1+2$

- $X = 3$

-

- deoarece, in primul caz, expresia $1+2$ denota o structura (termen) Prolog, unde $+$ este functorul, iar 1 si 2 sunt argumentele sale. Nimic din acest scop nu declanseaza adunarea, care este declansata de operatorul *is*.

-

Operatori relationali – continuare:

- Operatorul de inegalitate \neq se defineste ca un predicat opus celui de egalitate; scopul $X \neq Y$ reuseste daca scopul $X=Y$ nu este satisfacut si esueaza daca $X=Y$ reuseste (semnificatie: valorile lui X si Y nu sunt egale).
- Predicatul $==$ testeaza echivalenta a doua variabile; $X==Y$ reuseste ori de cate ori $X=Y$ reuseste, dar reciproca este falsa. Este vorba de egalitatea *literala* a doi termeni.
- $X==Y$ este adevarat daca termenii X si Y sunt identici, adica au exact aceeasi structura si toate componentele corespunzatoare coincid. In particular, numele de variabile trebuie sa fie aceleasi.
- Relatia complementara (de non-identitate) este \neq

Un exemplu:

?- $f(a, X) == f(a, Y)$.

no

- Operatorul $==$ face numai evaluare aritmetica si nici o instantiere; semnificatia lui $X == Y$ este: valorile *expresiilor aritmetice* X si Y sunt egale.

Diferenta dintre operatorii $=$ si $==$

?- $1+2 == 2+1$.

yes

?- $1+2 = 2+1$.

no

- Inegalitatea a doua expresii aritmetice se stabileste cu operatorul \neq (vezi folia anterioara).

Valorile expresiilor aritmetice pot fi comparate prin intermediul operatorilor care urmeaza. Acesti operatori forteaza evaluarea argumentelor lor.

$X > Y$ (X este mai mare ca Y)

$X < Y$ (X este mai mic ca Y)

$X \geq Y$ (X mai mare sau egal decat Y)

$X \leq Y$ (X mai mic sau egal cu Y)

$X ::= Y$ (valorile lui X si Y sunt egale)

$X \neq Y$ (valorile lui X si Y nu sunt egale)

Operatii aritmetice

- Exista proceduri incorporate care pot fi utilizate pentru efectuarea operatiilor aritmetice.
- **Efectuarea operatiilor aritmetice trebuie sa fie ceruta in mod explicit de catre procedura incorporata *is*.**
- **Exista proceduri incorporate asociate operatorilor predefiniti $+$, $-$, $*$, $/$, div si mod .**
- **In momentul in care se efectueaza evaluarea, toate argumentele trebuie sa fie deja instantiate la anumite numere.**
- **Valorile expresiilor aritmetice pot fi comparate prin intermediul unor operatori cum ar fi $<$, $=<$ etc. (vezi folia anterioara).
Acesti operatori forteaza evaluarea argumentelor lor.**

- Operatorul infixat *mod* da *restul* unei impartiri intregi.
- Operatorul infixat *div* da *catul* unei impartiri intregi.
- Operatorul */* poate sa desemneze impartire intreaga sau reala, in functie de implementare. (De obicei se refera la impartirea reala, iar *div* la cea intreaga).

➤ Comentariile in Prolog sunt precedate de caracterul *%*

➤ Operatori definiti de utilizator

- Definierea de noi operatori de catre programator se face prin introducerea in program a unor clauze de forma speciala, numite directive.
- Definierea de noi operatori se face cu ajutorul directivei

op(precedenta_operator, tip_operator, nume_operator).

precedata de simbolul :-

Exemplu:

:- op (600, xfx, are).

caz in care este legala expresia

coco are pene

Operatori definiti de utilizator – continuare

- Directivele actioneaza ca o definire de noi operatori ai limbajului, in care se specifica *numele, precedenta si tipul* (infixat, prefixat sau postfixat) operatorului.
- Nu se asociaza nici o operatie operatorilor definiti de programator.
- Operatorii noi astfel definiti sunt utilizati ca functori numai pentru a combina obiecte in structuri si nu pentru a executa actiuni asupra datelor.
- Exemplu: In loc de a utiliza structura

are (coco, pene)

se poate defini un nou operator *are*

:- op (600, xfx, are).

caz in care este legala expresia

coco are pene

- Operatorii sunt atomi, iar *precedenta* lor trebuie sa fie o valoare intreaga intr-un anumit interval si corelata cu precedenta operatorilor predefiniti in limbaj.
- *Tipul* operatorilor fixeaza caracterul infixat, prefixat sau postfixat al operatorului si precedenta operanzilor sai. Tipul operatorului se defineste utilizand una din urmatoarele forme standard:

(1) operatori infixati: xfx xfy yfx

(2) operatori prefixati: fx fy

(3) operatori postfixati: xf yf

unde f reprezinta operatorul, iar x si y operanzii sai.

- Utilizarea simbolului x sau a simbolului y depinde de precedenta operandului fata de operator.

Precedenta operanzilor se defineste astfel:

- ✓ un argument intre paranteze sau un argument nestructurat are precedenta 0;
 - ✓ un argument de tip structura are precedenta egala cu cea a functorului operator.
- Semnificatiile lui x si y in stabilirea tipului operatorului sunt urmatoarele:

- ✓ x reprezinta un argument (operand) cu precedenta strict mai mica decat cea a functorului (operatorului) f :

$$precedenta(x) < precedenta(f)$$

- ✓ y reprezinta un argument (operand) cu precedenta mai mica sau egala cu cea a functorului (operatorului) f :

$$precedenta(y) \leq precedenta(f)$$

- Numele operatorului poate fi orice atom Prolog care nu este deja definit in Prolog. Se poate folosi si o lista de atomi, daca se definesc mai multi operatori cu aceeasi precedenta si acelasi tip.

Exemplu

`:- op (100, xfx, [este, are]).`

`:- op (100, xf, zboara).`

`coco are pene.`

`coco zboara.`

`coco este papagal.`

`bozo este penguin.`

`?- Cine are pene.`

`Cine = coco`

`?- Cine zboara.`

`Cine = coco`

`?- Cine este Ce.`

`Cine = coco, Ce = papagal;`

`Cine = bozo, Ce = penguin;`

`no`

In conditiile in care se adauga la baza de cunostinte anterioara si definitia operatorilor *daca, atunci si si*

:- op (870, fx, daca).

:- op (880, xfx, atunci).

:- op (880, xfy, si).

urmatoarea structura este valida in Prolog:

**daca Animalul are pene
si Animalul zboara
atunci Animalul este pasare.**

Controlul procesului de Backtracking: *cut* si *fail*

- Predicatul *cut*, notat cu atomul special **!** este un predicat standard, fara argumente, care se indeplineste (este adevarat) intotdeauna si nu poate fi resatisfacut.
- Comportarea predicatului *cut* este urmatoarea:

(C1) $H :- D_1, D_2, \dots, D_m, !, D_{m+1}, \dots, D_n.$

(C2) $H :- A_1, \dots, A_p.$

(C3) $H.$

- Daca D_1, D_2, \dots, D_m sunt satisfacute, ele nu mai pot fi resatisfacute datorita lui *cut* (se inhiba backtracking-ul).
- Daca D_1, D_2, \dots, D_m sunt satisfacute, C2 si C3 nu vor mai fi utilizate pentru resatisfacerea lui H. Resatisfacerea lui H se poate face numai prin resatisfacerea unuia din scopurile D_{m+1}, \dots, D_n , daca acestea au mai multe solutii.

Observatie: Exista doua contexte diferite in care se poate utiliza predicatul *cut*, si anume: intr-un context predicatul *cut* se introduce numai pentru cresterea eficientei programului, caz in care el se numeste *cut verde*; in celalalt context utilizarea lui *cut* modifica semnificatia procedurala a programului, caz in care el se numeste *cut rosu*. (La *cut*-ul verde semnificatia procedurala a programului este aceeaasi, adica nu conteaza ordinea in care se scriu clauzele. La un *cut* rosu efectul programului este total diferit daca se schimba ordinea clauzelor).

Cut rosu

Introducerea unui *cut* rosu modifica corespondenta dintre semnificatia declarativa si semnificatia procedurala a programelor Prolog. El permite exprimarea in Prolog a unor structuri de control de tipul

Daca conditie atunci actiune₁
altfel actiune₂

astfel:

daca_atunci_altfel (Cond, Act1, Act2) :- Cond, !, Act1.

daca_atunci_altfel (Cond, Act1, Act2) :- Act2.

Obs.: Utilizarea predicatului *cut* in definirea predicatului asociat structurii de control *daca_atunci_altfel* introduce un *cut rosu* deoarece efectul programului este total diferit daca se schimba ordinea clauzelor.

Exemplu

Definirea predicatului de aflare a minimului dintre doua numere, in doua variante:

min1(X, Y, X) :- X=<Y,!. **% cut verde**

min1(X, Y, Y) :- X>Y.

min2(X, Y, X) :- X=<Y,!. **% cut rosu**

min2(X, Y, Y).

Ordinea clauzelor de definire a lui min1 poate fi schimbata fara nici un efect asupra rezultatului programului. In cazul predicatului min2 se utilizeaza un cut rosu, asemanator structurii

daca_atunci_altfel

Daca se schimba ordinea clauzelor de definire a predicatului min2:

min2(X,Y,Y).

min2(X, Y, X) :- X=<Y, !.

rezultatul programului va fi incorect pentru valori $X < Y$.