

predicat(*listă de variabile si constante*).

De fapt e: predicat(lista de termeni).

---

De exersat la laborator:

?- var(A). <b>true.</b>	?- atom(a). <b>true.</b>	?- number(a). <b>false.</b>
?- nonvar(A). <b>false.</b>	?- atom(alta_constanta). <b>true.</b>	?- number(10+2). <b>false.</b>
?- var(_). <b>true.</b>	?- atom(-#-->). <b>true.</b>	?- number(12). <b>true.</b>
?- nonvar(_). <b>false.</b>	?- atom(1). <b>false.</b>	?- number(12.5e2). <b>true.</b>
?- var('A'). <b>false.</b>	?- atom([]). <b>false.</b>	?- integer(5). <b>true.</b>
?- nonvar('A'). <b>true.</b>	?- atom(p(X)). <b>false.</b>	?- integer(-5). <b>true.</b>
?- var(a). <b>false.</b>	?- atom(X). <b>false.</b>	?- integer(5.0). <b>false.</b>
?- nonvar(a). <b>true.</b>	?- atomic(a). <b>true.</b>	?- float(5). <b>false.</b>
?- var(1). <b>false.</b>	?- atomic(alta_constanta). <b>true.</b>	?- float(5.0). <b>true.</b>
?- nonvar(1). <b>true.</b>	?- atomic(-#-->). <b>true.</b>	?- float(-5.0e2). <b>true.</b>
?- var(p(X)). <b>false.</b>	?- atomic(1). <b>true.</b>	?- float(-5e2). <b>true.</b>
?- nonvar(p(X)). <b>true.</b>	?- atomic([]). <b>true.</b>	?- float(5e0). <b>true.</b>
?- var(-->). <b>false.</b>	?- atomic(p(X)). <b>false.</b>	?- float(-15.25e-2). <b>true.</b>
?- nonvar(-->). <b>true.</b>	?- atomic(X). <b>false.</b>	?- float(-15.25E-2). <b>true.</b>
?- compound([]). <b>false.</b>		
?- compound(X+1). <b>true.</b>		

(Operatori pentru unificare, respectiv testarea egalității, inegalității, literal identității: predicate binare infixate predefinite)

```
= testeaza egalitatea ca expresii; face si unificarea, in unele versiuni de Prolog  
cu testarea ocurentelor variabilelor in termeni, in alte versiuni fara acest test  
unify_with_occurs_check face unificarea cu testarea ocurentelor variabilelor in termeni  
\= testeaza nonegalitatea ca expresii  
:= testeaza egalitatea ca valori de expresii aritmetice calculate  
\\= testeaza nonegalitatea ca valori de expresii aritmetice calculate  
== testeaza literal identitatea  
\\= testeaza literal nonidentitatea  
\\+ sau not reprezinta negatia  
=< este mai mic sau egal  
>= este mai mare sau egal
```

Si de facut:

```
concat([],L,L).  
concat([H|T],L,[H|M]) :- concat(T,L,M).  
  
inversa([],[]).  
inversa([H|T],L) :- inversa(T,M), concat(M,[H],L).
```

## Predicatul *cut* (!)

Observați, mai jos, predicatul zeroar (adică fără argumente) *fail*, care întotdeauna eșuează, adică întotdeauna e evaluat la **false**, și predicatul unar predefinit *not* sau  $\backslash +$ , care primește drept argument un predicat și întoarce **true** când acel predicat eșuează și **false** când acel predicat e satisfăcut.

După cum am văzut mai sus, pentru satisfacerea unui scop, faptele și regulile sunt aplicate în ordinea în care sunt scrise în baza de cunoștințe, printr-un backtracking încorporat în interpretorul Prologului.

Predicatul predefinit *cut* ("!") are funcția de a tăia backtrackingul executat pentru satisfacerea unui scop dintr-o interogare, astfel că, după ce scopul respectiv e satisfăcut prin aplicarea regulii în care apare *cut*, nu se mai caută alte satisfaceri ale acelui scop.

### Exemplu (din fișierul *testcut.p*)

Predicatul următoare testează apartenența unui element la o listă:

```
membru(_,[]) :- fail. %% corect si: not(membru(_,[])).
membru(H,[H|_]).
membru(X,[_|T]) :- membru(X,T).

apartine(_,[]) :- fail. %% corect si: not(apartine(_,[])).
apartine(H,[H|_]) :- !.
apartine(X,[_|T]) :- apartine(X,T).
```

## De facut la curs:

La fel se caută soluțiile pentru predicatul care produce nu numai unificări de termeni, ci și calcule, ca, de exemplu, predicatul pentru calculul lungimii unei liste, afișări pe ecran, de exemplu cu predicatul unar predefinit *write*, care scrie argumentul său pe ecran, apoi întoarce **true**, etc.

- De exemplu, pentru baza de cunoștințe:

```
lung([],0).
lung([_|T],N) :- lung(T,K), N is K+1.
```

și interogare:

```
?- lung([1,2,3],NrElem).
```

Scopul din această interogare unifică doar cu membrul stâng al regulii din această bază de cunoștințe:

Țădar următorul scop este scopul compus:

```
?- lung([2,3],K), N is K+1.
```

Să reținem că predicatul predefinit binar infix *is* produce calculul expresiei aritmetice din argumentul său drept, apoi unificarea între valoarea obținută din acest calcul și argumentul său stâng.

Pentru aceeași bază de cunoștințe ca mai sus, să vedem cum răspunde Prologul la interogare:

```
?- lung(CeLista,3).
```

Unificarea  $0 = 3$  nu are loc, Țădar acest scop nu unifică, cu faptul, dar unifică, cu membrul stâng al regulii din această bază de cunoștințe:

Rezultatul acestei unificări este:  $CeLista = [H|T]$  și  $N = 3$ , Țădar următorul scop este:

```
?- lung(T,K), 3 is K+1.
```

Primer termen al conjuncției care dă acest scop compus unifică atât cu faptul, cât și cu membrul stâng al regulii:

```
?- lung(T2,K2), N2 is K2+1.
```

Primer termen al acestei conjuncții unifică atât cu faptul, cât și cu membrul stâng al regulii:

Acest scop este satisfăcut cu prima dintre aceste unificări:  $T2 = []$  și  $K2 = 0$ , urmată de  $N2 = 0 + 1$ , care produce unificarea  $N2 = 1$ . Țădar scopul lung( $T1, K1$ ),  $N1$  is  $K1 + 1$  este satisfăcut cu unificarea:  $T1 = [H2|T2]$  și  $K1 = N2 = 1$ , urmată de  $N1 = 1 + 1$ , care produce unificarea  $N1 = 2$ .

Prolog-ul satisface apoi scopul lung( $T, K$ ),  $3$  is  $K + 1$  cu unificarea:  $T = [H1|H2]$  și  $K = N1 = 2$ , urmată de  $3$  is  $2 + 1$ , care produce unificarea  $3 = 3$ . Iar scopul inițial, lung( $CeLista, 3$ ), este acum satisfăcut cu unificarea:  $CeLista = [H1|H2]$  și  $N = 3$ .

Dacă mai cerem soluții, Prolog-ul va efectua și unificarea  $T2 = [H3|T3]$  și  $K2 = N3$ , urmată de scopul compus lung( $T3, K3$ ),  $N3$  is  $K3 + 1$ . Ca mai sus, unificarea scopului lung( $T3, K3$ ) cu faptul din baza de cunoștințe conduce la  $T = [H1, H2, H3]$  și  $3$  is  $3 + 1$ , care produce unificarea  $3 = 4$ . Aceasta eșuează, Țădar Prolog-ul va aplica regula din baza de cunoștințe pentru satisfacerea scopului lung( $T3, K3$ ). Urmând pașii de mai sus, se va ajunge la  $T = [H1, H2, H3, H4]$  și  $3$  is  $4 + 1$ , care produce unificarea  $3 = 5$ , care eșuează. Toate aceste următoare unificări:  $3 = 6$ ,  $3 = 7$  și a.m.d. eșuează, până la umplerea stivei de lucru.

Primer dintre aceste două scopuri unifică tot numai cu membrul stâng al regulii:

Următorul scop este:

```
?- lung([3],K1), N1 is K1+1.
```

din cadrul căruia primer scop unifică numai cu membrul stâng al regulii:

Următorul scop este acest scop compus, din cadrul căruia primer scop unifică numai cu faptul:

```
?- lung([],K2), N2 is K2+1.
```

Unificarea  $K2 = 0$  conduce la:  $N2 = 1$ , pentru că  $N2$  is  $0 + 1$  produce unificarea  $N2 = 1$ , apoi întoarce **true**, și se continuă în acest mod la revenirea din pașii acestui backtracking:  $K1 = N2 = 1$ ,  $K = N1 = 2$  ( $K1 + 1$ ),  $NrElem = N = 3$  ( $K + 1$ ). Țădar unica soluție a interogării de mai sus este:  $NrElem = 3$ .

Prima dintre aceste unificări, având rezultatul  $T = []$  și  $K = 0$ , produce satisfacerea scopului lung( $T, K$ ) (fiind unificare cu un fapt). Prolog-ul trece apoi la satisfacerea membrului drept al conjuncției de mai sus, cu această valoare pentru  $K$ :  $3$  is  $0 + 1$ , conducând la unificarea  $3 = 1$ , care eșuează.

Rezultatul celei de-a doua dintre aceste unificări este:  $T = [H1|T1]$  și  $K = N1$ , Țădar următorul scop este:

```
?- lung(T1,K1), N1 is K1+1.
```

Primer termen al acestei conjuncții unifică atât cu faptul, cât și cu membrul stâng al regulii:

Prima dintre aceste unificări, având rezultatul  $T1 = []$  și  $K1 = 0$ , produce satisfacerea scopului lung( $T1, K1$ ). Prolog-ul trece apoi la satisfacerea membrului drept al acestei conjuncții, cu această valoare pentru  $K1$ :  $N1$  is  $0 + 1$ , care conduce la unificarea  $N1 = 1$ . Unificarea anterioară devine:  $T = [H1|T1]$  și  $K = N1 = 1$ , iar scopul anterior devine: lung( $[H1], 1$ ),  $3$  is  $1 + 1$ . Membrul drept al acestei conjuncții duce la unificarea  $3 = 2$ , care eșuează.

A doua dintre aceste unificări, având rezultatul  $T1 = [H2|T2]$  și  $K1 = N2$ , conduce la scopul:

- Acum să considerăm următoarea bază de cunoștințe:

```
concat([],L,L).
concat([H|T],L,[H|M]) :- concat(T,L,M). % (r0)

inversa([],[]).
inversa([H|T],L) :- inversa(T,M), concat(M,[H],L). % (r1)
```

și interogare:

```
?- inversa([1,2,3],CeLista).
```

Cum  $[1, 2, 3] = [1, 2, 3]$  nu unifică, cu  $[]$ , scopul *inversa*( $L, [1, 2, 3]$ ) unifică doar cu membrul stâng al regulii ( $r1$ ):

Țădar următorul scop este scopul compus:

```
?- inversa([2,3],M), concat(M,[1],CeLista).
```

Ca mai sus, Prologul caută să satisfacă mai întâi primerul dintre aceste 2 scopuri,