Tema 10

Scopul acestei teme este de a va familiariza cu conceptele urmatoare:

* Backtracking
* Predicatul cut (de taiere)
* Acumulatori
* Combinatia cut - fail

Cerinte.

**1.Scrieți un predicat care elimină toate elementele care apar de mai multe ori dintr-o listă, folosind un acumulator.**

Rezolvare:

remove\_duplicates\_acc([], Acc, Acc).

remove\_duplicates\_acc([H|T], Acc, Result) :-

member(H, Acc),

remove\_duplicates\_acc(T, Acc, Result).

remove\_duplicates\_acc([H|T], Acc, Result) :-

\+ member(H, Acc),

remove\_duplicates\_acc(T, [H|Acc], Result).

Am testat codul pe urmatoarele exemple specifice:

?- remove\_duplicates([1, 2, 3, 2, 4, 1, 5, 3, 6, 2, 4], Result).

Result = [6, 5, 1, 2, 3, 4]

?- remove\_duplicates([1, 2, 3, 4], Result).

Result = [4, 3, 2, 1]

?- remove\_duplicates([], Result).

Result = []

**2.Scrieți un predicat care verifică dacă o listă este ordonată în mod crescător, folosind un acumulator.**

ordered\_list(List) :-

ordered\_list\_acc(List, \_).

ordered\_list\_acc([], \_).

ordered\_list\_acc([X|Rest], Prev) :-

Prev =< X,

ordered\_list\_acc(Rest, X).

Am testat codul pentru urmatoarele exemple:

?- ordered\_list([1, 2, 3, 4, 5]).

True

?- ordered\_list([1]).

True

?- ordered\_list([]).

True

?- ordered\_list([1, 3, 2, 4, 5]).

false

**3.Scrieți un predicat care verifică dacă un element apare exact de două ori într-o listă.**

Rezolvare:

% Predicatul count\_occurrences/3 numără aparițiile unui element într-o listă.

count\_occurrences([], \_, 0).

count\_occurrences([X|T], X, N) :-

count\_occurrences(T, X, N1),

N is N1 + 1.

count\_occurrences([H|T], X, N) :-

H \= X,

count\_occurrences(T, X, N).

Am testat codul pt urmatoarele:

?- count\_occurrences([1, 2, 3, 2, 4], 2, Count).

Count = 2

?- count\_occurrences([a, b, a, c, d], a, Count).

Count = 2

**4.Folosind predicatul cut, scrieti un predicat care calculeaza suma elementelor pare din lista si produsul elementelor impare. Presupunem, pentru simplificare, ca lista contine doar numere!**

Rezolvare:

% Predicatul sum\_even\_odd/3 folosește cut pentru a separa cazurile elementelor pare și impare.

% Predicatul sum\_even\_odd/2 inițializează acumulatorii și apelează predicatul auxiliar.

sum\_even\_odd(List, Sum, Product) :- sum\_even\_odd(List, 0, 1, Sum, Product).

sum\_even\_odd([], Sum, Product, Sum, Product).

% Cazul elementelor pare. Adăugăm elementul la suma parțială

sum\_even\_odd([H|T], SumAcc, ProductAcc, Sum, Product) :-

H mod 2 =:= 0,

NewSumAcc is SumAcc + H,

sum\_even\_odd(T, NewSumAcc, ProductAcc, Sum, Product).

% Cazul elementelor impare. Înmulțim elementul cu produsul parțial

sum\_even\_odd([H|T], SumAcc, ProductAcc, Sum, Product) :-

H mod 2 =\= 0,

NewProductAcc is ProductAcc \* H,

sum\_even\_odd(T, SumAcc, NewProductAcc, Sum, Product).

Am testat codul pt urmatoarele cazuri:

?- sum\_even\_odd([1, 2, 3, 4, 5], Sum, Product).

Sum = 6,

Product = 15

?- sum\_even\_odd([2, 4, 6, 8], Sum, Product).

Sum = 20,

Product = 1

?- sum\_even\_odd([], Sum, Product).

Sum = 0,

Product = 1

**5.Pornind de la urmatoarea baza de fapte si reguli, raspundeti la urmatoarele intrebari**

**Ce returneaza urmatoarele interogari ? Explicati, pas cu pas, in words, ce se intampla in spate (i.e. ce reguli si ce fapte sunt apelate)**

|  |
| --- |
| ?- enrolled(john, csc101).  ?- enrolled\_in\_department(john, computer\_science).  ?- grade(jane, csc101, Grade), Grade >= 90.  ?- teaches(X, Y), department(Y) = computer\_science. |

?- enrolled(john, csc101).

true

Se verifică dacă există o înregistrare "enrolled(john, csc101)" în baza de fapte.

Deoarece există înregistrarea "enrolled(john, csc101)" se returneaza true.

?- enrolled\_in\_department(john, computer\_science).

False

Se verifică dacă există un student numit "john" în departamentul "computer\_science" pe baza regulilor și faptelor disponibile.

Se verifică dacă "john" este un student și "computer\_science" este un departament.

Se verifică dacă există o înregistrare "enrolled(john, Course)" în baza de fapte.

Se verifică dacă există un profesor care predă cursul "Course"

Deoarece nu exista fapte a un profesor predă departamentului computer science , se va returna false

?- grade(jane, csc101, Grade), Grade >= 90.

Grade = 92

Verifică dacă există o înregistrare "grade(jane, csc101, Grade)" în baza de fapte și dacă "Grade" este mai mare sau egal cu 90.

Pentru ca o înregistrare "grade(jane, csc101, 92)" și 92 este mai mare sau egal cu 90, interogarea va returna adevărat (true).

?- teaches(X, Y), department(Y) = computer\_science.

false

Se caută toate înregistrările "teaches(X, Y)" care satisfac condiția "department(Y) = computer\_science".

Se verifică dacă există un profesor "X" care predă un curs "Y" și dacă "Y" este un departament "computer\_science".

Pentru ca există o înregistrare "teaches(jim, csc101)" și "csc101" nu este un departament "computer\_science", interogarea va returna fals (false).

**Construiti query-uri (puteti adauga si reguli aditionale, daca este cazul) pentru urmatoarele intrebari:**

1.Care sunt toate cursurile predate de un anumit profesor?

courses\_taught\_by(Professor, Course) :-

professor(Professor),

teaches(Professor, Course).

Am testat pentru:

courses\_taught\_by(jim,Course).

Am obtinut :

Course = csc101

2.Care este media notelor pentru un anumit curs?

course\_average(Course, Average) :-

course(Course),

findall(Grade, grade(\_, Course, Grade), Grades),

length(Grades, NumGrades),

sum\_list(Grades, Total),

Average is Total / NumGrades.

Am testat pentru : course\_average(csc101 , Average). ; Am obtinut: Average = 88.5

3.Cine sunt toți studenții care au obținut o notă mai mare decât 80 la un anumit curs?

students\_above\_grade(Course, MinGrade, Students) :-

course(Course),

findall(Student, (grade(Student, Course, Grade), Grade > MinGrade), Students).

Am testat: ?- students\_above\_grade(csc101, 80, Students).

Raspuns: Students = [john, jane]