

PROGRAMARE LOGICĂ

Cursul I

Claudia MUREȘAN
cmuresan@fmi.unibuc.ro, c.muresan@yahoo.com

Universitatea din București
Facultatea de Matematică și Informatică
București

2019–2020, Semestrul II

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

Prescurtări uzuale care vor fi folosite în lecții

- **i. e.** (*id est*) = adică
- **a. î.** = astfel încât
- **ddacă** = dacă și numai dacă
- **ș. a. m. d.** = și așa mai departe
- --- : să se demonstreze că
- ∇ : contradicție

Paradigme ale programării calculatoarelor

Programarea imperativă:

- programatorul trebuie să-i spună calculatorului, pas cu pas, ce să facă:
“parcurge cu un indice această mulțime, la fiecare iterație verifică această condiție...” etc..

Limbaje de programare imperativă: *Pascal, C, Java* etc..

Programarea logică/declarativă:

- programatorul descrie un cadru de lucru, și cere calculatorului să rezolve o cerință în acel cadru;
- pe baza unui backtracking și a altor tehnici de programare încorporate în interpretorul/compilerul limbajului de programare logică folosit, calculatorul determină proprietățile pe care le poate folosi pentru a rezolva cerința respectivă și ordinea în care trebuie să le aplice.

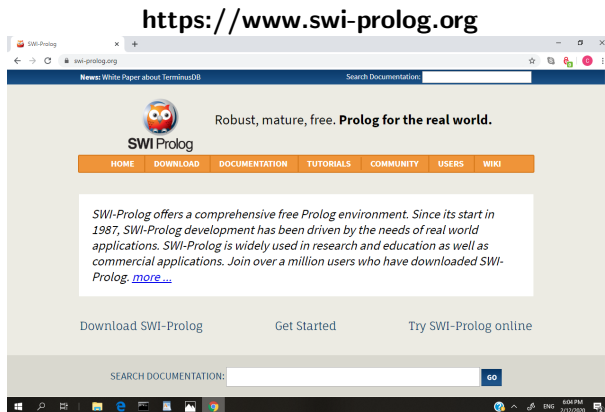
Limbaje de programare logică/declarativă

- *Prolog* (PROGRAMMING IN LOGIC): bazat pe **predicate**, pe relații între obiecte; utilizat pentru jocuri/deduții logice, în procesarea limbajului natural etc.;
- *CafeObj*, *Maude*: bazate pe **specificatii**, pe descrierea unor tipuri de structuri algebrice; destinate demonstrării automate de proprietăți matematice; utilizate și în verificarea sistemelor software;
- *Haskell*: bazat pe **funcții** și **recursii**;

recursia este foarte importantă în toate limbajele de programare logică: este principalul mijloc de calcul, înlocuind, de exemplu, instrucțiunile repetitive

- etc..

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente



Din josul paginii de la această adresă:

- se poate descărca SWI-Prolog;
- se poate căuta în documentația SWI-Prolog;
- se poate accesa versiunea online a SWI-Prolog.

De exemplu, dând o căutare după "boolean expressions", găsim:

The screenshot shows a web browser window with the address bar displaying `swi-prolog.org/pldoc/man?section=clpb-exprs`. The page title is "SWI-Prolog -- Manual". On the left, there is a navigation menu with the following items: "Documentation", "Reference manual", "The SWI-Prolog library", "library(clpb): CLP(B): Constraint L", "Introduction", "Boolean expressions" (highlighted in blue), "Interface predicates", "Examples", "Obtaining BDDs", "Enabling monotonic CLP(B)", "Example: Pigeons", "Example: Boolean circuit", "Acknowledgments", "CLP(B) predicate index", and "Packages". The main content area is titled "A.8.2 Boolean expressions" and contains the text: "A Boolean expression is one of:". Below this text is a table with two columns. The first column lists various expressions, and the second column provides their meaning. The table is as follows:

0	false
1	true
<i>variable</i>	unknown truth value
<i>atom</i>	universally quantified variable
$\sim Expr$	logical NOT
$Expr + Expr$	logical OR
$Expr * Expr$	logical AND
$Expr \# Expr$	exclusive OR
$Var \wedge Expr$	existential quantification
$Expr = Expr$	equality
$Expr \neq Expr$	disequality (same as #)
$Expr < Expr$	less or equal (implication)
$Expr >= Expr$	greater or equal
$Expr < Expr$	less than
$Expr > Expr$	greater than
$card(Is, Exprs)$	cardinality constraint (see below)
$\vee(Exprs)$	n-fold disjunction (see below)
$\wedge(Exprs)$	n-fold conjunction (see below)

where *Expr* again denotes a Boolean expression.

The Boolean expression $card(Is, Exprs)$ is true iff the number of true expressions in the list *Exprs* is a member of the list *Is* of integers and integer ranges of the form *From-To*. For example, to state that precisely two of the three variables *X*, *Y* and *Z* are true, you can use $sat(card([2], [X, Y, Z]))$.

$\vee(Exprs)$ and $\wedge(Exprs)$ denote, respectively, the disjunction and conjunction of all elements in the list *Exprs* of Boolean expressions.

Atoms denote parametric values that are universally quantified. All universal quantifiers appear implicitly in front of the entire expression. In residual goals, universally quantified variables always appear on the right-hand side of equations. Therefore, they can be used to express functional dependencies on input variables.

Structura unui program în Prolog

Un program în Prolog este format din **clauze**; acestea sunt de trei tipuri:

1 fapte:

propoziție.

predicat(*listă de variabile si constante*).

predicatele exprimă relații între obiecte, proprietăți ale obiectelor, și au valori booleene;

propozițiile sunt predicatele fără variabile, i.e. constantele booleene;

faptele semnifică proprietăți întotdeauna adevărate;

2 reguli:

fapt :- succesiune de fapte.

faptele din partea dreaptă a unei reguli pot fi separate prin virgulă (care reprezintă *conjunctia logică*), punct și virgulă (*disjunctia logică*, având prioritate mai mică decât conjunctia) sau alți conectori logici (a se vedea slideul anterior); simbolul ":-" (două puncte și minus) este numit "neck";

(semnificația unei reguli)

are loc acest fapt :- dacă au loc aceste fapte.

3 întrebări (interogări):

formate din **scopuri**, constând în predicate care trebuie satisfăcute de către Prolog: prin scopurile din interogări cerem

Prologului să determine dacă un predicat e adevărat sau valori ale variabilelor pentru care acel predicat e adevărat.

Faptele și regulile formează baza de cunoștințe.

Numele de **variabile** în Prolog încep cu *literă mare* sau *underscore* (`_`).

Variabilă nedenumită: *underscore* (`_`):

- simbol generic pentru variabile care apar o singură dată într-un fapt sau o regulă;
- sunt folosite doar pentru a indica locații de variabile, nu și pentru a lucra cu acele variabile.

Orice alt nume, inclusiv șiruri de caractere cuprinse între apostrofuri, denumește o constantă, o funcție cu argumente (i.e. operație cu cel puțin un operand) sau un predicat.

Sintaxa pentru liste în Prolog:

`[]` lista vidă

`[elem1, elem2, ..., elemn]` lista formată din elementele *elem₁*, *elem₂*, ..., *elem_n*

`[elem1, elem2, ..., elemn | T]` lista cu primele *n* elemente *elem₁*, *elem₂*, ..., *elem_n* și coada *T*

Exemplu

`[1, 2, 3, 4] = [1, 2, 3, 4 | []]` `= [1 | [2, 3, 4]]` `= [1, 2 | [3, 4]]` `= [1 | [2, 3 | [4]]]` `= [1 | [2, 3 | [4 | []]]]`

Exemplu

Cum putem scrie predicate pentru:

- calculul lungimii unei liste;
- concatenarea a două liste?

Să scriem un predicat $lung(L, N)$, care este satisfăcut ddacă:

- L este o listă, N este un număr natural și
- N este lungimea listei L , adică numărul elementelor lui L :

$lung([], 0).$

$lung([_|T], N) :- lung(T, K), N \text{ is } K + 1.$

Operatorul **is** produce executarea calculului în acea expresie aritmetică.

Înlocuiți **is** cu **=** și vedeți ce obțineți!

Și un predicat $concat(L1, L2, L)$, care este satisfăcut ddacă:

- $L1$, $L2$ și L sunt liste și
- concatenarea listei $L1$ cu $L2$ este L :

$concat([], L, L).$

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

Încărcăm acest program în versiunea online a SWI-Prolog

The screenshot displays the SWISH web interface for running Prolog code. The browser address bar shows 'swish.swi-prolog.org'. The SWISH application has a menu bar with 'File', 'Edit', 'Examples', and 'Help'. A search bar and a '359 users online' indicator are also present.

The main editor shows a Prolog program with the following code:

```
1 % Aici: fapte si reguli.
2
3 lung([],0). % fapt: intotdeauna adevarat
4 lung([_|T],N) :- lung(T,K), N is K+1. % regula: are structura:
5 % are loc acest fapt :- daca au loc aceste fapte.
6
7 /* Daca denumim capul listei [_|T] in loc sa-l dam ca
8 * variabila nedenumita: _, atunci primim avertismentul:
9 * variabila singleton. */
10
11 concat([],L,L). % fapt
12 concat([H|T],L,[H|M]) :- concat(T,L,M). % regula
13
14
15
16
17
18
19
20
21
22 % Aici: intrebari: formate din scopuri. ----->
23
```

The right-hand pane shows the execution results of several queries:

- `lung([a,b,c],3).` returns `true`.
- `lung([a,b,c],5).` returns `false`.
- `lung([a,b,c,d,e],Cat).` returns `Cat = 5`.
- `concat([1,2],[3,4,5],[1,2,3,4,5]).` returns `true`.
- `concat([1,2],[],[1,2,3,4,5]).` returns `false`.
- `concat([a,b,c],[1,2,3,4,5],CeLista).` returns `CeLista = [a, b, c, 1, 2, 3, 4, 5]`.
- The final query `?- concat([a,b,c],[1,2,3,4,5],CeLista).` is currently being executed, indicated by a question mark.

At the bottom of the interface, there are buttons for 'Examples', 'History', 'Solutions', and 'Run!'. A checkbox for 'table results' is also visible.

Interogări cu scopuri sau variabile multiple

The screenshot displays the SWISH web interface for SWI-Prolog. The left pane contains a Prolog program with comments in Romanian. The right pane shows the execution results, including variable bindings and the execution of a query.

Program:

```
1 % Aici: fapte si reguli.
2
3 lung([],0). % fapt: intotdeauna adevarat
4 lung([_|T],N) :- lung(T,K), N is K+1. % regula: are structura:
5 % are loc acest fapt :- daca au loc aceste fapte.
6
7 /* Daca denumim capul listei [_|T] in loc sa-l dam ca
8 * variabila nedenumita: _, atunci primim avertismentul:
9 * variabila singleton. */
10
11 concat([],L,L). % fapt
12 concat([H|T],L,[H|M]) :- concat(T,L,M). % regula
13
14
15 % Ati observat sintaxa pentru comentarii:
16 % pe un rand
17 /* pe mai
18 * multe randuri */
19
20
21
22 % Sa punem si intrebari formate din mai multe scopuri.
23
24 % Sa vedem si intrebari care au mai multe raspunsuri.
25 % Sa cerem cu "Next" mai multe rezultate.
26 % In loc de "Next", in versiunea desktop a SWI-Prolog, se foloseste ";".
27 % La final, se afiseaza "false", semnificand ca nu mai exista alte solutii.
28
```

Execution Results:

```
concat([a,b,c],[1,2,3],Ce),concat(Ce,[10,-1],Alta),lung(Alta,Cat).
Alta = [a, b, c, 1, 2, 3, 10, -1].
Cat = 8.
Ce = [a, b, c, 1, 2, 3]

concat(CeLista,[1,2,3],[-1,0,1,2,3]).
CeLista = [-1, 0]
false

concat([1,2,3],CuCeLista,[1,2,3,4,5]).
CuCeLista = [4, 5]

concat(CeLista,CuCeLista,[1,2,3]).
CeLista = [].
CuCeLista = [1, 2, 3]
CeLista = [1].
CuCeLista = [2, 3]
CeLista = [1, 2].
CuCeLista = [3]

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

Navigation buttons: Next, 10, 100, 1,000, Stop. Bottom buttons: Examples, History, Solutions, table results, Run.

Întrebări cu mai multe răspunsuri posibile

The screenshot displays the SWISH web interface, a tool for running Prolog code. The browser address bar shows the URL `swish.swi-prolog.org`. The SWISH application has a menu bar with `File`, `Edit`, `Examples`, and `Help`. A search bar and a status indicator showing "340 users online" are also present.

The main editor area contains a Prolog program with the following code:

```
1 % Aici: fapte si reguli.
2
3 lung([],0).                % fapt: intotdeauna adevarat
4 lung([_:T],N) :- lung(T,K), N is K+1. % regula: are structura:
5                               % are loc acest fapt :- daca au loc aceste fapte.
6
7 /* Daca denumim capul listei [_:T] in loc sa-l dam ca
8 * variabila nedenumita: _, atunci primim avertismentul:
9 * variabila singleton. */
10
11 concat([],L,L).            % fapt
12 concat([H:T],L,[H:M]) :- concat(T,L,M). % regula
13
14
15 % Ati observat sintaxa pentru comentarii:
16 % pe un rand
17 /* pe mai
18 * multe randuri */
19
20
21
22 % Sa punem si intrebari formate din mai multe scopuri.
23
24 % Sa vedem si intrebari care au mai multe raspunsuri.
25 % Sa cerem cu "Next" mai multe rezultate.
26 % In loc de "Next", in versiunea desktop a SWI-Prolog, se foloseste ";".
27 % La final, se afiseaza "false", semnificand ca nu mai exista alte solutii.
28
```

The right-hand pane shows the execution results for three queries:

```
concat(CeLista,CuCeLista,[1,2,3]).
CeLista = [],
CuCeLista = [1, 2, 3]
CeLista = [1],
CuCeLista = [2, 3]
CeLista = [1, 2],
CuCeLista = [3]
CeLista = [1, 2, 3],
CuCeLista = []
false

concat([1,2,X;Y],[5,Z,7,T],[1,2,3,4,5,6,7]).
false

concat([1,2,X;Y],[5,Z,7],[1,2,3,4,5,6,7]).
X = 3,
Y = 4,
Z = 6
```

At the bottom of the interface, there are buttons for `Examples`, `History`, `Solutions`, and a `Run!` button next to a `table results` checkbox.

Vom vedea cum găsește Prologul aceste răspunsuri

The screenshot shows the SWISH web interface for SWI-Prolog. The browser address bar shows `swish.swi-prolog.org`. The SWISH logo and navigation menu (File, Edit, Examples, Help) are at the top. A search bar and a notification icon (25) are also present. The main area is divided into two panes. The left pane, titled "Program", contains the following Prolog code:

```
1 lung([],0).
2 lung([_|T],N) :- lung(T,K), N is K+1.
3
4 concat([],L,L).
5 concat([H|T],L,[H|M]) :- concat(T,L,M).
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 % Sa vedem si intrebari care au mai multe raspunsuri.
23 % Sa cerem cu "Next" mai multe rezultate.
24 % In loc de "Next", in versiunea desktop a SWI-Prolog, se foloseste ";".
25 % La final, se afiseaza "false", semnificand ca nu mai exista alte solutii.
26
```

The right pane shows the execution results for two queries. The first query is `concat([1,2,X,Y],L,[1,2,3,4,5,6]).` with results:

```
L = [5, 6],
X = 3,
Y = 4
```

The second query is `concat([1,2,X,Y|T],L,[1,2,3,4,5,6]).` with results:

```
L = [5, 6],
T = [],
X = 3,
Y = 4
```

A greyed-out section shows the next results for the second query:

```
L = [6],
T = [5],
X = 3,
Y = 4
```

Below this, the results are:

```
L = [],
T = [5, 6],
X = 3,
Y = 4
```

Finally, the result `false` is displayed. At the bottom of the right pane, there is a button labeled "Run!" and a checkbox for "table results".

Vedeți și celelalte opțiuni ale SWI-Prolog online

The screenshot displays the SWI-Prolog online environment. The browser address bar shows `swish.swi-prolog.org`. The interface includes a menu bar with **File**, **Edit**, **Examples**, and **Help**. A search bar is located in the top right corner. The main editor area on the left contains the following Prolog code:

```
1 lung([],0).
2 lung([_|T],N) :- lung(T,K), N is K+1.
3
4 concat([],L,L).
5 concat([H|T],L,[H|M]) :- concat(T,L,M).
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20 % Am cerut 10 rezultate deodata. ----->
```

The right-hand pane shows the execution results, displaying the state of variables `L` and `M` after each step of the `concat` predicate. The results are as follows:

- Initial state: `L = [1, 2, 3].`, `M = []`
- Step 1: `concat(L,M,[1,2,3,4,5]).`
- Step 2: `L = [].`, `M = [1, 2, 3, 4, 5]`
- Step 3: `L = [1].`, `M = [2, 3, 4, 5]`
- Step 4: `L = [1, 2].`, `M = [3, 4, 5]`
- Step 5: `L = [1, 2, 3].`, `M = [4, 5]`
- Step 6: `L = [1, 2, 3, 4].`, `M = [5]`
- Step 7: `L = [1, 2, 3, 4, 5].`, `M = []`
- Step 8: `concat(L,M,[1,2,3,4,5,6,7]).`
- Step 9: `L = [].`, `M = [1, 2, 3, 4, 5, 6, 7]`

At the bottom of the results pane, there are buttons for `Next`, `10`, `100`, `1,000`, and `Stop`. The bottom of the interface features a footer with `Examples`, `History`, and `Solutions` tabs, a checkbox for `table results`, and a `Run!` button.

Inclusiv opțiuni pentru editare rapidă: vedeți butonul "History"

The screenshot displays the SWISH web interface for Prolog programming. The browser address bar shows `swish.swi-prolog.org`. The interface includes a search bar, a user count of 334 users online, and a navigation menu with `File`, `Edit`, `Examples`, and `Help`.

The main editor area contains a Prolog program:

```
1 lung([],0).  
2 lung([_|T],N) :- lung(T,K), N is K+1.  
3  
4 concat([],L,L).  
5 concat([H|T],L,[H|M]) :- concat(T,L,M).  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20 % Am cerut 10 rezultate deodata. ----->
```

The right-hand pane shows the execution history, displaying the state of variables `L` and `M` after each step of the `concat` predicate. The history includes:

- `L = [1, 2, 3].`
`M = []`
`concat(L,M,[1,2,3,4,5]).`
- `L = [].`
`M = [1, 2, 3, 4, 5]`
`L = [1].`
`M = [2, 3, 4, 5]`
`L = [1, 2].`
`M = [3, 4, 5]`
`L = [1, 2, 3].`
`M = [4, 5]`
`L = [1, 2, 3, 4].`
`M = [5]`
`L = [1, 2, 3, 4, 5].`
`M = []`
- `concat(L,M,[1,2,3,4,5,6,7]).`
`L = [].`
`M = [1, 2, 3, 4, 5, 6, 7]`

At the bottom of the history pane, there are buttons for `Next`, `10`, `100`, `1,000`, and `Stop`. Below the history pane, the current query is shown: `?- concat(L,M,[1,2,3,4,5,6,7]).`

The bottom of the interface features buttons for `Examples`, `History`, and `Solutions`, along with a checkbox for `table results` and a `Run!` button.

Și acum în versiunea desktop a SWI-Prolog

Am scris cele doua predicate de mai sus într-un fișier text cu extensia *.pl*, care poate fi încărcat În Prologul desktop cu **File** → **Consult** din meniul ferestrei interpretorului Prolog sau folosind comanda (atenție la [], apostrofuri și punct): `['unitate_disc:/calea_catre_fisier/nume_fisier.pl']`.

```

SWI-Prolog (Multi-threaded, version 8.0.3)
File Edit Settings Run Debug Help
For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- ['d:/work/coursurilexele/pl_2019_2020/laborator_pl_2019_2020/exspliste.pl'].
true

?- concat(L,M,[a,b,c]).
L = [],
M = [a,b,c] ;
L = [b,c],
M = [] ;
L = [c],
M = [a,b,c] ;
L = [],
M = [a,b,c] ;
false.

?- trace.
true.

[trace] ?- concat(L,M,[a,b,c]).
Call: (8) concat(_4518,_4520,[a,b,c]) ? creep
Exit: (8) concat([], [a,b,c], [a,b,c]) ? creep
L = [],
M = [a,b,c] ;
Redo: (8) concat(_4518,_4520,[a,b,c]) ? creep
Call: (9) concat(_4792,_4520,[b,c]) ? creep
Exit: (9) concat([], [b,c], [b,c]) ? creep
Exit: (9) concat([a], [b,c], [a,b,c]) ? creep
L = [a],
M = [b,c] ;
Redo: (9) concat(_4792,_4520,[b,c]) ? creep
Call: (10) concat(_4798,_4520,[c]) ? creep
Exit: (10) concat([], [c], [c]) ? creep
Exit: (10) concat([b], [c], [b,c]) ? creep
Exit: (10) concat([a,b], [c], [a,b,c]) ? creep
L = [a,b],
M = [c] ;
Redo: (10) concat(_4798,_4520,[c]) ? creep
Call: (11) concat(_4804,_4520,[], []) ? creep
Exit: (11) concat([], [], []) ? creep
Exit: (10) concat([c], [], [c]) ? creep
Exit: (9) concat([b,c], [], [b,c]) ? creep
Exit: (9) concat([a,b,c], [], [a,b,c]) ? creep
L = [a,b,c],
M = [] ;
Redo: (11) concat(_4804,_4520,[], []) ? creep
Fail: (10) concat(_4798,_4520,[c]) ? creep
Fail: (9) concat(_4792,_4520,[b,c]) ? creep
Fail: (8) concat(_4518,_4520,[a,b,c]) ? creep
false.

[trace] ?- notrace.
true.

```

Comanda `trace` produce detalierea pașilor urmați de Prolog pentru a rezolva interogările. Renunțare la această afișare detaliată: `notrace`.

trace și săgeată "Step into" în SWI-Prolog online

The screenshot displays the SWISH (SWI-Prolog for SHaring) web interface. The browser address bar shows `swish.swi-prolog.org`. The interface is divided into a left pane for the Prolog program and a right pane for the execution trace.

Program Pane:

```
1 lung([],0).
2 lung([_|T],N) :- lung(T,K), N is K+1.
3
4 concat([],L,L).
5 concat([H|T],L,[H|M]) :- concat(T,L,M).
6
```

Trace Pane:

The trace shows the execution of `trace,concat(L,M,[sgl])`. It details the recursive calls to `concat` and the state of variables `L` and `M` at each step.

Trace 1: `trace,concat(L,M,[sgl])`

- Call: `concat(_3768,_3764,[sgl])`
- Exit: `concat([], [sgl], [sgl])`
- `L = []`
- `M = [sgl]`
- Redo: `concat(_3768,_3764,[sgl])`
- Call: `concat(_4448,_3764, [])`
- Exit: `concat([], [], [])`
- Exit: `concat([sgl], [], [sgl])`
- `L = [sgl]`
- `M = []`
- Redo: `concat(_4448,_3764, [])`
- Fail: `concat(_4448,_3764, [])`
- Fail: `concat(_3768,_3764,[sgl])`

Trace 2: `trace,concat(L,M,[1,2,3])`

- Call: `concat(_3868,_3864,[1,2,3])`
- Exit: `concat([], [1,2,3], [1,2,3])`

Below the trace, there is a prompt `?-` followed by `trace,concat(L,M,[1,2,3]).`

At the bottom of the trace pane, there are buttons for "Examples", "History", "Solutions", and a "Run!" button. A checkbox for "table results" is also present.

The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 12:05 AM on 2/14/2020.

Tipurile de date în Prolog

Practic, există un *unic tip de date*: **termenii**, cu, câteva *subtipuri*:

- **variabilele**;
- **atomii**: șirurile de caractere speciale și constantele, inclusiv șirurile de caractere cuprinse între apostrofuri, dar excluzând lista vidă `[]` și numerele;
- **numerele** $\begin{cases} \text{întregi și} \\ \text{reale;} \end{cases}$
- **termenii compuși**: $f(arg_1, \dots, arg_n)$, unde $n \in \mathbb{N}^*$, f este o funcție (și predicatele sunt tot funcții) n -ară (adică de aritate n , cu n argumente), iar arg_1, \dots, arg_n sunt **termeni**.

Observație (predicate, i.e. cu valori booleene, nu orice fel de funcții)

Faptele constau din predicate.

Regulile sunt formate dintr-un predicat urmat de simbolul `:-`, urmat de alte predicate unite prin conectori logici.

Interogările sunt formate din predicate prin conectori logici.

Observație (imbricarea)

Sintaxa Prolog permite imbricarea predicatelor cu operații arbitrare, dar imbricarea doar construiește termeni, nu produce, de exemplu, calcule aritmetice în subtermeni (a se vedea predicatele e , f din *exlisteetc.pl*).

Operatori Prolog și predicate predefinite în Prolog

Exemplu

Nu putem interoga (pentru că nu este predicat: rezultatul e aritmetic, nu boolean):

?- -((10-1)*2-3)/5.

dar putem interoga:

?- X is -((10-1)*2-3)/5.

Alți operatori aritmetici:	//	sau <i>div</i>	câtul împărțirii întregi
		<i>mod</i>	restul împărțirii întregi
		**	ridicare la putere

A se vedea, în fișierul *test.pl* din prima lecție de laborator, predicatul **not** sau `\+` și operatorii de comparație între valori aritmetice și de testare a egalității ca expresii și ca valori aritmetice.

Predicatul *zeroar* (i.e. de aritate 0, i.e. fără argumente) **fail** este întotdeauna evaluat la **FALS**.

(Predicate unare (i.e. de aritate 1, i.e. cu câte un singur argument))

Predicatele **number**, **integer**, **float** testează dacă argumentul lor este număr, respectiv număr întreg, respectiv număr real (în format float).

Predicatele predefinite **var**, **nonvar**, **atom**, **atomic** testează dacă argumentul lor este variabilă, respectiv non-variabilă, respectiv atom, respectiv atom sau număr sau lista vidă [].

```
?- var(A).  
true.
```

```
?- nonvar(A).  
false.
```

```
?- var(_).  
true.
```

```
?- nonvar(_).  
false.
```

```
?- var('A').  
false.
```

```
?- nonvar('A').  
true.
```

```
?- var(a).  
false.
```

```
?- nonvar(a).  
true.
```

```
?- var(1).  
false.
```

```
?- nonvar(1).  
true.
```

```
?- var(p(X)).  
false.
```

```
?- nonvar(p(X)).  
true.
```

```
?- var(-->).  
false.
```

```
?- nonvar(-->).  
true.
```

```
?- atom(a).  
true.
```

```
?- atom(alta_constanta).  
true.
```

```
?- atom(-#-->).  
true.
```

```
?- atom(1).  
false.
```

```
?- atom([]).  
false.
```

```
?- atom(p(X)).  
false.
```

```
?- atom(X).  
false.
```

```
?- atomic(a).  
true.
```

```
?- atomic(alta_constanta).  
true.
```

```
?- atomic(-#-->).  
true.
```

```
?- atomic(1).  
true.
```

```
?- atomic([]).  
true.
```

```
?- atomic(p(X)).  
false.
```

```
?- atomic(X).  
false.
```

```
?- number(a).  
false.
```

```
?- number(10+2).  
false.
```

```
?- number(12).  
true.
```

```
?- number(12.5e2).  
true.
```

```
?- integer(5).  
true.
```

```
?- integer(-5).  
true.
```

```
?- integer(5.0).  
false.
```

```
?- float(5).  
false.
```

```
?- float(5.0).  
true.
```

```
?- float(-5.0e2).  
true.
```

```
?- float(-5e2).  
true.
```

```
?- float(5e0).  
true.
```

```
?- float(-15.25e-2).  
true.
```

```
?- float(-15.25E-2).  
true.
```

Pentru a răspunde la o întrebare, Prologul încearcă să UNIFICE **scopurile** din **întrebare** cu predicatele din:

- **fapte**;
- **membrii stângi ai regulilor** – dacă o astfel de unificare reușește, atunci **următorul scop** pe care va încerca să-l satisfacă este membrul drept al aceleiași reguli, cu argumentele rezultate în urma unificării.

(UNIFICARE (orientativ) – detalii într-un curs următor)

Două funcții UNIFICĂ dacă:

$$\left\{ \begin{array}{l} \text{acele funcții coincid și} \\ \text{au aceleași argumente:} \end{array} \right. \quad f(arg_1, \dots, arg_n) = g(a_1, \dots, a_k) \iff \left\{ \begin{array}{l} f = g (\implies n = k) \text{ și} \\ arg_1 = a_1, \\ \vdots \\ arg_n = a_n. \end{array} \right.$$

Mai precis: două funcții UNIFICĂ dacă: $\left\{ \begin{array}{l} \text{acele funcții coincid și} \\ \text{RECURSIV, argumentele lor UNIFICĂ.} \end{array} \right.$

RECURSIA merge până la **termeni fără argumente**, adică **variabile** sau **constante**:

- **constantele** sunt **operații** (anume exact **operațiile fără argumente** – vom vedea), așadar, conform regulii de mai sus, **NU UNIFICĂ DECÂT cu ele însele**;
- **variabilele** UNIFICĂ, CU **orice termen care nu le conține**; (desigur, o unificare de tipul $X = h(X)$ ar conduce la $X = h(X) = h(h(X)) = h(h(X)) = \dots$, deci nu ar fi corectă.)

Exemplu (lista vidă nu unifică, cu nicio listă nevidă)

Lista vidă $[]$ e o constantă, i.e. operație fără argumente (vom vedea), iar o listă nevidă $[H|T]$ (care conține măcar elementul H) este termenul format din operația binară $[|]$ cu argumentele H și T ($[|](H, T)$). Cum operațiile $[]$ și $[|]$ nu coincid, rezultă că $[]$ nu unifică, cu $[H|T]$.

Exemplu (utilitate *var*, *nonvar*)

Să numărăm aparițiile unui element într-o listă, cu un predicat ternar $numar_aparitii(Element, Lista, NrAparitii)$.

Următoarea implementare numără, de fapt, elementele listei cu care unifică acel element:

$numar_aparitii(-, [], 0)$.

$numar_aparitii(H, [H|T], N) :- numar_aparitii(H, T, K), N \text{ is } K + 1$.

$numar_aparitii(X, [H|T], N) :- X \neq H, numar_aparitii(X, T, N)$.

Cu implementarea de mai sus, iată răspunsul Prologului la această interogare:

```
?- numar_aparitii(a,[X,a,1,a,a,2,3,Y,a,4,Z],N).  
X = Y, Y = Z, Z = a,  
N = 7 ;  
false.
```

Am cerut mai multe soluții, cu ";", și răspunsul Prologului a fost "false", așadar acesta e singurul răspuns găsit de Prolog la interogarea de mai sus.

În schimb, dacă implementăm astfel predicatul de mai sus:

```
numar_aparitii(_,[],0).  
numar_aparitii(X,[H|T],N):- var(H), numar_aparitii(X,T,N).  
numar_aparitii(X,[H|T],N):- nonvar(H), X=H, numar_aparitii(H,T,K), N is  
K + 1.  
numar_aparitii(X,[H|T],N):- X\=H, numar_aparitii(X,T,N).
```

atunci răspunsul unic găsit de Prolog la aceeași interogare este:

```
?- numar_aparitii(a,[X,a,1,a,a,2,3,Y,a,4,Z],N).  
N = 4 ;  
false.
```

A se vedea, în fișierul *.pl* și lecțiile video screenshot atașate, scrierea unui predicat care determină exact elementele listei **identice** cu acel element.

Cum răspunde Prologul la interogări?

Considerăm baza de cunoștințe:

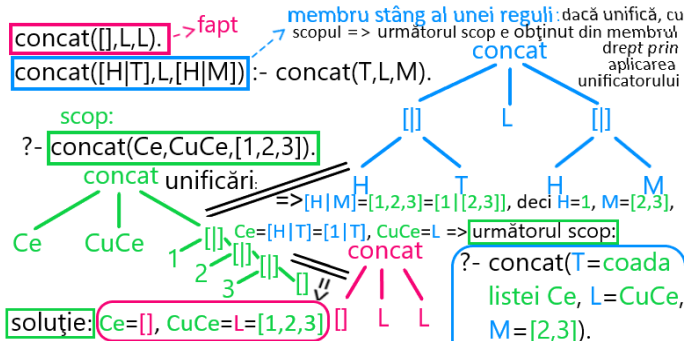
```
concat([], L, L).
```

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

Să vedem cum răspunde Prologul la interogarea:

```
?- concat(Ce, CuCe, [1,2,3]).
```

Procedeul de rezolvare a acestei interogări este cel descris mai sus:



Vom vedea într-un curs următor ce este un *unificator*.

Dacă dăm *trace* și apoi interogarea:

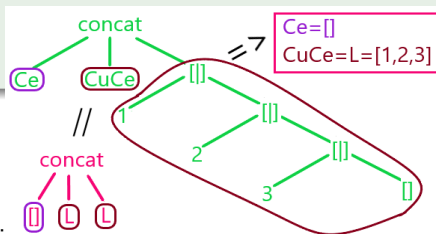
?- concat(Ce,CuCe,[a,b,c]).

observăm din pașii afișați că primul lucru executat de Prolog este redenumirea variabilelor din interogare în nume de forma *_număr*.

Dacă în baza de cunoștințe, avem și definițiile altor predicate, de exemplu definiția de mai sus a predicatului lung, atunci:

cum `concat` \neq `lung`, scopul `concat(Ce, CuCe, [a, b, c])` nu unifică:

- nici cu `lung([], 0)`,
- nici cu `lung([H|T], N)`.



În schimb, `concat(Ce,CuCe,[1,2,3])` unifică:

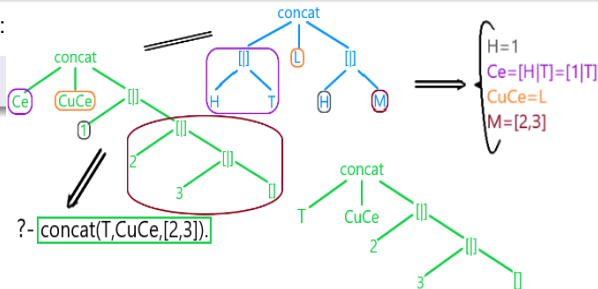
cu `concat([], L, L)`, pentru **Ce = []** și **CuCe = L = [1, 2, 3]** – aceasta este *prima soluție* a interogării,

dar și:

cu $\text{concat}([H|T], L, [H|M])$, pentru $H = 1$, $Ce = [1|T]$, $CuCe = L$ și $M = [2, 3]$,

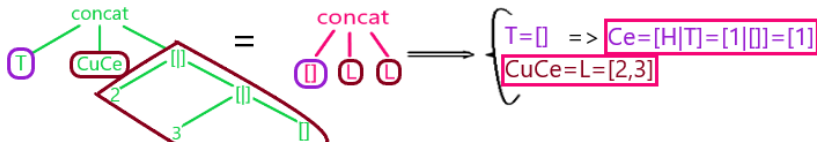
așadar următorul scop este:

?- $\text{concat}(T, CuCe, [2,3])$.



Acesta unifică:

cu $\text{concat}([], L, L)$, pentru $T = []$ și $CuCe = L = [2, 3]$ – așadar a doua soluție a interogării este: $Ce = [1|T] = [1|[]] = [1]$ și $CuCe = [2, 3]$,

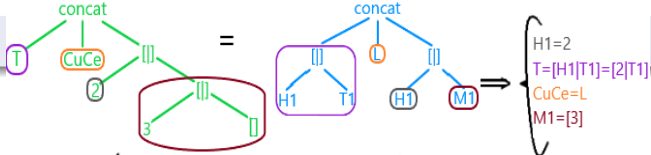


dar și:

cu $\text{concat}([H1|T1], L, [H1|M1])$, pentru $H1 = 2$, $T = [2|T1]$, $CuCe = L$ și $M1 = [3]$,

așadar următorul scop este:

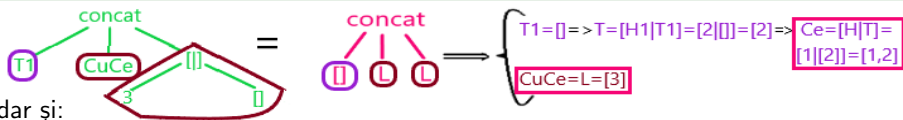
?- concat(T1,CuCe,[3]).



Acesta unifică:

?- concat(T1,CuCe,[3]).

cu `concat([], L, L)`, pentru $T1 = []$ și $CuCe = L = [3]$ – așadar a treia soluție a interogării este: $Ce = [1|T] = [1|[2|T1]] = [1|[2|[]]] = [1, 2]$ și $CuCe = [3]$,

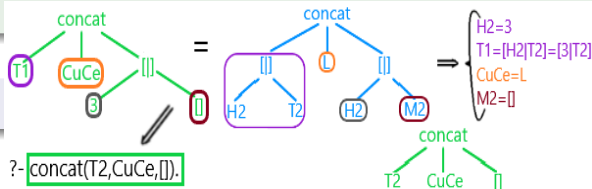


dar și:

cu `concat([H2|T2], L, [H2|M2])`, pentru $H2 = 3$, $T1 = [3|T2]$, $CuCe = L$ și $M2 = []$,

așadar următorul scop este:

?- concat(T2,CuCe,[]).



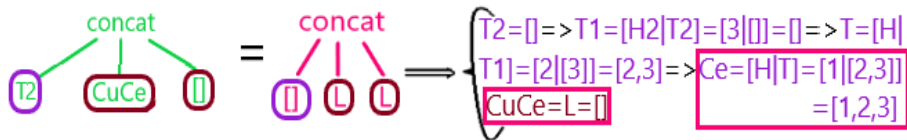
Întrucât:

constanta $[]$ nu unifică, cu termenul $[H3|M3]$, având operația dominantă $[|]$,
avem:

$\text{concat}(T2, CuCe, [])$ nu unifică, cu $\text{concat}([H3|T3], L, [H3|M3])$.

Prin urmare:

$\text{concat}(T2, CuCe, [])$ unifică numai cu $\text{concat}([], L, L)$, pentru **$T2 = []$** și **$CuCe = L = []$** – așadar *a patra și ultima soluție* a interogării este:
 $Ce = [1|T] = [1|[2|T1]] = [1|[2|[3|T2]]] = [1|[2|[3|[]]]] = [1, 2, 3]$ și **$CuCe = []$** .



În cazul unui scop compus, de exemplu:

?- $\text{concat}(A, [2,3], [1,2,3]), \text{concat}(A, [4,5], C)$.

toate scopurile trebuie satisfăcute, cu **aceleași valori** pentru *variabilele comune*.
Pentru exemplul de mai sus, *unica soluție*: **$A = [1]$** , **$C = [1, 4, 5]$** .

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor**
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

Cuantificatorii și simbolul \exists !

Cuantificatorii:

- *cuantificatorul universal*: \forall
- *cuantificatorul existențial*: \exists

Notăție

Alăturarea de simboluri \exists ! semnifică “există un unic”, “există și este unic”.

Observație

\exists ! nu este un cuantificator, ci este o notație prescurtată pentru enunțuri compuse: dacă x este o variabilă, iar $p(x)$ este o proprietate asupra lui x , atunci scrierea $(\exists! x)(p(x))$ este o abreviere pentru enunțul scris, desfășurat, astfel:

$$(\exists x)(p(x)) \text{ și } (\forall y)(\forall z)[(p(y) \text{ și } p(z)) \Rightarrow y = z],$$

unde y și z sunt variabile.

(negarea enunțurilor cuantificate)

Dacă x este o variabilă, iar $p(x)$ este o proprietate referitoare la x (mai precis o proprietate referitoare la elementele pe care le parcurge/le poate denumi x), atunci:

- $\text{non } [(\forall x)(p(x))] \Leftrightarrow (\exists x)(\text{non } p(x))$
- $\text{non } [(\exists x)(p(x))] \Leftrightarrow (\forall x)(\text{non } p(x))$

Negarea enunțurilor cuantificate

Cum se neagă un enunț cu mai mulți cuantificatori? Aplicând proprietățile de mai sus, și iterând acest procedeu:

Exemplu

Fie x, y, z, t, u variabile, iar $p(x, y, z, t, u)$ o proprietate depinzând de x, y, z, t, u . Atunci:

$$\begin{aligned} \text{non } [(\forall x) (\forall y) (\exists z) (\forall t) (\exists u) (p(x, y, z, t, u))] &\Leftrightarrow \\ (\exists x) [\text{non } [(\forall y) (\exists z) (\forall t) (\exists u) (p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x) (\exists y) [\text{non } [(\exists z) (\forall t) (\exists u) (p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x) (\exists y) (\forall z) [\text{non } [(\forall t) (\exists u) (p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x) (\exists y) (\forall z) (\exists t) [\text{non } [(\exists u) (p(x, y, z, t, u))]] &\Leftrightarrow \\ (\exists x) (\exists y) (\forall z) (\exists t) (\forall u) (\text{non } p(x, y, z, t, u)) &\end{aligned}$$

Nu vom mai aplica acest procedeu pas cu pas. Reținem că procedeul constă în transformarea fiecărui cuantificator universal într-unul existențial și invers, și negarea proprietății de sub acești cuantificatori.

Cuantificatori aplicați fixând un domeniu al valorilor

În scrierile de mai sus, domeniile în care pot lua valori variabilele sunt neprecizate, putând fi considerate, de exemplu, *universul discuției* (i.e. colecția tuturor elementelor/obiectelor cu care se lucrează în problema curentă):

- $\forall x$ semnifică: oricare ar fi x "de oriunde" sau "din universul discuției";
- $\exists x$ semnifică: există x "de oriunde" sau "în universul discuției".

Fie M o mulțime, x o variabilă, iar $p(x)$ o proprietate referitoare la elementele lui M . Atunci următoarele scrieri sunt abrevieri pentru scrierile fără domeniu al valorilor lângă cuantificatori:

- $(\forall x \in M)(p(x)) \stackrel{\text{not.}}{\Leftrightarrow} (\forall x)(x \in M \Rightarrow p(x))$
- $(\exists x \in M)(p(x)) \stackrel{\text{not.}}{\Leftrightarrow} (\exists x)(x \in M \text{ și } p(x))$

Toate proprietățile logice pentru enunțuri cuantificate, inclusiv negarea enunțurilor cuantificate de mai sus și comutarea cuantificatorilor de mai jos, se scriu la fel și sunt valabile și pentru cuantificatori urmați de un domeniu al valorilor pentru variabila cuantificată.

Cuantificatorii de același fel comută, cei diferiți nu

Fie x și y variabile, iar $p(x, y)$ o proprietate asupra lui x și y . Atunci:

- $(\forall x) (\forall y) (p(x, y)) \Leftrightarrow (\forall y) (\forall x) (p(x, y))$
- $(\exists x) (\exists y) (p(x, y)) \Leftrightarrow (\exists y) (\exists x) (p(x, y))$
- $(\forall x) (\exists y) (p(x, y)) \not\Leftrightarrow (\exists y) (\forall x) (p(x, y))$ (pentru fiecare valoare a lui x , valoarea lui y pentru care e satisfăcut enunțul din stânga depinde de valoarea lui x)

Analog, dacă A și B sunt mulțimi, avem:

- $(\forall x \in A) (\forall y \in B) (p(x, y)) \Leftrightarrow (\forall y \in B) (\forall x \in A) (p(x, y))$
- $(\exists x \in A) (\exists y \in B) (p(x, y)) \Leftrightarrow (\exists y \in B) (\exists x \in A) (p(x, y))$
- $(\forall x \in A) (\exists y \in B) (p(x, y)) \not\Leftrightarrow (\exists y \in B) (\forall x \in A) (p(x, y))$

Desigur, la fel pentru cazul în care doar unul dintre cuantificatori este aplicat cu un domeniu al valorilor pentru variabila cuantificată:

$$(\forall x) (\forall y \in B) (p(x, y)) \Leftrightarrow (\forall y \in B) (\forall x) (p(x, y)) \text{ etc..}$$

Exemplu

Enunțul $(\forall x \in \mathbb{N}) (\exists y \in \mathbb{Z}) (x + y = 0)$ este adevărat.

Enunțul $(\exists y \in \mathbb{Z}) (\forall x \in \mathbb{N}) (x + y = 0)$ este fals.

Cuantificatori, disjuncții și conjuncții logice

Să observăm și următoarele proprietăți logice: dacă x este o variabilă, iar $p(x)$ și $q(x)$ sunt enunțuri referitoare la x , atunci:

- $(\forall x)(p(x) \text{ și } q(x)) \Leftrightarrow (\forall x)(p(x)) \text{ și } (\forall x)(q(x))$
- $(\exists x)(p(x) \text{ sau } q(x)) \Leftrightarrow (\exists x)(p(x)) \text{ sau } (\exists x)(q(x))$
- $(\forall x)(p(x) \text{ sau } q(x)) \not\Leftrightarrow (\forall x)(p(x)) \text{ sau } (\forall x)(q(x))$
- $(\exists x)(p(x) \text{ și } q(x)) \not\Leftrightarrow (\exists x)(p(x)) \text{ și } (\exists x)(q(x))$

Exemplu

Enunțul $(\forall x \in \mathbb{N})(2 \mid x \text{ sau } 2 \nmid x)$ este adevărat.

Enunțul $(\forall x \in \mathbb{N})(2 \mid x)$ este fals. Enunțul $(\forall x \in \mathbb{N})(2 \nmid x)$ este tot fals. Prin urmare, enunțul $[(\forall x \in \mathbb{N})(2 \mid x) \text{ sau } (\forall x \in \mathbb{N})(2 \nmid x)]$ este fals.

Exemplu

Enunțul $(\exists x \in \mathbb{R})(x < 0 \text{ și } x \geq 10)$ este fals.

Enunțul $(\exists x \in \mathbb{R})(x < 0)$ este adevărat. Enunțul $(\exists x \in \mathbb{R})(x \geq 10)$ este tot adevărat. Prin urmare, enunțul $[(\exists x \in \mathbb{R})(x < 0) \text{ și } (\exists x \in \mathbb{R})(x \geq 10)]$ este adevărat.

Scoaterea de sub un cuantificator a unui enunț care nu depinde de variabila cuantificată

Dacă, în enunțurile compuse cuantificate de mai sus, în locul lui $q(x)$ avem un enunț q care nu depinde de x , atunci:

$$(\forall x) q \Leftrightarrow q \Leftrightarrow (\exists x) q,$$

așadar, în acest caz, din proprietățile anterioare obținem:

- $(\forall x) (p(x) \text{ și } q) \Leftrightarrow (\forall x) (p(x)) \text{ și } q$
- $(\exists x) (p(x) \text{ sau } q) \Leftrightarrow (\exists x) (p(x)) \text{ sau } q$

dar au loc și:

- ① $(\forall x) (p(x) \text{ sau } q) \Leftrightarrow (\forall x) (p(x)) \text{ sau } q$
- ② $(\exists x) (p(x) \text{ și } q) \Leftrightarrow (\exists x) (p(x)) \text{ și } q$

după cum se poate verifica foarte ușor analizând cazurile: $\begin{cases} q \text{ e falsă} \\ \text{și} \\ q \text{ e adevărată.} \end{cases}$

Putem observa și că oricare dintre proprietățile ① și ② se deduce din cealaltă folosind modalitatea în care se neagă enunțurile cuantificate și proprietățile:

- $\text{non}(p \text{ și } q) \Leftrightarrow (\text{non } p) \text{ sau } (\text{non } q)$
- și
- $\text{non}(p \text{ sau } q) \Leftrightarrow (\text{non } p) \text{ și } (\text{non } q)$

pentru orice enunțuri p și q .

Mnemonic despre proprietățile cuantificatorilor

Scrieri echivalente ale enunțurilor din exemplele anterioare, fără domeniu al valorilor după cuantificatori:

$$\begin{aligned}(\forall x \in \mathbb{N}) (2 \mid x \text{ sau } 2 \nmid x) &\Leftrightarrow \\(\forall x) [x \in \mathbb{N} \Rightarrow (2 \mid x \text{ sau } 2 \nmid x)] &\Leftrightarrow \\(\forall x) [x \in \mathbb{N} \Rightarrow (2 \mid x \text{ sau } 2 \nmid x)] &\Leftrightarrow \\(\forall x) [(x \in \mathbb{N} \Rightarrow 2 \mid x) \text{ sau } (x \in \mathbb{N} \Rightarrow 2 \nmid x)];\end{aligned}$$

$$\begin{aligned}[(\forall x \in \mathbb{N}) (2 \mid x) \text{ sau } (\forall x \in \mathbb{N}) (2 \nmid x)] &\Leftrightarrow \\[(\forall x) (x \in \mathbb{N} \Rightarrow 2 \mid x) \text{ sau } (\forall x) (x \in \mathbb{N} \Rightarrow 2 \nmid x)];\end{aligned}$$

$$\begin{aligned}(\exists x \in \mathbb{R}) (x < 0 \text{ și } x \geq 10) &\Leftrightarrow \\(\exists x) (x \in \mathbb{R} \text{ și } x < 0 \text{ și } x \geq 10) &\Leftrightarrow \\(\exists x) [(x \in \mathbb{R} \text{ și } x < 0) \text{ și } (x \in \mathbb{R} \text{ și } x \geq 10)];\end{aligned}$$

$$\begin{aligned}[(\exists x \in \mathbb{R}) (x < 0) \text{ și } (\exists x \in \mathbb{R}) (x \geq 10)] &\Leftrightarrow \\[(\exists x) (x \in \mathbb{R} \text{ și } x < 0) \text{ și } (\exists x) (x \in \mathbb{R} \text{ și } x \geq 10)].\end{aligned}$$

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

Definiția unei funcții

Definiție

Fie A și B mulțimi oarecare. Se numește *funcție* de la A la B un triplet $f := (A, G, B)$, unde $G \subseteq A \times B$, a. î., pentru orice $a \in A$, există un unic $b \in B$, cu proprietatea că $(a, b) \in G$.

Formal: $(\forall a \in A) (\exists ! b \in B) ((a, b) \in G)$. Scris desfășurat:

$$(\forall a) [a \in A \Rightarrow [(\exists b) (b \in B \text{ și } (a, b) \in G) \text{ și} \\ (\forall c) (\forall d) [(c \in B \text{ și } d \in B \text{ și } (a, c) \in G \text{ și } (a, d) \in G) \Rightarrow c = d]]].$$

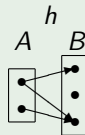
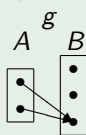
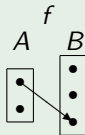
Faptul că f este o funcție de la A la B se notează cu $f : A \rightarrow B$ sau $A \xrightarrow{f} B$.

Mulțimea A se numește *domeniul* funcției f , B se numește *codomeniul* sau *domeniul valorilor* lui f , iar G se numește *graficul* lui f .

Pentru fiecare $a \in A$, unicul $b \in B$ cu proprietatea că $(a, b) \in G$ se notează cu $f(a)$ și se numește *valoarea funcției f în punctul a* .

Exemplu

Care dintre următoarele corespondențe este o funcție de la A la B ?



Egalitatea a două funcții

Remarcă $((a, b) \in G \Leftrightarrow f(a) = b)$

Dacă $f = (A, G, B)$ este o funcție ($f : A \rightarrow B$), atunci graficul G al lui f este mulțimea de perechi: $G = \{(a, f(a)) \mid a \in A\} \subseteq A \times B$.

Definiție

Fie $f = (A, F, B)$ și $g = (C, G, D)$ două funcții ($f : A \rightarrow B$, iar $g : C \rightarrow D$).

- Egalitatea $f = g$ semnifică egalitatea de triplete $(A, F, B) = (C, G, D)$, i. e. spunem că $f = g$ ddacă:

$A = C$ (are loc egalitatea domeniilor),
 $B = D$ (are loc egalitatea codomeniilor) și
 $F = G$ (are loc egalitatea graficelor celor două funcții,
ceea ce, conform scrierii acestor grafice
din remarca anterioară, se transcrie în egalitate
punctuală, adică egalitate în fiecare punct:
pentru orice $a \in A = C$, $f(a) = g(a)$).

- Dacă X este o mulțime a. î. $X \subseteq A$ și $X \subseteq C$, atunci spunem că f și g coincid pe X ddacă f și g au aceleași valori în elementele lui X , adică: oricare ar fi $x \in X$, $f(x) = g(x) \in B \cap D$.

Există o unică funcție de la \emptyset la o mulțime arbitrară

Notăție (putere de mulțimi)

Pentru orice mulțimi A și B , se notează cu B^A mulțimea funcțiilor de la A la B :

$$B^A = \{f \mid f : A \rightarrow B\}$$

Remarcă (pentru orice B , B^\emptyset are exact un element)

Fie B o mulțime oarecare (**poate fi vidă și poate fi nevidă**). Atunci există o unică funcție $f : \emptyset \rightarrow B$.

Într-adevăr, o funcție $f : \emptyset \rightarrow B$ trebuie să fie un triplet $f = (\emptyset, G, B)$, cu $G \subseteq \emptyset \times B = \emptyset$, deci $G = \emptyset$. Așadar, există cel mult o funcție $f : \emptyset \rightarrow B$, anume $f = (\emptyset, \emptyset, B)$ este unica posibilitate. Să arătăm că acest triplet satisface definiția funcției:

$$(\forall a \in \emptyset) (\exists ! b \in B) ((a, b) \in \emptyset), \text{ i. e.:}$$

$$(\forall a) [a \in \emptyset \Rightarrow (\exists ! b) (b \in B \text{ și } (a, b) \in \emptyset)].$$

Pentru orice element a , proprietatea $a \in \emptyset$ este falsă, așadar, pentru orice element a , implicația $[a \in \emptyset \Rightarrow \dots]$ este adevărată. Iar acest lucru înseamnă exact faptul că întreaga proprietate $(\forall a)[a \in \emptyset \Rightarrow \dots]$ este adevărată, deci f este funcție. Prin urmare, există o unică funcție $f : \emptyset \rightarrow B$, anume $f = (\emptyset, \emptyset, B)$.

Nu există nicio funcție de la o mulțime nevidă la \emptyset

Remarcă (pentru orice $A \neq \emptyset$, $\emptyset^A = \emptyset$)

Fie A o mulțime **nevidă** (i. e. $A \neq \emptyset$). Atunci nu există nicio funcție $f : A \rightarrow \emptyset$. Într-adevăr, o funcție $f : A \rightarrow \emptyset$ trebuie să fie un triplet $f = (A, G, \emptyset)$, cu $G \subseteq A \times \emptyset = \emptyset$, deci $G = \emptyset$. Așadar, dacă ar exista o funcție $f : A \rightarrow \emptyset$, atunci am avea neapărat $f = (A, \emptyset, \emptyset)$. Să vedem dacă acest triplet verifică definiția funcției:

$$(\forall a \in A) (\exists ! b \in \emptyset) ((a, b) \in \emptyset), \quad \text{i. e. :}$$

$$(\forall a) [a \in A \Rightarrow [(\exists b) (b \in \emptyset \text{ și } (a, b) \in \emptyset) \text{ și}$$

$$(\forall c) (\forall d) [(c \in \emptyset \text{ și } d \in \emptyset \text{ și } (a, c) \in \emptyset \text{ și } (a, d) \in \emptyset) \Rightarrow c = d]]].$$

Oricare ar fi elementul b , proprietatea $b \in \emptyset$ este falsă, deci, oricare ar fi elementele a și b , conjuncția $(b \in \emptyset \text{ și } (a, b) \in \emptyset)$ este falsă, deci, oricare ar fi elementul a , proprietatea $(\exists b) (b \in \emptyset \text{ și } (a, b) \in \emptyset)$ este falsă, așadar, oricare ar fi elementul a , conjuncția care succede mai sus implicației având ca antecedent pe $a \in A$ este falsă. În schimb, întrucât A este nevidă, rezultă că proprietatea $a \in A$ este adevărată pentru măcar un element a . Prin urmare, implicația $[a \in A \Rightarrow \dots]$ de mai sus este falsă pentru cel puțin un element a , ceea ce înseamnă că întreaga proprietate $(\forall a) [a \in A \Rightarrow \dots]$ este falsă, și deci f nu este funcție.

Imaginea și preimagea printr-o funcție

Definiție

Pentru orice mulțimi A și B , orice funcție $f : A \rightarrow B$ și orice submulțimi $X \subseteq A$ și $Y \subseteq B$, se definesc:

- *imagea lui X prin f sau imagea directă a lui X prin f* , notată $f(X)$, este submulțimea lui B :

$$f(X) = \{f(x) \mid x \in X\} \subseteq B$$

- $f(A)$ se mai notează cu $Im(f)$ și se numește *imagea lui f* :

$$Im(f) = f(A) = \{f(a) \mid a \in A\} \subseteq B$$

- *preimagea lui Y prin f sau imagea inversă a lui Y prin f* , notată $f^{-1}(Y)$ ($f^*(Y)$ în unele cărți, pentru a o deosebi de imagea lui Y prin inversa f^{-1} a lui f , care există numai atunci când f este inversabilă, deci numai atunci când f este bijectivă – a se vedea în cele ce urmează –, pe când preimagea unei submulțimi a codomeniului poate fi definită pentru orice funcție), este submulțimea lui A :

$$f^{-1}(Y) = \{x \in A \mid f(x) \in Y\} \subseteq A$$

Funcții injective, surjective, bijective

Definiție

Fie A și B mulțimi și $f : A \rightarrow B$ o funcție. f se zice:

- *injectivă* ddacă are loc oricare dintre următoarele condiții echivalente:
 - pentru orice $b \in B$, există cel mult un $a \in A$, astfel încât $f(a) = b$
 - pentru orice $a_1, a_2 \in A$, dacă $a_1 \neq a_2$, atunci $f(a_1) \neq f(a_2)$
 - pentru orice $a_1, a_2 \in A$, dacă $f(a_1) = f(a_2)$, atunci $a_1 = a_2$
- *surjectivă* ddacă are loc oricare dintre următoarele condiții echivalente:
 - pentru orice $b \in B$, există (cel puțin un) $a \in A$, astfel încât $f(a) = b$ (formal: $(\forall b \in B) (\exists a \in A) (f(a) = b)$)
 - $f(A) = B$
- *bijectivă* ddacă are loc oricare dintre următoarele condiții echivalente:
 - f este simultan injectivă și surjectivă
 - pentru orice $b \in B$, există exact un $a \in A$, astfel încât $f(a) = b$ (formal: $(\forall b \in B) (\exists ! a \in A) (f(a) = b)$)

Funcțiile injective, surjective, respectiv bijective se mai numesc *injecții*, *surjecții*, respectiv *bijecții*.

- 1 Introducere
- 2 Inițiere în programarea în Prolog
- 3 Mnemonic despre proprietățile cuantificatorilor
- 4 Mnemonic despre funcții
- 5 Constantele sunt operații fără argumente

Familii arbitrare de elemente, familii arbitrare de mulțimi

- Ce este un șir de numere reale indexat de \mathbb{N} ? Un *șir* $(x_n)_{n \in \mathbb{N}} \subset \mathbb{R}$ este o funcție $f : \mathbb{N} \rightarrow \mathbb{R}$. Pentru orice $n \in \mathbb{N}$, se notează $x_n := f(n) \in \mathbb{R}$.
- Ce este o familie arbitrară de numere reale? Fie I o mulțime arbitrară. Ce este o familie de numere reale indexată de I ? O *familie* $(x_i)_{i \in I} \subseteq \mathbb{R}$ este o funcție $f : I \rightarrow \mathbb{R}$. Pentru orice $i \in I$, se notează $x_i := f(i) \in \mathbb{R}$. Elementele mulțimii I se numesc *indicii* familiei $(x_i)_{i \in I}$.

Dată o mulțime arbitrară A :

- ce este un șir de elemente ale lui A indexat de \mathbb{N} ?
- ce este o familie arbitrară de elemente ale lui A ?

Înlocuind mai sus pe \mathbb{R} cu A , se obțin definițiile acestor noțiuni. Să reținem:

Definiție (familie de elemente ale lui A indexată de I : $(x_i)_{i \in I} \subseteq A$)

Fie I și A mulțimi arbitrare.

O *familie de elemente ale lui A indexată de I* este o funcție $f : I \rightarrow A$. Pentru orice $i \in I$, se notează $x_i := f(i) \in A$, astfel că familia f se mai notează sub forma $(x_i)_{i \in I} \subseteq A$.

Elementele mulțimii I se numesc *indicii* familiei $(x_i)_{i \in I}$.

- Ce este un șir de mulțimi indexat de \mathbb{N} ?
- Ce este o familie arbitrară de mulțimi?

Egalitatea între familii de elemente

- Există o singură familie vidă de elemente ale unei mulțimi arbitrare A , pentru că există o singură funcție de la \emptyset la A (a se vedea și mai jos).
- Familia vidă nu este egală cu nicio familie nevidă, ci este egală doar cu ea însăși.

Fie I și A două mulțimi nevide, iar $(a_i)_{i \in I}$ și $(b_i)_{i \in I}$ două familii de elemente din A indexate de I :

- $I \neq \emptyset, A \neq \emptyset$
- $(a_i)_{i \in I} \subseteq A$, i. e., pentru orice $i \in I, a_i \in A$
- $(b_i)_{i \in I} \subseteq A$, i. e., pentru orice $i \in I, b_i \in A$

Conform celor menționate anterior, familiile de elemente din A indexate de I sunt, prin definiție, funcții de la I la A , iar notația lor ca mai sus se adoptă pentru comoditate:

- $(a_i)_{i \in I} = f : I \rightarrow A$, unde, pentru orice $i \in I, f(i) = a_i$
- $(b_i)_{i \in I} = g : I \rightarrow A$, unde, pentru orice $i \in I, g(i) = b_i$

Egalitatea între familii de elemente

Egalitatea de funcții semnifică egalitatea domeniilor și a codomeniilor (valabilă pentru f și g , pentru că au ambele domeniul I și codomeniul A) și *egalitatea punctuală*, adică egalitatea în fiecare punct: două funcții cu același domeniu și același codomeniu sunt egale dacă sunt egale în fiecare punct al domeniului lor comun:

$$f = g \quad \text{dacă, pentru orice } i \in I, f(i) = g(i).$$

Deci ce semnifică egalitatea a două familii de elemente din aceeași mulțime indexate de aceeași mulțime? *Egalitatea pe componente*: cele două familii sunt egale dacă sunt egale pe componente:

$$(a_i)_{i \in I} = (b_i)_{i \in I} \quad \text{dacă, pentru orice } i \in I, a_i = b_i.$$

- **Caz particular:** cazul finit nevid: $I = \overline{1, n}$, cu n natural nenul:

$$(a_1, a_2, \dots, a_n) = (b_1, b_2, \dots, b_n) \quad \text{dacă} \quad \begin{cases} a_1 = b_1, \\ a_2 = b_2, \\ \vdots \\ a_n = b_n. \end{cases}$$

Familii arbitrare de mulțimi

Definiție

Fie T o mulțime arbitrară. Se numește *șir de submulțimi ale lui T indexat de \mathbb{N}* o funcție $f : \mathbb{N} \rightarrow \mathcal{P}(T)$. Pentru fiecare $n \in \mathbb{N}$, se notează $A_n := f(n) \in \mathcal{P}(T)$, iar șirul de submulțimi ale lui T se notează cu $(A_n)_{n \in \mathbb{N}}$. Scriem $(A_n)_{n \in \mathbb{N}} \subseteq \mathcal{P}(T)$ cu semnificația că, pentru fiecare $n \in \mathbb{N}$, $A_n \in \mathcal{P}(T)$.

Definiție

Fie T și I două mulțimi arbitrare. Se numește *familie de submulțimi ale lui T indexată de I* o funcție $f : I \rightarrow \mathcal{P}(T)$. Pentru fiecare $i \in I$, se notează $A_i := f(i) \in \mathcal{P}(T)$, iar familia de submulțimi ale lui T se notează cu $(A_i)_{i \in I}$. Scriem $(A_i)_{i \in I} \subseteq \mathcal{P}(T)$ cu semnificația că, pentru fiecare $i \in I$, $A_i \in \mathcal{P}(T)$. Elementele mulțimii I se numesc *indicii* familiei $(A_i)_{i \in I}$.

Putem generaliza definițiile anterioare la șiruri de mulțimi oarecare și familii de mulțimi oarecare, nu neapărat părți ale unei mulțimi precizate, dar vom avea nevoie de acea definiție mai cuprinzătoare a noțiunii de funcție, care permite unei funcții f definite pe \mathbb{N} , respectiv pe I , să aibă drept codomeniu o clasă (nu neapărat o mulțime), anume clasa tuturor mulțimilor în acest caz.

Definiție

Fie I o mulțime arbitrară și $(A_i)_{i \in I}$ o familie de mulțimi (părți ale unei mulțimi T sau mulțimi arbitrare) indexată de I .

Se definesc următoarele operații:

- *reuniunea familiei* $(A_i)_{i \in I}$ este mulțimea notată $\bigcup_{i \in I} A_i$ și definită prin:

$$\bigcup_{i \in I} A_i = \{x \mid (\exists i \in I) (x \in A_i)\} = \{x \mid (\exists i) (i \in I \text{ și } x \in A_i)\}$$

- *intersecția familiei* $(A_i)_{i \in I}$ este mulțimea notată $\bigcap_{i \in I} A_i$ și definită prin:

$$\bigcap_{i \in I} A_i = \{x \mid (\forall i \in I) (x \in A_i)\} = \{x \mid (\forall i) (i \in I \Rightarrow x \in A_i)\}$$

Operații cu familii arbitrare de mulțimi

Definiție (continuare)

- *produsul cartezian al familiei* $(A_i)_{i \in I}$ (numit și *produsul direct al familiei* $(A_i)_{i \in I}$) este mulțimea notată $\prod_{i \in I} A_i$ și definită prin:

$$\begin{aligned}\prod_{i \in I} A_i &= \{(a_i)_{i \in I} \mid (a_i)_{i \in I} \subseteq \bigcup_{i \in I} A_i \mid (\forall i \in I) (a_i \in A_i)\} = \\ &= \{(a_i)_{i \in I} \mid (\forall i) (i \in I \Rightarrow a_i \in A_i)\},\end{aligned}$$

sau, altfel scris (cu definiția unei familii de elemente exemplificate mai sus pe familii de numere reale):

$$\begin{aligned}\prod_{i \in I} A_i &= \{f \mid f : I \rightarrow \bigcup_{i \in I} A_i, (\forall i \in I) (f(i) \in A_i)\} = \\ &= \{f \mid f : I \rightarrow \bigcup_{i \in I} A_i, (\forall i) (i \in I \Rightarrow f(i) \in A_i)\}.\end{aligned}$$

Notăție

Fie $n \in \mathbb{N}^*$ și mulțimile A_1, A_2, \dots, A_n . Produsul direct $\prod_{i \in \overline{1, n}} A_i$ se mai notează cu

$\prod_{i=1}^n A_i$, iar un element $(a_i)_{i \in \overline{1, n}}$ al acestui produs direct se mai notează cu

(a_1, a_2, \dots, a_n) . În cazul particular în care $A_1 = A_2 = \dots = A_n = A$,

$$\prod_{i \in \overline{1, n}} A \stackrel{\text{notație}}{=} \prod_{i=1}^n A \stackrel{\text{notație}}{=} A^n.$$

Remarcă (puterile unei mulțimi: caz particular al produsului direct)

În definiția anterioară, dacă $I \neq \emptyset$ și, pentru orice $i \in I$, $A_i = A$, atunci

$$\bigcup_{i \in I} A_i = \bigcup_{i \in I} A = A, \text{ așadar } \prod_{i \in I} A = \{f \mid f : I \rightarrow A, (\forall i \in I) (f(i) \in A)\} =$$

$\{f \mid f : I \rightarrow A\} = A^I$. În particular, pentru orice $n \in \mathbb{N}^*$, dacă $|I| = n$, avem:

$$A^n = \{(a_1, \dots, a_n) \mid (\forall i \in \overline{1, n}) (a_i \in A)\} = \{f \mid f : \overline{1, n} \rightarrow A\} = A^{\overline{1, n}} = A^I.$$

Pentru orice mulțimi A, B , notăm cu $A \cong B$ faptul că există o bijecție între A și B .

Remarcă

Pentru orice mulțimi A, I și J , dacă $I \cong J$, atunci $A^I \cong A^J$.

Operații de diferite arități

Aritatea unei operații a unei structuri algebrice (cu o singură *mulțime suport*, o singură mulțime de elemente) este **numărul argumentelor (operandilor, variabilelor) acelei operații**. Dacă structura algebrică are mulțimea suport A , iar f este o operație n -ară pe A , cu $n \in \mathbb{N}$, atunci $f : \underbrace{A \times A \times \dots \times A}_{n \text{ de } A} \rightarrow A$ (f este o funcție cu n argumente din A , cu valori tot în A).

Exemplu

Într-un grup $(G, \circ, ^{-1}, e)$, avem trei operații:

- operația *binară* \circ (**compunerea elementelor grupului, două câte două**) (operație de aritate 2, operație cu două argumente): $\circ : G \times G \rightarrow G$
- operația *unară* $^{-1}$ (**inversarea fiecărui element din grup**) (operație de aritate 1, operație cu un singur argument): $^{-1} : G \rightarrow G$
- operația *zeroară* (sau *nulară*) e (**elementul neutru al grupului**) (operație de aritate 0, operație fără argumente, i. e. **constantă** din G): $e \in G$

Operațiile zeroare sunt elemente distinse, i. e. constante

De ce operațiile zeroare sunt același lucru cu **elemente distinse, constante** din mulțimea suport a structurii algebrice?

Urmând regula de mai sus, elementul neutru e al grupului G trebuie să fie o funcție de la produsul direct al familiei vide la G .

Produsul direct al familiei vide nu este \emptyset , cum s-ar putea crede, ci este un *singleton*, adică o mulțime cu un singur element.

Într-adevăr, dacă recitim de mai sus definiția produsului direct al unei familii arbitrare de mulțimi, observăm că produsul direct al familiei vide este mulțimea funcțiilor de la mulțimea vidă la reuniunea familiei vide, care satisfac o proprietate întotdeauna adevărată (anume $(\forall i) (i \in \emptyset \Rightarrow \dots)$, iar $i \in \emptyset$ este fals pentru orice i , deci implicația anterioară este adevărată pentru orice i , deci această proprietate cu variabila i cuantificată universal este adevărată), adică produsul direct al familiei vide este mulțimea funcțiilor de la mulțimea vidă la reuniunea familiei vide, care are un singur element, pentru că de la \emptyset la orice mulțime există o unică funcție, așa cum am văzut mai sus.

Remarcă (reuniunea familiei vide este vidă)

Dacă recitim și definiția reuniunii unei familii arbitrare, observăm că reuniunea familiei vide este mulțimea elementelor pentru care există un $i \in \emptyset$ cu o anumită proprietate, condiție care este întotdeauna falsă, deci reuniunea familiei vide este \emptyset .

Operațiile zeroare sunt elemente distinse, i. e. constante

Remarcă (produsul direct al familiei vide este un singleton)

Cele de mai sus arată că produsul direct al familiei vide este mulțimea cu unicul element dat de unica funcție de la \emptyset la \emptyset , anume $(\emptyset, \emptyset, \emptyset)$, adică produsul direct al familiei vide este singletonul $\{(\emptyset, \emptyset, \emptyset)\}$.

Prin urmare, elementul neutru al grupului G este o funcție φ de la singletonul $G^\emptyset = G^0 = \{(\emptyset, \emptyset, \emptyset)\}$ la G :

$$\varphi : G^\emptyset = G^0 = \{(\emptyset, \emptyset, \emptyset)\} \rightarrow G,$$

iar o funcție definită pe un singleton are o singură valoare ($\varphi((\emptyset, \emptyset, \emptyset)) \in G$), i.e. are imaginea tot un singleton:

$$\varphi(G^\emptyset) = \varphi(G^0) = \{\varphi((\emptyset, \emptyset, \emptyset))\} \subseteq G,$$

conținând acea unică valoare, deci poate fi identificată cu această unică valoare a ei, care este un element distins, o constantă din G :

$$\varphi((\emptyset, \emptyset, \emptyset)) = e \in G,$$

și identificăm:

$$\varphi = e.$$

Exemplu (structură algebrică având mai multe mulțimi suport, i. e. mai multe mulțimi (tipuri) de elemente)

Ca o paranteză, o **structură algebrică cu două mulțimi suport** este **spațiul vectorial**, care are ca mulțimi suport **mulțimea scalarilor** (formând un corp, K), și **mulțimea vectorilor** (formând un grup abelian, V). Nu este același lucru cu o structură algebrică având ca unică mulțime suport pe $K \times V$, pentru că nu avem operații pe $K \times V$ (i. e. operații de la $K \times V \times K \times V \times \dots \times K \times V$ la $K \times V$), ci avem operații de grup pe V , operații de corp pe K și operația de compunere (înmulțire) a scalarilor cu vectorii, de la $K \times V$ la V .

Observație (cu ce unifică o constantă, i.e. o operație zeroară (operație fără argumente, fără operanzi))

Dacă revedeți discuția informală de mai sus despre **unificare**, puteți observa că o constantă **nu unifică**:

- nici cu o altă constantă,
- nici cu o expresie având ca operator dominant o funcție (i. e. operație) cu argumente (i.e. operanzi), i.e. de aritate nenulă.

O constantă **unifică** numai cu **ea însăși** sau cu o **variabilă**.