

Predicatul *fail*

- Prolog permite exprimarea directa a esecului unui scop cu ajutorul predicatului *fail* – un predicat:
 - standard,
 - fara argumente,
 - care esueaza intotdeauna.
- Dupa *fail* nu se mai poate satisface nici un scop.
- Introducerea unui predicat *fail* intr-o conjunctie de scopuri, de obicei la sfarsit (caci dupa *fail* nu se mai poate satisface nici un scop), determina *intrarea in procesul de backtracking*.
- Daca *fail* se intalneste dupa predicatul *cut*, nu se mai face backtracking.

Exemplul 1 (folosirea predicatului *fail* pentru a determina esecul):

Enuntul “Un individ este rau daca nu este bun” se poate exprima astfel:

rau (X) :- bun (X),!,fail.

rau (X).

Exemplu de program – folosirea lui *fail* pentru a determina esecul:

bun (gelu).

bun (vlad).

bun (mihai).

rau (X) :- bun (X),!,fail. (*)

rau (X). ()**

Exemplu de executie a programului:

?- rau (gelu).

no

?- rau (mihai).

no

rau (petru).

yes

Comentariu:

- **la prima interogare:** din clauza (*) avem rau (gelu) daca bun (gelu), care este adevarat din primul fapt al bazei de cunostinte; apoi ! este intotdeauna adevarat si urmeaza *fail* care genereaza esec; datorita existentei lui *cut*, clauza (**) nu va mai fi utilizata pentru resatisfacerea scopului; deci raspunsul ramane *no*, ca si in al doilea caz;
- **la a treia interogare:** pentru clauza (*) ar trebui sa am bun (petru), dar acest fapt nu exista in baza de cunostinte; deoarece bun(petru) nu a fost satisfacut, am voie sa utilizez clauza (**); clauza (**) furnizeaza rau (petru), deci satisface scopul curent; atunci raspunsul este “yes”.

Observatie: Atunci cand este folosit pentru a determina esecul, *fail* este de obicei precedat de

cut, deoarece procesul de backtracking pe scopurile care il preced este inutil, scopul esuand oricum, datorita lui *fail*.

Exemplul 2 – introducerea predicatului *fail* pentru a genera procesul de backtracking pe scopurile care il preced:

rosu (mar).

rosu (cub).

rosu (soare).

afisare (X) :- rosu (X), write (X), fail.

afisare (_). (*)

Comentariu: Intrucat, pentru fiecare obiect considerat, scopul afisare (X) esueaza datorita lui *fail*, se trece la clauza (*), care afiseaza obiectul respectiv, adica raspunde *yes*. Trecerea la clauza (*) este posibila deoarece prima clauza nu contine *cut* inainte de *fail*. In acest fel, vor fi afisate toate obiectele rosii cunoscute de programul Prolog, datorita procesului de backtracking generat de *fail*; in acest fel se realizeaza, prin *fail*, o iteratie

peste faptele rosu (). Clauza afisare () este adaugata pentru ca raspunsul final la satisfacerea scopului sa fie afirmativ. (Aceasta clauza raspunde *yes* la orice).

Cum lucreaza programul anterior cu si fara *cut*
inaintea lui *fail*:

1. Cazul cand NU avem *cut* inaintea lui *fail*:

programul raspunde YES la orice (datorita clauzei a doua); afiseaza (datorita lui write (X)) numai obiectele rosii cunoscute de program (pentru aceste obiecte se scrie denumirea lor si apoi *yes*).

2. Cazul cand inaintea lui *fail* exista *cut*:

- ✓ Pentru un obiect rosu cunoscut de program este scris numele obiectului si se raspunde NO (pentru ca, datorita lui *cut*, nu se mai ajunge la clauza a doua).
- ✓ Pentru un obiect necunoscut de program nu se mai afiseaza nimic si se raspunde YES. (Aceasta deoarece, nefiind satisfacute clauzele dinaintea lui *cut*, se ajunge la

**clauza a doua, care genereaza raspunsul
YES).**

Observatie: Combinatia **!,fail** poate fi utilizata cu rol de *negatie*.

Exemplu: “Mihai iubeste toate sporturile cu exceptia boxului.”

Pseudocod:

daca X este sport si X este box

atunci Mihai iubeste X este fals

altfel daca X este sport

atunci Mihai iubeste X este adevarat

PROLOG

Baza de cunostinte:

box(box).

sport(tenis).

sport(polo).

sport(innot).

sport(box).

Varianta 1 (cut rosu):

iubeste (mihai, X) :- sport (X), box (X), !, fail.

iubeste (mihai, X) :- sport (X).

Comentariu: Predicatul *cut* utilizat aici este un cut rosu.

Comentariu: Predicatul *cut* utilizat aici este un *cut* rosu. Combinatia **!,fail** este utilizata cu rol de negatie (pentru ca *fail* raspunde *no*, iar *cut* ma impiedica sa folosesc clauza urmatoare). Se mai spune ca limbajul Prolog modeleaza negatia ca esec al satisfacerii unui scop (negatia ca insucces), aceasta fiind, de fapt, o particularizare a ipotezei lumii inchise. Combinatia **!,fail** este echivalenta cu un predicat standard existent in Prolog, si anume predicatul *not*.

- Predicatul *not* admite ca argument un predicat Prolog si reuseste daca predicatul argument esueaza.
- In Sicstus Prolog sintaxa pentru predicatul *not* este \+

Varianta 2 (cu predicatul *not*):

iubeste (mihai, X) :- sport (X), not (box(X)).

iubeste (mihai, X) :- sport (X).

Varianta 1 – program:

box(box).

sport(tenis).

sport(polo).

sport(innot).

sport(box).

iubeste(mihai,X):-sport(X),box(X),!,fail.

iubeste(mihai,X):-sport(X).

Exemple de interogari:

?- sport(X).

X=tenis ?;

X=polo ?;

X=innot ?;

X=box ?;

no

?-sport(tenis).

yes

?- iubeste(mihai,tenis).

yes

?- iubeste(mihai,box).

no

Varianta 2 – program:

box(box).

sport(tenis).

sport(polo).

sport(innot).

sport(box).

iubeste(mihai,X) :- sport(X),\+(box(X)).

iubeste(mihai,X) :- sport(X).

Exemple de interogari:

- La primele trei interogari

?- sport(X).

?- sport(tenis).

?-iubeste(mihai,tenis).

rezultatul este identic cu cel de la programul anterior.

- La ultima interogare rezultatul difera:

?- iubeste(mihai,box).

yes

Aici raspunsul este YES deoarece, dupa ce esueaza prima clauza, se trece la urmatoarea, care este satisfacuta. (Intrucat nu exista predicatul *cut*, se poate trece la clauza urmatoare).

Observatie: Chiar daca prima clauza este satisfacuta, se trece oricum la clauza urmatoare (a doua) pentru ca este posibil ca ea sa furnizeze o noua solutie.

Predicatul *call*

- *call* este un alt predicat standard. El admite ca argument un predicat Prolog si are ca efect incercarea de satisfacere a predicatului argument.
- *call* reuseste daca predicatul argument reuseste si esueaza in caz contrar.
- Utilizand acest predicat, se poate explicita efectul general al predicatului standard *not* astfel:

$\text{not}(P) \text{ :- call}(P),!,\text{fail.}$

$\text{not}(P).$

Observatie: Atat predicatul *not*, cat si predicatul *call* sunt predicate de ordinul II in Prolog, deoarece admit ca argumente alte predicate.

Alte predicate incorporate

Predicatul =..

- Se scrie ca un operator infixat.
- Scopul

Term =.. L

este satisfacut daca L este o lista ce contine principalul functor al lui Term, urmat de argumentele sale.

Exemplu:

?- f(a,b) =.. L.

L = [f,a,b]

?- T =.. [dreptunghi, 3, 5].

T = dreptunghi(3,5)

?- Z =.. [p,X,f(X,Y)].

Z = p(X,f(X,Y))

Observatie: Acest predicat descompune un termen in componentele sale – adica functorul sau si argumentele acestuia.

Predicatul *bagof*

Scopul

bagof(X,P,L)

va produce lista L a tuturor obiectelor X astfel incat sa fie satisfacut un scop P.

Observatie: Aceasta are sens numai daca X si P au unele variabile comune.

Exemplu: Presupunem ca programul contine urmatoarele fapte

varsta (petru,7).

varsta(ana,5).

varsta(patricia,8).

varsta(tom,5).

Atunci putem obtine lista tuturor copiilor care au 5 ani prin urmatoarea interogare:

?-bagof(Copil,varsta(Copil,5),Lista).

Lista = [ana, tom]

Predicatul *findall*

`findall(X,P,L)`

produce, de asemenea, o lista de obiecte care satisfac P. Diferenta fata de *bagof* este aceea ca sunt colectate toate obiectele X, indiferent de solutii posibil diferite pentru variabile din P care nu sunt partajate cu X.

Daca nu exista nici un obiect X care sa satisfaca P, atunci *findall* va avea succes cu $L = []$.

Predicate standard de intrare / iesire
(intotdeauna adevarate)

➤ **Predicatul *write***

- are forma $\text{write}(E_1, E_2, \dots, E_k)$

unde E_1, E_2, \dots, E_k sunt variabile sau obiecte elementare. Orice variabila trebuie sa fie legata in momentul scrierii!

- *nl* face trecerea pe linia urmatoare

➤ **Predicatul *readln***

Permite citirea unui string sau simbol. Valoarea obtinuta in urma citirii este legata la X din

$\text{readln}(X)$

LISTE

O lista este de forma [Cap|Coadă].

Operatii cu liste:

1. Apartenenta la o lista

- se foloseste predicatul *membru(X,L)*, unde X este un obiect si L este o lista. X este membru al lui L daca

(1) X este capul lui L

sau

(2) X este membru al cozii lui L

ADICA

$\text{membru}(X, [X|\text{Coadă}])$.

$\text{membru}(X, [\text{Cap}|\text{Coadă}]) :- \text{membru}(X, \text{Coadă})$.

2. Concatenarea

- se foloseste relatia $\text{conc}(L_1, L_2, L_3)$
- definitia predicatului *conc* este:

$\text{conc}([], L, L).$

$\text{conc}([X|L_1], L_2, [X|L_3]) :- \text{conc}(L_1, L_2, L_3).$

3. Adaugarea unui element

- elementul adaugat devine noul cap:

$\text{add}(X, L, [X|L]).$

4. Stergerea unui element

- se foloseste relatia $\text{del}(X, L, L_1)$
- definitia predicatului *del* este:

$\text{del}(X, [X|Coadă], Coadă).$

$\text{del}(X, [Y|Coadă], [Y|Coadă1]) :- \text{del}(X, Coadă, Coadă1).$

Observatie: Inserarea unui element intr-o lista poate fi definita folosind relatia *del*, astfel:

insert(X, Lista, ListaMaiMare):-

del (X, ListaMaiMare, Lista).

Subliste

relatia **sublista([c,d,e], [a,b,c,d,e,f])** este adevarata, dar **sublista([c,e], [a,b,c,d,e,f])** nu este adevarata.

Definitie:

S este o sublista a lui L daca

(1) L poate fi descompusa in doua liste, L1 si L2

si

(2) L2 poate fi descompusa in doua liste, S si o alta lista L3

adica

sublista(S,L) :- conc(L1,L2,L), conc(S,L3,L2).