Tema 7: Estilo y eficiencia en programación lógica

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Principios generales

- **Un programa debe ser:**
  - **Correcto**
  - **Eficiente**
  - **De lectura fácil**
  - **Modificable: Modular y transparente**
  - **Robusto**
  - **Documentado**

Declarativo y procedimental

- **Pensar procedimental y declarativamente**
  - **Programas**

```
siguiente(a,1).
siguiente(1,b).

sucesor_1(X,Y) :-
   siguiente(X,Y).
sucesor_1(X,Y) :-
   siguiente(X,Z),
   sucesor_1(Z,Y).

sucesor_2(X,Y) :-
   siguiente(X,Y).
sucesor_2(X,Y) :-
   sucesor_2(Z,Y),
   siguiente(X,Z).
```

  - **Sesiones**

```
?- sucesor_1(X,Y).
X = a  Y = 1 ; X = 1  Y = b ; X = a  Y = b ;
No
?- findall(X-Y,sucesor_1(X,Y),L).
L = [a-1, 1-b, a-b]
Yes
?- sucesor_2(X,Y).
X = a  Y = 1 ; X = 1  Y = b ; X = a  Y = b ;
ERROR: Out of local stack
?- findall(X-Y,sucesor_2(X,Y),L).
ERROR: Out of local stack
```

Orden de los literales

• Orden de los literales

- **Programa:**

```
crea(X) :-
   between(1,X,N),
   assertz(numero(N)),
   N=X,
   X1 is X // 100,
   between(1,X1,M),
   M1 is M*100,
   assertz(multiplo_de_100(M1)),
   X1=M.
```

- **Sesión:**

```
?- crea(1000).
Yes
?- listing(numero).
numero(1). numero(2). ... numero(1000).
Yes
?- listing(multiplo_de_100).
multiplo_de_100(100). ... multiplo_de_100(1000).
Yes
?- time((numero(_N),multiplo_de_100(_N))).
101 inferences in 0.00 seconds (Infinite Lips)
Yes
?- time((multiplo_de_100(_N),numero(_N))).
2 inferences in 0.00 seconds (Infinite Lips)
Yes
```

Comprobaciones parciales

- **Cuadrado mágico:**

```
+---+---+---+
| A | B | C |
+---+---+---+
| D | E | F |
+---+---+---+
| G | H | I |
+---+---+---+
```

  - **Programa 1:**

```
calcula_cuadrado_1([A,B,C,D,E,F,G,H,I]) :-
  permutacion([1,2,3,4,5,6,7,8,9],[A,B,C,D,E,F,G,H,I]),
  A+B+C =:= 15,  D+E+F =:= 15,
  G+H+I =:= 15,  A+D+G =:= 15,
  B+E+H =:= 15,  C+F+I =:= 15,
  A+E+I =:= 15,  C+E+G =:= 15.

permutacion([],[]).
permutacion([X|L1],L2) :-
   permutacion(L1,L3),
   select(X,L2,L3).
```

  - **Sesión 1:**

```
?- calcula_cuadrado_1(L).
L = [6, 1, 8, 7, 5, 3, 2, 9, 4] ;
L = [8, 1, 6, 3, 5, 7, 4, 9, 2]
Yes
?- findall(_X,calcula_cuadrado_1(_X),_L),length(_L,N).
N = 8
Yes
```

Comprobaciones parciales

- **Programa 2:**

```
calcula_cuadrado_2([A,B,C,D,E,F,G,H,I]) :-
   select([1,2,3,4,5,6,7,8,9],A,L1),
   select(L1,B,L2),
   select(L2,C,L3),  A+B+C =:= 15,
   select(L3,D,L4),
   select(L4,G,L5),  A+D+G =:= 15,
   select(L5,E,L6),  C+E+G =:= 15,
   select(L6,I,L7),  A+E+I =:= 15,
   select(L7,F,[H]), C+F+I =:= 15, D+E+F =:= 15.
```

- **Sesión 2:**

```
?- calcula_cuadrado_2(L).
L = [2, 7, 6, 9, 5, 1, 4, 3, 8] ;
L = [2, 9, 4, 7, 5, 3, 6, 1, 8]
Yes
?- setof(_X,calcula_cuadrado_1(_X),_L),
   setof(_X,calcula_cuadrado_2(_X),_L).
Yes
```

- **Comparación de eficiencia**

```
?- time(calcula_cuadrado_1(_X)).
% 161,691 inferences in 0.58 seconds (278778 Lips)
?- time(calcula_cuadrado_2(_X)).
% 1,097 inferences in 0.01 seconds (109700 Lips)
?- time(setof(_X,calcula_cuadrado_1(_X),_L)).
% 812,417 inferences in 2.90 seconds (280144 Lips)
?- time(setof(_X,calcula_cuadrado_2(_X),_L)).
% 7,169 inferences in 0.02 seconds (358450 Lips)
```

Uso de listas

- **En general sólo cuando el número de argumentos no es fijo o es desconocido**
- **Ejemplos**

```
?- setof(N,between(1,1000,N),L1),
   asserta(con_lista(L1)),
   Term =.. [f|L1],
   asserta(con_term(Term)).

?- listing(con_lista).
con_lista([1, 2, ..., 999, 1000]).

?- listing(con_term).
con_term(f(1, 2,...,999, 1000)).

?- time((con_lista(_L),member(1000,_L))).
1,001 inferences in 0.00 seconds (Infinite Lips)
Yes

?- time((con_term(_T),arg(_,_T,1000))).
2 inferences in 0.00 seconds (Infinite Lips)
Yes
```

Uso de la unificación

- `intercambia(+T1,-T2)` **se verifica si** `T1` **es un término con dos argumentos y** `T2` **es un término con el mismo símbolo de función que** `T1` **pero sus argumentos intercambiados**
- **Ejemplo**

```
?- intercambia_1(opuesto(3,-3),T).
T = opuesto(-3, 3)
Yes
```

- **Versión 1:**

```
intercambia_1(T1, T2) :-
   functor(T1, F, 2), functor(T2, F, 2),
   arg(1,T1,X1),      arg(2,T1,Y1),
   arg(1,T2,X2),      arg(2,T2,Y2),
   X1 = Y2,           X2 = Y1.
```

- **Versión 2:**

```
intercambia_2(T1,T2) :-
   T1 =.. [F,X,Y],
   T2 =.. [F,Y,X].
```

- `lista_de_tres(L)` **se verifica si** `L` **es una lista de 3 elementos**
- **Versión 1:**

```
lista_de_tres(L) :- length(L, N), N = 3.
```

- **Versión 2:**

```
lista_de_tres(L) :- length(L,3).
```

- **Versión 3:**

```
lista_de_tres([_,_,_]).
```

Uso de lemas

- **Fibonacci 1:**

```
fib_1(1,1).
fib_1(2,1).
fib_1(N,F) :-
   N > 2,
   N1 is N-1,
   fib_1(N1,F1),
   N2 is N-2,
   fib_1(N2,F2),
   F is F1 + F2.
```

- **Fibonacci 2:**

```
:- dynamic fib_2/2.

fib_2(1,1).
fib_2(2,1).
fib_2(N,F) :-
   N > 2,
   N1 is N-1,
   fib_2(N1,F1),
   N2 is N-2,
   fib_2(N2,F2),
   F is F1 + F2,
   asserta(fib_2(N,F)).
```

- **Comparación**

```
?- time(fib_1(20,N)).
40,585 inferences in 1.68 seconds (24158 Lips)
N = 6765
?- time(fib_2(20,N)).
127 inferences in 0.00 seconds (Infinite Lips)
N = 6765
?- listing(fib_2).
fib_2(20, 6765). fib_2(19, 4181). ... fib_2(2, 1), ...
?- time(fib_2(20,N)).
3 inferences in 0.00 seconds (Infinite Lips)
N = 6765
```

Determinismo

- **descompone(+E,-N1,-N2) se verifica si N1 y N2 son dos enteros no negativos tales que N1+N2=E.**
  - **Versión 1:**

```
descompone_1(E, N1, N2) :-
   between(0, E, N1),
   between(0, E, N2),
   E =:= N1 + N2.
```

  - **Versión 2:**

```
descompone_2(E, N1, N2) :-
   between(0, E, N1),
   N2 is E - N1.
```

  - **Comparación**

```
?- time(setof(_N1+_N2,descompone_1(10,_N1,_N2),L)).
159 inferences in 0.00 seconds (Infinite Lips)
L = [0+10,1+9,2+8,3+7,4+6,5+5,6+4,7+3,8+2,9+1,10+0]
?- time(setof(_N1+_N2,descompone_2(10,_N1,_N2),L)).
38 inferences in 0.00 seconds (Infinite Lips)
L = [0+10,1+9,2+8,3+7,4+6,5+5,6+4,7+3,8+2,9+1,10+0]
?- time(setof(_N1+_N2,descompone_1(1000,_N1,_N2),_L)).
1,004,019 inferences in 1.29 seconds (778309 Lips)
?- time(setof(_N1+_N2,descompone_2(1000,_N1,_N2),_L)).
2,018 inferences in 0.01 seconds (201800 Lips)
```

Iteración

• Longitud de una lista

• Programa 1:

```
longitud_1([],0).
longitud_1([_|L],N) :-
   longitud_1(L,N1),
   N is N1 +1.
```

• Traza 1:

```
?- trace(longitud_1).
        longitud_1/2: call redo exit fail

Yes
?- longitud_1([a,b,c],N).
T Call:  (  8) longitud_1([a, b, c], _G179)
T Call:  (  9) longitud_1([b, c], _L131)
T Call:  ( 10) longitud_1([c], _L144)
T Call:  ( 11) longitud_1([], _L157)
T Exit:  ( 11) longitud_1([], 0)
T Exit:  ( 10) longitud_1([c], 1)
T Exit:  (  9) longitud_1([b, c], 2)
T Exit:  (  8) longitud_1([a, b, c], 3)
N = 3
Yes
```

Iteración

- **Programa 2:**

```
longitud_2(L,N) :-
   longitud_2_aux(L,0,N).

longitud_2_aux([],N,N).
longitud_2_aux([_|L],N,Long) :-
   N1 is N+1,
   longitud_2_aux(L,N1,Long).
```

- **Traza**

```
?- trace([longitud_2,longitud_2_aux]).

?- longitud_2([a,b,c],N).
T Call:  (  8) longitud_2([a, b, c], _G179)
T Call:  (  9) longitud_2_aux([a, b, c], 0, _G179)
T Call:  ( 10) longitud_2_aux([b, c], 1, _G179)
T Call:  ( 11) longitud_2_aux([c], 2, _G179)
T Call:  ( 12) longitud_2_aux([], 3, _G179)
T Exit:  ( 12) longitud_2_aux([], 3, 3)
T Exit:  ( 11) longitud_2_aux([c], 2, 3)
T Exit:  ( 10) longitud_2_aux([b, c], 1, 3)
T Exit:  (  9) longitud_2_aux([a, b, c], 0, 3)
T Exit:  (  8) longitud_2([a, b, c], 3)
N = 3
Yes
```

Iteración

- **sumalista(+L,-N) se verifica si N es la suma de los números de la lista de números L**
  - **Versión 1:**

```
sumalista_1([], 0).
sumalista_1([X|R],S) :-
   sumalista_1(R, S1),
   S is S1 + X.
```

  - **Versión 2:**

```
sumalista_2(L,S) :-
   sumalista_2_aux(L,0,S).

sumalista_2_aux([],Ac,Ac).
sumalista_2_aux([X|Resto],Ac,Suma) :-
   Nuevo_ac is Ac + X,
   sumalista_2_aux(Resto,Nuevo_ac,Suma).
```

  - **Traza**

```
?- trace([sumalista_2,sumalista_2_aux]).
?- sumalista_2([3,6],N).
T Call:  (  8) sumalista_2([3, 6], _G170)
T Call:  (  9) sumalista_2_aux([3, 6], 0, _G170)
T Call:  ( 10) sumalista_2_aux([6], 3, _G170)
T Call:  ( 11) sumalista_2_aux([], 9, _G170)
T Exit:  ( 11) sumalista_2_aux([], 9, 9)
T Exit:  ( 10) sumalista_2_aux([6], 3, 9)
T Exit:  (  9) sumalista_2_aux([3, 6], 0, 9)
T Exit:  (  8) sumalista_2([3, 6], 9)
N = 9
```

Iteración

- **Inversa de una lista**
  - **Versión 1:**

```
inv_1([],[]).
inv_1([X|L1],L2) :-
   inv_1(L1,L3),
   append(L3,[X],L2).
```

  - **Versión 2:**

```
inv_2(L1,L2) :-
   inv_2_aux(L1,[],L2).

inv_2_aux([],L,L).
inv_2_aux([X|L],Acum, Sal) :-
   inv_2_aux(L,[X|Acum],Sal).
```

  - **Sesión:**

```
?- setof(_N,between(1,1000,_N),_L1),time(inv_1(_L1,_)).
501,501 inferences in 0.44 seconds (1139775 Lips)
Yes

?- setof(_N,between(1,1000,_N),_L1),time(inv_2(_L1,_)).
1,002 inferences in 0.00 seconds (Infinite Lips)
Yes
```

Iteración

- **Fibonacci con un acumulador**

```
fib_3(N,F) :-
   fib_3_aux(N,_,F).

fib_3_aux(0,_,0).
fib_3_aux(1,0,1).
fib_3_aux(N,F1,F) :-
   N > 1,
   N1 is N-1,
   fib_3_aux(N1,F2,F1),
   F is F1 + F2.
```

  - **Comparación**

```
?- time(fib_1(20,N)).
40,585 inferences in 1.68 seconds (24158 Lips)
N = 6765
Yes
?- time(fib_2(20,N)).
127 inferences in 0.00 seconds (Infinite Lips)
N = 6765
Yes
?- time(fib_3(20,N)).
21 inferences in 0.00 seconds (Infinite Lips)
N = 6765
Yes
```

Iteración

- **Traza:**

```
?- trace(fib_3).
        fib_3/2: call redo exit fail
Yes

?- trace(fib_3_aux).
        fib_3_aux/3: call redo exit fail
Yes

?- fib_3(6,X).
T Call:  (  8) fib_3(6, _G107)
T Call:  (  9) fib_3_aux(6, _L140, _G107)
T Call:  ( 10) fib_3_aux(5, _L143, _L140)
T Call:  ( 11) fib_3_aux(4, _L157, _L143)
T Call:  ( 12) fib_3_aux(3, _L171, _L157)
T Call:  ( 13) fib_3_aux(2, _L185, _L171)
T Call:  ( 14) fib_3_aux(1, _L199, _L185)
T Exit:  ( 14) fib_3_aux(1, 0, 1)
T Exit:  ( 13) fib_3_aux(2, 1, 1)
T Exit:  ( 12) fib_3_aux(3, 1, 2)
T Exit:  ( 11) fib_3_aux(4, 2, 3)
T Exit:  ( 10) fib_3_aux(5, 3, 5)
T Exit:  (  9) fib_3_aux(6, 5, 8)
T Exit:  (  8) fib_3(6, 8)
X = 8
Yes
?- trace(fib_3,-all).
?- trace(fib_3_aux,-all).
```

Añadir al principio

- `lista_de_cuadrados(+N,?L)` **se verifica si** `L` **es la lista de los cuadrados de los números de** `1` **a** `N`
- **Ejemplo**

```
?- lista_de_cuadrados(5,L).
L = [1, 4, 9, 16, 25]
```

- **Programa 1**

```
lista_de_cuadrados_1(1,[1]).
lista_de_cuadrados_1(N,L) :-
   N > 1,
   N1 is N-1,
   lista_de_cuadrados_1(N1,L1),
   M is N*N,
   append(L1,[M],L).
```

- **Programa 2**

```
lista_de_cuadrados_2(N,L) :-
   lista_de_cuadrados_2_aux(N,L1),
   reverse(L1,L).

lista_de_cuadrados_2_aux(1,[1]).
lista_de_cuadrados_2_aux(N,[M|L]) :-
   N > 1,
   M is N*N,
   N1 is N-1,
   lista_de_cuadrados_2_aux(N1,L).
```

Añadir al principio

- **Comparación**

```
?- tiempo(lista_de_cuadrados_1(10000,_L)).
50,015,003 inferencias en 68.79 segundos y 549,416 bytes.
Yes
?- tiempo(lista_de_cuadrados_2(10000,_L)).
20,007 inferencias en 0.03 segundos y 309,464 bytes.
Yes
```

- **Programa 3**

```
lista_de_cuadrados_3(N,L) :-
   findall(M,(between(1,N,X), M is X*X),L).
```

- **Comparación**

```
?- tiempo(lista_de_cuadrados_3(10000,_L)).
20,016 inferencias en 0.03 segundos y 189,588 bytes.
Yes
```

Listas de diferencias

- **Representaciones de `[a,b,c]` como listas de diferencias**

```
[a,b,c,d] - [d]
[a,b,c,1,2,3] - [1,2,3]
[a,b,c|X] - [X]
[a,b,c] - []
```

- **Concatenación de listas de diferencias**
  - **Programa**

```
conc_ld(A-B,B-C,A-C).
```

  - **Sesión**

```
?- conc_ld([a,b|RX]-RX,[c,d|RY]-RY,Z-[]).
RX = [c, d]
RY = []
Z = [a, b, c, d]
Yes

?- conc_ld([a,b|_RX]-_RX,[c,d|_RY]-_RY,Z-[]).
Z = [a, b, c, d]
Yes
```

Ordenación

- **Ordenación de una lista**
  - `ordena(L1,L2)` **se verifica si** `L2` **es la lista obtenida ordenando la lista de números** `L1` **en orden creciente**
  - **Ejemplo**

```
ordena([2,1,2,3],L)  =>  L=[1,2,2,3]
```

  - **Versión 1:**

```
ordena(L,L1) :-
   permutacion(L,L1),
   ordenada(L1).

permutacion([],[]).
permutacion([X|L1],L2) :-
   permutacion(L1,L3),
   select(X,L2,L3).

ordenada([]).
ordenada([_]).
ordenada([X,Y|L]) :-
   X =< Y,
   ordenada([Y|L]).
```

  - **Versión 2:**

```
ordena_por_seleccion(L1,[X|L2]) :-
   selecciona_menor(X,L1,L3),
   ordena_por_seleccion(L3,L2).
ordena_por_seleccion([],[]).

selecciona_menor(X,L1,L2) :-
   select(X,L1,L2),
   not((member(Y,L2), Y < X)).
```

Ordenación

- **Versión 3:**

```
ord_quicksort([],[]).
ord_quicksort([X|R],Ordenada) :-
   divide(X,R,Menores,Mayores),
   ord_quicksort(Menores,Menores_ord),
   ord_quicksort(Mayores,Mayores_ord),
   append(Menores_ord,[X|Mayores_ord],Ordenada).

divide(_,[],[],[]).
divide(X,[Y|R],[Y|Menores],Mayores) :-
   Y < X, divide(X,R,Menores,Mayores).
divide(X,[Y|R],Menores,[Y|Mayores]) :-
   Y >= X,
   divide(X,R,Menores,Mayores).
```

- **Sesión:**

```
?- time(ordena([8,7,6,5,4,3,2,1],_L)).
271,350 inferences in 2.20 seconds (123341 Lips)
Yes
?- time(ord_quicksort([8,7,6,5,4,3,2,1],_L)).
119 inferences in 0.00 seconds (Infinite Lips)
Yes
```

Ordenación

- **Comparación de la ordenación de la lista `[N,N-1,N-2,...,2,1]`**

N	ordena	ordena_por_seleccion	ord_quicksort
1	5 inf 0.00 s	8 inf 0.00 s	5 inf 0.00 s
2	10 inf 0.00 s	19 inf 0.00 s	11 inf 0.00 s
4	80 inf 0.00 s	67 inf 0.00 s	29 inf 0.00 s
8	115,524 inf 0.11 s	323 inf 0.00 s	89 inf 0.00 s
16		1,923 inf 0.00 s	305 inf 0.00 s
32		13,059 inf 0.01 s	1,121 inf 0.00 s
64		95,747 inf 0.05 s	4,289 inf 0.00 s
128		732,163 inf 0.40 s	16,769 inf 0.02 s
256		5,724,163 inf 2.95 s	66,305 inf 0.17 s
512		45,264,899 inf 22.80 s	263,681 inf 1.52 s
1024			1,051,649 inf 5.70 s

Bibliografía

- Bratko, I. *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
  - Cap. 8: "Programming Style and Technique"
- Clocksin, W.F. y Mellish, C.S. *Programming in Prolog (Fourth Edition)* (Springer Verlag, 1994)
  - Cap. 6: "Using Data Structures"
- Covington, M.A. *Efficient Prolog: A Practical Guide*
  - `http://aisun0.ai.uga.edu/~mc/ai198908.ps`
- Hermenegildo, M. et al. *Lógica Algorítmica (1999-2000)* UPM
  - `http://www.clip.dia.fi.upm.es/~logalg/index.html`
- Flach, P. *Simply Logical (Intelligent Reasoning by Example)* (John Wiley, 1994)
  - Cap. 3: "Logic programming and Prolog".
- Van Le, T. *Techniques of Prolog Programming* (John Wiley, 1993)
  - Cap. 5: "Development of Prolog Programs"