5

Negaţia

Aici se tratează problema reprezentării informaţiei negative, se introduce noţiunea de negaţie prin eşec şi se consideră diverse capcane logice ale implementării negaţiei în Prolog. Nu va fi argumentat teoretic de ce nu poate fi dedusă o informaţie negativă dintr-un program. Vom confirma această afirmaţie doar printr-un exemplu. Fie un program constă din următoarele fapte:

```
floare(trandafir).
floare(narcis).

arbore(stejar).
arbore(plop).
```

Scopul *non floare(stejar)* nu este o consecinţă logică a programului de mai sus, aşa cum nu este consecinţă logică şi scopul *floare(stejar).* Aici *non* s-a utilizat în sensul clasic al operatorului de negaţie, care semnifică că propoziţia asupra căreia el se aplică este negată.

5.1. Ipoteza lumii închise

Un mod de soluţionare a problemei de mai sus constă în utilizarea unei reguli speciale de inferenţă, numită ipoteza lumii închise, conform căreia se acceptă, că dacă un predicat nu este o consecinţă logică a unui program, atunci se consideră că negaţia lui este consecinţă. De obicei, negaţia unui predicat în Prolog se notează, de exemplu,

```
not(floare(stejar)).
```

Se utilizează *not/1* pentru a deosebi această negaţie de negaţia clasică notată cu *non*.

În exemplul precedent, aplicând această regulă, se face concluzia că *not(floare(stejar))* este o consecinţă logică, în timp ce *floare(stejar)* nu e consecinţă logică a programului. Cu alte cuvinte, dacă nu este posibilă demonstrarea unui predicat de bază, negaţia acestui predicat se consideră adevărată.

Astfel, dacă sistemul Prolog răspunde la o întrebare *no,* aceasta nu atestă falsitatea ei, ci atestă numai că întrebarea dată nu este demonstrabilă cu faptele şi regulile prezente în program. Deci, la întrebarea

```
?-floare(stejar)
```

se obţine răspunsul

```
no
```

şi atunci, bazându-ne pe ipoteza lumii închise, la întrebarea

```
?-not(floare(stejar))
```

se obţine răspunsul

```
yes
```

5.2. Negaţia prin eşec

Negaţia prin eşec este termenul utilizat pentru negaţia, ce se bazează pe ipoteza lumii închise, implementată în Prolog. Astfel, întrebarea

```
?-not(Subscop)
```

reuşeşte, dacă scopul *Subscop* eşuează. Predicatul *not/1* este un predicat predefinit în mai multe sisteme Prolog. El are în calitate de argument un subscop pe care îl evaluează şi întoarce valoarea de adevăr. Dacă subscopul reuşeşte, rezultatul este eşec, iar dacă subscopul eşuează, rezultatul reuşeşte. De aici şi provine denumirea de negaţie prin eşec.

Să menţionăm, că predicatul *not/1* poate fi modelat cu ajutorul asocierii *!,fail*, dacă el lipseşte în sistem. Şi anume:

```
not(Subscop):-Subscop,!,fail.
not(_).
```

Predicatul *not(Subscop)*, ce realizează ipoteza lumii închise, termină cu succes, dacă *Subscop* eşuează şi – eşuează, în caz contrar. În această definiţie, dacă *Subscop* reuşeşte, conjuncţia subscopurilor *!,fail* face, în primul rând (*!/0*), să fie abandonată încercarea de a satisface a doua clauză şi, în al doilea rând (*fail/0*), provoacă un eşec al acestei clauze, deci, şi a predicatului în întregime. Adică, scopul not(*Subscop*) reuşeşte, dacă şi numai dacă *Subscop* nu poate fi satisfăcut.

Există o mulţime de probleme puţin studiate, ce ţin de logica negaţiei prin eşec. Iată doar câteva exemple ce ne arată că utilizarea negaţiei e delicată. Considerăm un program ce constă dintr-o singură clauză

```
egal(X,X).
```

şi să observăm răspunsurile la întrebările ce urmează.

```
?-not(egal(a,b))
yes

?-not(egal(a,a))
no
```

Răspunsurile sunt raţionale. Dar dacă sunt formulate întrebările

```
?-egal(X,a), not(egal(X,b))
X=a

?-not(egal(X,b)), egal(X,a)
no
```

atunci răspunsul la a doua întrebare este derutant. El, însă, are explicaţie. Predicatul *egal(X,b)* reuşeşte şi unifică *X* cu *b* şi deci, *not(egal(X,b))* eşuează.

Predicatul *not/1* poate fi folosit nu numai în întrebări, dar şi în corpul clauzelor. Revenim la unul din programele precedente, în care se definea predicatul *sora_lui(X,Y)* şi care genera o problemă simplă: orice femeie este soră sie însăşi ?. Modificăm programul precum urmează:

```
femeie(maria).
femeie(eliza).
femeie(emilia).

parinti(emilia, maria, paul).
parinti(andrei, maria, paul).
parinti(eliza, maria, paul).

egal(X,X).

sora_lui(X,Y):-
       not(egal(X,Y)),
       femeie(X),
       parinti(X,M,T),
       parinti(Y,M,T).
```

şi formulăm câteva întrebări:

```
?-sora_lui(eliza,emilia)
yes

?-sora_lui(X,emilia)
no

?-sora_lui(X,Y)
no
```

Răspunsurile neaşteptate provin din faptul că predicatul *sora_lui/2* conţine argumente variabile. Pentru ca predicatul să dea răspunsuri bune în toate cazurile, e necesar, când se ajunge la *not(egal(X,Y))*, ca variabilele să fie instanţiate. Deci, o versiune corectă a predicatului *sora_lui/2* poate fi

```
sora_lui(X,Y):-
       femeie(X),
       parinti(X,M,T),
       parinti(Y,M,T),
       not(egal(X,Y)).
```

Predicatul *not/1* nu corespunde în totul negaţiei logice. Astfel, într-o bază vidă de fapte *not/1* al unui predicat poate da în calitate de răspuns *yes*, dacă Prologul nu găseşte faptele în baza de fapte. Conform ipotezei lumii închise, dacă faptul nu este definit adevărat, atunci el este considerat fals chiar dacă lipseşte în baza de fapte.

Trebuie de asemenea să se ţină cont că o dublă negaţie nu întotdeauna e identică absenţei negaţiei. Astfel, de exemplu, pentru predicatul *apartine/2* avem

```
?-apartine(X,[a,b,c])

X=a;
X=b;
X=c
```

în schimb

```
?-not(not(apartine(X,[a,b,c])))
```

eşuează.

Ce se petrece în acest caz? Se ştie, că dacă un scop eşuează, atunci variabilele sale devin neinstanţiate. Deoarece *apartine(X,[a,b,c])* reuşeşte cu unificarea *X/a*, *not(apartine(X,[a,b,c]))* eşuează şi, prin urmare, variabila *X* e eliberată.

5.3. Capcanele regulilor implicite

Tăierea roşie în programarea logică deseori presupune existenţa unor subscopuri implicite. Folosirea predicatului *!/0* pentru funcţionarea implicită a intrat deja în clasica programării logice. Vom arăta în baza a două exemple simple, că e mai bine uneori de recurs la alte forme logice de descriere cu ajutorul predicatului *not/1,* decât la redarea implicită a subscopurilor în clauze.

Fie, de exemplu, clauzele

```
paritatea(X,impar):-impar(X),!.
paritatea(X,par).
```

împreună cu o definiţie a predicatului *impar/1*.

Întrebarea

```
?-paritatea(7,X)
X=impar
```

reuşeşte, utilizându-se prima clauză. Dar presupunem, că prima clauză eşuează din cauza unui subscop din definiţia predicatului *impar/1* sau din altă cauză. Atunci întrebarea se unifică cu a doua clauză şi se obţine răspunsul *X=par*. Acesta nu este un comportament dorit. Soluţia constă în utilizarea predicatului *not/1,* care redă o mai mare expresivitate faptului că nu este nevoie de definiţia unui predicat *par/1*.

```
paritatea(X,impar):-impar(X),!.
paritatea(X,par):-not(impar(X)).
```

Să examinăm, acum, un program ce descrie plata pensiilor. Predicatul *pensie(Persoana,Pensie)* defineşte tipul de pensie prescris unei persoane.

```
pensie(X,handicap):-handicap(X).
pensie(X,batranete):-
depasit_65(X),platit(X).
pensie(X,indemnizatii):-depasit_65(X).

handicap(ionescu).

depasit_65(ionescu).
depasit_65(munteanu).
depasit_65(vasilache).

platit(ionescu).
platit(munteanu).
```

Prima clauză afirmă că unei persoane cu handicap îi este stabilită o pensie pentru handicap. A doua clauză susţine că persoanei ce a depăşit vârsta de 65 ani i se fixează o pensie de bătrâneţe, dacă, bineînţeles, ea a făcut o perioadă de timp depuneri în fondul de pensii corespunzător. Această condiţie e reprezentată de predicatul *platit/1*. Persoanelor ce au depăşit vârsta de 65 ani li se vor plăti indemnizaţii pentru bătrâneţe, chiar dacă nu au completat vre-un fond de pensii.

Să lărgim acest program cu o clauză, care ne spune, că persoana care nu face parte nici dintr-o categorie, nu va primi nimic. Soluţia va consta în adăugarea predicatului *!/0* în fiecare din cele trei reguli şi a faptului implicit

```
pensie(X,nimic).
```

Atunci programul va arăta astfel:

```
pensie(X,handicap):-handicap(X),!.
```

```
pensie(X,batranete):-
depasit_65(X),platit(X),!.
pensie(X,indemnizatii):-depasit_65(X),!.
pensie(X,nimic).

handicap(ionescu).

depasit_65(ionescu).
depasit_65(munteanu).
depasit_65(vasilache).

platit(ionescu).
platit(munteanu).
```

Trebuie de menționat, că programul funcționează corect când e vorba de fixarea pensiei unei persoane, de exemplu

```
?-pensie(ionescu,X)
```

Dar, în general, programul nu este corect. Răspunsul la întrebarea de mai jos confirmă acest lucru.

```
?-pensie(ionescu,nimic)
yes
```

De asemenea, la întrebarea ce urmează obținem numai un singur răspuns, care tot nu corespunde realității.

```
?-pensie(X,batranete)
X=ionescu
```

Aici tăierile au împiedicat căutarea tuturor soluțiilor. O ieşire din această situație poate fi introducerea unui predicat nou *stabilit(X,Y),* care este adevărat, dacă persoanei *X* îi este stabilită pensia *Y*. Predicatul se defineşte cu ajutorul a două reguli.

```
stabilit(X,Y):-pensie(X,Y).
stabilit(X,nimic):-not(pensie(X,_)).
```

Aceste două clauze trebuie adăugate la prima variantă a programului

```
stabilit(X,Y):-pensie(X,Y).
stabilit(X,nimic):-not(pensie(X,_)).

pensie(X,handicap):-handicap(X).
pensie(X,batranete):-
depasit_65(X),platit(X).
pensie(X,indemnizatii):-depasit_65(X).

handicap(ionescu).

depasit_65(ionescu).
depasit_65(munteanu).
depasit_65(vasilache).

platit(ionescu).
platit(munteanu).
```

Programul astfel obținut are toate calitățile pozitive ale variantei a doua, dar e liber de toate dezavantajele lui enumerate mai sus.

Acest exemplu ne arată nu numai cum predicatul *not/1* poate îmbunătăți un program, dar şi cât e de complexă utilizarea predicatului *!*/0 şi ce capcane ne poate întinde.

5.4. Ipoteza lumii deschise

Negația prin eşec poate fi văzută ca o adăugare implicită la predicatele din program a tuturor negațiilor lor. În majoritatea cazurilor nu este posibilă adăugarea explicită a negațiilor într-o bază de fapte, când numărul de fapte negative despre un domeniu concret pot fi mult mai multe decât numărul faptelor pozitive. De exemplu, la întrebarea

```
?-floare(lacramioare)
```

vom obține răspunsul

```
no
```

Acest *no* denotă că aşa fapt nu există în baza de fapte, dar nu că lăcrămioarele nu sunt flori.

În aşa situații, dacă considerăm incomplete cunoştințele exprimate de baza de date, se adoptă ipoteza lumii deschise. Ultima, contrar celei precedente, presupune adevărate toate datele prezente. Mai exact, sunt reprezentate explicit toate faptele, fie pozitive, fie negative, iar celelalte rămase, implicit se consideră necunoscute.

În acord cu ipoteza lumii deschise o întrebare poate avea una din următoarele valori de adevăr: *adevărat, fals, necunoscut*. Dacă o întrebare este evaluată ca necunoscută, programul poate face acțiuni speciale, cum ar fi consultarea altor surse de cunoştințe. Implicit Prologul se conduce de ipoteza lumii închise. Un program trebuie descris explicit pentru ca să funcționeze conform ipotezei lumii deschise.

Iată un exemplu simplu ce funcționează în acord cu ipoteza lumii deschise. Răspunsul poate fi unul din cele trei valori de adevăr. Faptele false sunt explicit declarate în program. Ele au forma *fals(C).* Dacă un fapt nu poate fi determinat că este adevărat sau fals, el e considerat necunoscut.

```
demonstrare(C):-
       C,
       write("**adevarat"),nl,!.
demonstrare(C):-
       fals(C),
       write("**fals"),nl,!.
demonstrare(C):-
       not(C),
       not(fals(C)),
       write("**necunoscut"),nl,!.

tata_lui(ion, vasile).

fals(tata_lui(_, ion)).

?-demonstrare(tata_lui(ana, vasile))
**necunoscut

?-demonstrare(tata_lui(ion, vasile))
**adevarat

?-demonstrare(tata_lui(ion, X))
**adevarat
X=vasile

?-demonstrare(tata_lui(maria,ion))
**fals
```