# Cours de Programmation Déclarative et Bases de Données

# Prolog - récursivité

Nicolas Jouandeau

n@up8.edu

2022

## récursivité

- quand un ou plusieurs sous-buts sont définis par le même prédicat que le but
- variation de la position de l'appel récursif dans les sous-buts
  - selon le fonctionnement d'un prédicat et les arguments
  - selon la forme de récursion (i.e. terminale ou non)

## motifs possibles

```
f(...):-a(...),f(...).          % sous-buts avant
f(...):-a(...),f(...),b(...).   % sous-buts avant et après
f(...):-f(...),a(...).          % sous-buts après
f(...):-a(...);f(...).          % avec un OU avant
f(...):-f(...);a(...).          % avec un OU après
f(X):-f(X,0).                   % paramètres par défaut
```

## premier exemple

- définir le prédicat `nprint(N)` qui affiche `N` fois "message"

## exécution

```
?- nprint(0).
true .

?- nprint(3).
message
message
message
true .
```

## première solution

```
nprint(N):-N=0.
nprint(N):-N>0,write('message\n'),N1 is N-1,nprint(N1).
```

## deuxième solution

```
nprint(0).
nprint(N):-N>0,write('message\n'),N1 is N-1,nprint(N1).
```

## incrémenter/décrémenter

- définir `print_increase(N)` qui affiche les nombres de 1 à `N`
- définir `print_decrease(N)` qui affiche les nombres de `N` à 1

## exécution

```
?- print_increase(5).
1 2 3 4 5
true .
?- print_decrease(5).
5 4 3 2 1
true .
```

## solution

```
print_increase(0).
print_increase(N):-N>0,N1 is N-1,
  print_increase(N1),writef('%w ',[N]).

print_decrease(0).
print_decrease(N):-N>0,writef('%w ',[N]),
  N1 is N-1,print_decrease(N1).
```

### autre solution (équivalente avec un OU et en une seule déclaration)

```
print_increase(N):-N=0;N>0,N1 is N-1,
  print_increase(N1),writef('%w ',[N]).
```

### autre solution (supposant un utilisateur bienveillant)

```
print_increase(0).
print_increase(N):-N1 is N-1,
  print_increase(N1),writef('%w ',[N]).
```

### autre solution (avec accumulateur)

```
print_increase(X,Y):-X=<Y,writef('%w ',[X]),
  X1 is X+1,print_increase(X1,Y).
print_increase(X):-X1 is X+1,print_increase(1,X1).
```

### autre solution (avec accumulateur et utilisateur bienveillant)

```
print_increase(X,Y):-X\=Y,writef('%w ',[X]),
  X1 is X+1,print_increase(X1,Y).
print_increase(X):-X1 is X+1,print_increase(1,X1).
```

## visualiser les déclarations de `print_increase` avec `listing/1`

```
$> swipl
?- [print_inc_dec].
true.

?- listing(print_increase).
print_increase(0).
print_increase(A) :-
A>0,
B is A+ -1,
print_increase(B),
writef('%w ', [A]).
true.

?- ^D
$>
```

### print_increase.pl

```
print_increase(0).
print_increase(N):-N>0,
  N1 is N-1,
  print_increase(N1),
  writef('%w ',[N]).
```

## commentaires

- les variables sont renommées
- un fait, un but ou un sous-but par ligne
- certains sous-buts sont réécrits différemment

## trace de l'exécution de `print_increase`

```
print_increase(3).
   Call: (8) print_increase(3) ? creep
   Call: (9) 3=0 ? creep
   Fail: (9) 3=0 ? creep
   Redo: (8) print_increase(3) ? creep
   Call: (9) 3>0 ? creep
   Exit: (9) 3>0 ? creep
   Call: (9) _2596 is 3+ -1 ? creep
   Exit: (9) 2 is 3+ -1 ? creep
   Call: (9) print_increase(2) ? creep
   Call: (10) 2=0 ? creep
   Fail: (10) 2=0 ? creep
   Redo: (9) print_increase(2) ? creep
   Call: (10) 2>0 ? creep
   Exit: (10) 2>0 ? creep
   Call: (10) _2602 is 2+ -1 ? creep
   Exit: (10) 1 is 2+ -1 ? creep
   Call: (10) print_increase(1) ? creep
   Call: (11) 1=0 ? creep
   Fail: (11) 1=0 ? creep
```

### print_increase.pl

```
print_increase(0).
print_increase(N):-N>0,
  N1 is N-1,
  print_increase(N1),
  writef('%w ',[N]).
```

## trace de l'exécution de `print_increase` (suite)

```
   Redo: (10) print_increase(1) ? creep
   Call: (11) 1>0 ? creep
   Exit: (11) 1>0 ? creep
   Call: (11) _2608 is 1+ -1 ? creep
   Exit: (11) 0 is 1+ -1 ? creep
   Call: (11) print_increase(0) ? creep
   Call: (12) 0=0 ? creep
   Exit: (12) 0=0 ? creep
   Exit: (11) print_increase(0) ? creep
   Call: (11) writef:writef('%w ', [1]) ? creep
1
   Exit: (11) writef:writef('%w ', [1]) ? creep
   Exit: (10) print_increase(1) ? creep
   Call: (10) writef:writef('%w ', [2]) ? creep
2
   Exit: (10) writef:writef('%w ', [2]) ? creep
   Exit: (9) print_increase(2) ? creep
   Call: (9) writef:writef('%w ', [3]) ? creep
3
   Exit: (9) writef:writef('%w ', [3]) ? creep
   Exit: (8) print_increase(3) ? creep
true .
```

### print_increase.pl

```
print_increase(0).
print_increase(N):-N>0,
  N1 is N-1,
  print_increase(N1),
  writef('%w ',[N]).
```

## arbre de l'exécution de `print_increase(3)`

```
?- print_increase(3).
   |__________
   |          |
  { 3=0 }    { N=3 }
  faux       ?- 3>0,X is 3-1,print_increase(X),write('3 ').
                |
               vrai
                |
             ?- X is 3-1,print_increase(X),write('3').
                |
               { X=2 }
                |
             ?- print_increase(2),write('3 ').
                |_________
                |         |
               { 2=0 }  { N=2 }
               faux     ?- 2>0,X is 2-1,print_increase(X),write('2 ').
                           |
                          vrai
                           |
                        ?- X is 2-1,print_increase(X),write('2 ').
                           |
                          { X=1 }
                           |
                        ?- print_increase(1),write('2 ').
                           |
                          ...
```

### print_increase.pl

```
print_increase(0).
print_increase(N):-N>0,
  N1 is N-1,
  print_increase(N1),
  writef('%w ',[N]).
```

## arbre de l'exécution de `print_increase(3)` (suite)

```
?- print_increase(1),write('2 ').
   |____________
   |             |
  { 1=0 }       { N=1 }
  faux          ?- 1>0,X is 1-1,print_increase(X),write('1 ').
                   |
                  vrai
                   |
                ?- X is 1-1,print_increase(X),write('1 ').
                   |
                  { X=0 }
                   |
                ?- print_increase(0),write('1 ').
                   |
                  { 0=0 }
                   |
                ?- write('1 ').
                   |
                /* affiche 1 */
                  vrai
/* reprise de ?- vrai, write('2 '). */
   |
 ?- write('2 ').
   |
 /* affiche 2 */
  vrai
/* reprise de ?- vrai, write('3 '). */
   |
 ?- write('3 ').
   |
 /* affiche 3 */
  vrai
```

### print_increase.pl

```
print_increase(0).
print_increase(N):-N>0,
  N1 is N-1,
  print_increase(N1),
  writef('%w ',[N]).
```

## incrémenter de A à B

- définir `print_numbers(A,B)` qui affiche les nombres de `A` à `B`

## exécution

```
?- print_numbers(2,5).
2 3 4 5
true .

?- print_numbers(5,5).
5
true .

?- print_numbers(6,5).
true .
```

## solution

```
print_numbers(A,B):-A>B.
print_numbers(A,B):-A=<B,writef('%w ',[A]),
  N is A+1,print_numbers(N,B).
```

## être pair

- définir le prédicat `is_even(X)` qui est vrai pour toute valeur de `X` paire positive ou nulle sachant que
  - 0 est pair
  - pour un nombre pair non nul,
    on obtient le nombre pair précédent en lui retranchant 2
- en utilisant `>`, `is`, `-`
- en utilisant `\=`, `is`, `-`

## exécution

```
?- is_even(0).
true .

?- is_even(3).
false.

?- is_even(10).
true .
```

## solution avec >

```
is_even(0).
is_even(X):-X>0,X1 is X-2,is_even(X1).
```

## solution avec \=

```
is_even(0).
is_even(X):-X\=1,X1 is X-2,is_even(X1).
```

## somme des entiers

- définir le prédicat `integer_sum(N,R)` qui est vrai si `R` est la somme des entiers de 1 à `N`
- en forme récursive non terminale
- en forme récursive terminale

## exécution

```
?- integer_sum(10,X).
X = 55 .
```

## somme des entiers

- définir le prédicat `integer_sum(N,R)` qui est vrai si `R` est la somme des entiers de 1 à `N`
- en forme récursive non terminale
- en forme récursive terminale

## exécution

```
?- integer_sum(10,X).
X = 55 .
```

## forme récursive non terminale

```
integer_sum(1,1).
integer_sum(N,R):-N>1,N1 is N-1,integer_sum(N1,R1),R is R1+N.
```

## forme récursive terminale

```
integer_sum_t(0,R,R).
integer_sum_t(N,ACC,R):-N>0,N1 is N-1,ACC1 is ACC+N,
  integer_sum_t(N1,ACC1,R).
integer_sum_t(X,Y):-integer_sum_t(X,0,Y).
```

## somme des carrés

- définir le prédicat `square_sum(N,R)` qui est vrai si `R` est la somme des carrés de 1 à `N`

## exécution

```
?- square_sum(1,X).
X = 1 .

?- square_sum(5,X).
X = 55 .
```

## somme des carrés

- définir le prédicat `square_sum(N,R)` qui est vrai si `R` est la somme des carrés de 1 à `N`

## exécution

```
?- square_sum(1,X).
X = 1 .

?- square_sum(5,X).
X = 55 .
```

## solution

```
square_sum(1,1).
square_sum(N,R):-N>1,N1 is N-1,square_sum(N1,R1),R is R1+N*N.
```

## somme des entiers par incrémentation et décrémentation

- définir `sum_decrease(X1,X2,R)` qui est vrai si `R` est la somme des entiers de `X1` à `X2` (en décrémentant `X2`)
- définir `sum_increase(X1,X2,R)` qui est vrai si `R` est la somme des entiers de `X1` à `X2` (en incrémentant `X1`)
- sous l'hypothèse de `X1` ≤ `X2`

## exécution

```
?- sum_decrease(1,3,X).
X = 6 .

?- sum_decrease(5,10,X).
X = 45 .

?- sum_increase(1,3,X).
X = 6 .

?- sum_increase(5,10,X).
X = 45 .
```

### solution

```
sum_decrease(X1,X1,X1).
sum_decrease(X1,X2,R):-X1\=X2,X3 is X2-1,
  sum_decrease(X1,X3,R1),
  R is R1+X2.

sum_increase(X1,X1,X1).
sum_increase(X1,X2,R):- X1\=X2,X3 is X1+1,
  sum_increase(X3,X2,R1),
  R is R1+X1.
```

## somme des nombres pairs entre A et B

- définir le prédicat `sum_even(A,B,R)` qui est vrai si `R` est la somme des nombres pairs entre `A` et `B` compris
- utiliser `is_even(X)`, `not`, `\=`, `is`, `+`

## exécution

```
?- sum_even(1,3,X).
X = 2 .

?- sum_even(5,10,X).
X = 24 .
```

## solution

```
is_even(0).
is_even(X):-X\=1,X1 is X-2,is_even(X1).

sum_even(A,A,B):-is_even(A), B is A.
sum_even(A,A,0):-not(is_even(A)).
sum_even(A,B,0):-A>B.
sum_even(A,B,R):-A\=B,is_even(A),A1 is A+2,
  sum_even(A1,B,R1), R is A+R1.
sum_even(A,B,R):-A\=B,not(is_even(A)),
  A1 is A+1,sum_even(A1,B,R).
```

## autre solution (avec even, odd et modulo)

```
even(X):- 0 is mod(X,2).
odd(X):- 1 is mod(X,2).

sum_even(X,X,X).
sum_even(X,Y,Z):-X<Y,even(X),even(Y),X1 is X+2,
  sum_even(X1,Y,Z1), Z is Z1+X.
sum_even(X,Y,Z):-X<Y,odd(X),even(Y),X1 is X+1,
  sum_even(X1,Y,Z).
sum_even(X,Y,Z):-X<Y,odd(Y),Y1 is Y-1,
  sum_even(X,Y1,Z).
```

## somme des racines-carrées

- définir le prédicat `sqrt_sum(A,B,R)` qui est vrai si `R` est la somme des racines-carrées entre `A` et `B` compris
  - en utilisant le prédicat `sqrt(X,Y)`
  - en utilisant l'opérateur arithmétique `sqrt(X)`
- définir le prédicat `int_sqrt_sum(A,B,R)` qui est vrai si `R` est l'arrondi entier de `X` satisfaisant `sqrt_sum(A,B,X)`
  - en utilisant le prédicat `sqrt_sum(X,Y,Z)`
  - en utilisant l'opérateur arithmétique `integer(X)`

## exécution

```
?- sqrt_sum(1,10,X).
X = 22.4682781862041 .

?- int_sqrt_sum(1,10,X).
X = 22 .
```

### solution avec le prédicat `sqrt(X,Y)`

```
sqrt_sum(X1,X1,R):-sqrt(X1,R).
sqrt_sum(X1,X2,R):-X1<X2,X3 is X1+1,
  sqrt_sum(X3,X2,R1),sqrt(X1,R2),R is R1+R2.
```

### solution avec l'opérateur arithmérique `sqrt(X)`

```
sqrt_sum(X,X,Y):-Y is sqrt(X).
sqrt_sum(X,Y,Z):-X<Y,X1 is X+1,
  sqrt_sum(X1,Y,Z1),Z is Z1+sqrt(X).
```

### solution avec un arrondi entier

```
int_sqrt_sum(X1,X2,R):-sqrt_sum(X1,X2,R1),R is integer(R1).
```

## factoriel

- définir le prédicat `fac(X,Y)` qui est vrai si `Y=X!`

## exécution

```
?- fac(3,X).
X = 6 .

?- fac(5,X).
X = 120 .
```

## factoriel

- définir le prédicat `fac(X,Y)` qui est vrai si `Y=X!`

## exécution

```
?- fac(3,X).
X = 6 .

?- fac(5,X).
X = 120 .
```

## solution

```
fac(1,1).
fac(N,R):-N>1,N1 is N-1,fac(N1,R1),R is R1*N.
```

## fibonacci et tribonacci

- définir le prédicat `fibo(A,F)` qui est vrai si `F` est le $A^{\text{ième}}$ nombre de Fibonacci sachant

$$\begin{cases} \text{fibo}(0) & = 0 \\ \text{fibo}(1) & = 1 \\ \text{fibo}(n+2) = \text{fibo}(n+1) + \text{fibo}(n) \end{cases}$$

- définir le prédicat `tribo(A,F)` qui est vrai si `F` est le $A^{\text{ième}}$ nombre de Tribonacci sachant

$$\begin{cases} \text{tribo}(0) & = 0 \\ \text{tribo}(1) & = 1 \\ \text{tribo}(2) & = 1 \\ \text{tribo}(n+2) = \text{tribo}(n+2) + \text{tribo}(n+1) + \text{tribo}(n) \end{cases}$$

## exécution

```
?- fibo(10,X).
X = 55 .

?- tribo(10,X).
X = 149 .
```

## fibonacci

```
fibo(0,0).
fibo(1,1).
fibo(A,F):-A>1,A1 is A-1,fibo(A1,F1),
  A2 is A-2,fibo(A2,F2),F is F1+F2.
```

## tribonacci

```
tribo(0,0).
tribo(1,1).
tribo(2,1).
tribo(X,R):-X>2,X1 is X-1,tribo(X1,R1),
  X2 is X-2,tribo(X2,R2),
  X3 is X-3,tribo(X3,R3),R is R1+R2+R3.
```

## puissance

- définir le prédicat `pow(X,N,P)` qui est vrai si `P` est `X` puissance `N` sachant

$$\begin{cases} X^0 = 1 \\ X^n = X * X^{n-1} \end{cases}$$

## exécution

```
?- pow(2,5,X).
X = 32 .

?- pow(3,3,X).
X = 27 .
```

## solution

```
pow(_,0,1).
pow(X,N,P):-N>0,N1 is N-1,pow(X,N1,P1),P is X*P1.
```

## ackermann et sudan

- définir le prédicat `ackermann(M,N,R)` qui est vrai si `R` est $A(M,M)$ pour $A$ la fonction d'Ackermann sachant

$$\begin{cases} A(m,n) = n+1 & \text{si } m=0 \\ A(m,n) = A(m-1,1) & \text{si } m>0 \text{ et } n=0 \\ A(m,n) = A(m-1,A(m,n-1)) & \text{si } m>0 \text{ et } n>0 \end{cases}$$

- définir le prédicat `sudan(N,X,Y,R)` qui est vrai si `R` est $S(N,X,Y)$ pour $S$ la fonction de Sudan sachant

$$\begin{cases} S(n,x,y) = x+y & \text{si } n=0 \\ S(n,x,y) = x & \text{si } n>0 \text{ et } y=0 \\ S(n,x,y) = S(n-1,S(n,x,y-1),S(n,x,y-1)+y) & \text{sinon} \end{cases}$$

## exécution

```
?- ackermann(3,4,X).
X = 125 .

?- sudan(1,5,6,X).
X = 440 .

?- sudan(2,5,1,X).
X = 440 .
```

## ackermann

```
ackermann(0,N,R):-R is N+1.
ackermann(M,N,R):-M>0,N=0,M1 is M-1,
  ackermann(M1, 1, R).
ackermann(M,N,R):-M>0,N\=0,M1 is M-1,N1 is N-1,
  ackermann(M,N1,N2),ackermann(M1,N2,R).
```

## sudan

```
sudan(0,X,Y,R):-R is X+Y.
sudan(N,X,0,R):-N>0,R is X.
sudan(N,X,Y,R):-N>0,Y>0,N1 is N-1,Y1 is Y-1,
  sudan(N,X,Y1,R1),R2 is R1+Y,
  sudan(N1,R1,R2,R).
```