# Tema 5: Metodología y aplicaciones

**José A. Alonso Jiménez**
**Miguel A. Gutiérrez Naranjo**

**Dpto. de Ciencias de la Computación e Inteligencia Artificial**

Universidad de Sevilla

# 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(L2,X,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 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:**

```
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)).
40,587 inferences in 10.20 seconds (3979 Lips)
N = 6765
?- time(fib_2(20,N)).
129 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
```

# 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)).
40,587 inferences in 10.20 seconds (3979 Lips)
N = 6765
Yes
?- time(fib_2(20,N)).
129 inferences in 0.00 seconds (Infinite Lips)
N = 6765
Yes
?- time(fib_3(20,N)).
80 inferences in 0.00 seconds (Infinite Lips)
N = 6765
Yes
```

# 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)).
281 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)).
49 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(100,_N1,_N2),_L)).
20,621 inferences in 0.04 seconds (515525 Lips)
?- time(setof(_N1+_N2,descompone_2(100,_N1,_N2),_L)).
319 inferences in 0.00 seconds (Infinite Lips)
```

# 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**

```
?- time(lista_de_cuadrados_1(100,_L)).
5,448 inferences in 0.01 seconds (544800 Lips)
Yes
?- time(lista_de_cuadrados_2(100,_L)).
502 inferences in 0.00 seconds (Infinite Lips)
Yes
```

- **Programa 3**

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

- **Comparación**

```
?- time(lista_de_cuadrados_3(100,_L)).
414 inferences in 0.00 seconds (Infinite Lips)
Yes
```

# Ordenación

- **Ordenación de una lista**
  - `ordena(L1,L2)` **se verifica si** `L2` **es la lista obtenida ordenanando 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).

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

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

# Ordenación

- **Versión 2:**

```
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:**

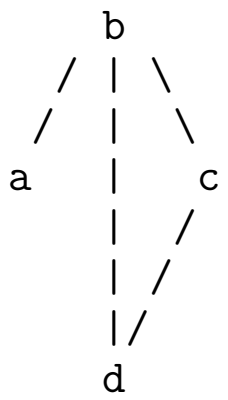
```
?- time(ordena([8,7,6,5,4,3,2,1],_L)).
271,350 inferences in 2.20 seconds (123341 Lips)
Yes
?- time(ord_quicksort([8,7,6,5,4,3,2,1],_L)).
119 inferences in 0.00 seconds (Infinite Lips)
Yes
?- time(sort([8,7,6,5,4,3,2,1],_L)).
3 inferences in 0.00 seconds (Infinite Lips)
Yes
```

- **Verificación**

```
?- ord_quicksort([8,7,6,5,4,3,2,1],_L),
   ordena([8,7,6,5,4,3,2,1],_L).
Yes
```

# Grafos

- **Primera representación de grafos**
  - **Ejemplo de grafo**




  - `conectado(G,X,Y)` **se verifica si** `X` **e** `Y` **son dos nodos conectados en el grafo** `G`**. (Sólo se escribe un hecho por cada arco).**

```
conectado(g1,a,b).
conectado(g1,b,c).
conectado(g1,b,d).
conectado(g1,c,d).
```

# Grafos

- **`adyacente(G,X,Y)` se verifica si los nodos `X` e `Y` son adyacentes en el grafo `G`.**

```
adyacente(G,X,Y) :-
   conectado(G,X,Y).
adyacente(G,X,Y) :-
   conectado(G,Y,X).
```

- **`nodos(G,L)` se verifica si `L` es la lista de los nodos del grafo `G`**

```
nodos(G,L) :- setof(X,Y^adyacente(G,X,Y),L).
```

- **`nodo(N,G)` se verifica si `N` es un nodo de `G`.**

```
nodo(N,G) :-
   nodos(G,L),
   member(N,L).
```

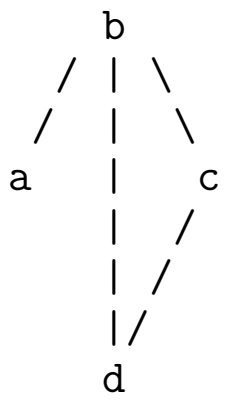
- **`arcos(G,L)` se verifica si `L` es la lista de los arcos del grafo `G`, de forma que el arco de extremo `X` e `Y` se represente por `X-Y`.**

```
arcos(G,L) :- setof(X-Y,conectado(G,X,Y),L).
```

# Grafos

## • Segunda representación

### • Ejemplo de grafo




- `grafo(G,N,A)` **se verifica si** `G` **es el nombre del grafo,** `N` **es la lista de nodos de** `G` **y** `A` **es la lista de arcos de N (cada uno representado mediante un término** `a(X,Y)`**).**

```
grafo(g2,[a,b,c,d],[a(a,b),a(b,c),a(b,d),a(c,d)]).
```

- `adyacente`

```
adyacente(G,X,Y) :-
   grafo(G,_,A),
   (member(a(X,Y),A) ; member(a(Y,X),A)).
```

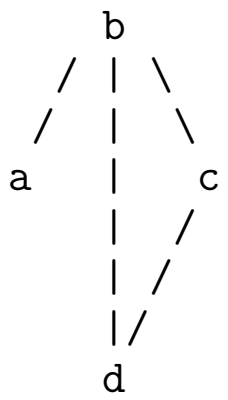
- `nodo`

```
nodo(N,G) :-
   grafo(G,Nodos,_),
   member(N,Nodos).
```

# Grafos

## • Tercera representación

### • Ejemplo de grafo




- `grafo(G,L)` **se verifica si** `G` **es el nombre del grafo y** `L` **es una lista de pares formado por cada uno de los nodos de** `G` **y la lista de sus nodos adyacentes.**

```
grafo(g3,[[a,[b]], [b,[a,c,d]],[c,[b,d]],[d,[b,c]]]).
```

- `adyacente`

```
adyacente(G,X,Y) :-
   grafo(G,L), member([X,L1],L), member(Y,L1).
```

- `nodo`

```
nodo(N,G) :- grafo(G,L), member([N,_],L).
```

# Grafos

- `camino(A,Z,G,C)` **se verifica si** `C` **es un camino en el grafo** `G` **desde el nodo** `A` **al** `Z`**.**
- **Ejemplo**

```
?- camino(a,d,g1,X).
X = [a, b, d] ;
X = [a, b, c, d] ;
No
```

- **Definición de** `camino`

```
camino(A,Z,G,C) :-
   camino_aux(A,[Z],G,C).
```

- `camino_aux(A,CP,G,C)` **se verifica si** `C` **es una camino en** `G` **compuesto de un camino desde** `A` **hasta el primer elemento del camino parcial** `CP` **(con nodos distintos a los de** `CP`**) junto** `CP`**.**

```
camino_aux(A,[A|C1],_,[A|C1]).
camino_aux(A,[Y|C1],G,C) :-
   adyacente(G,X,Y),
   not(member(X,[Y|C1])),
   camino_aux(A,[X,Y|C1],G,C).
```

# Grafos

- **`hamiltoniano(G,H)` se verifica si `H` es un camino hamiltoniano en el grafo `G` (es decir, es un camino en `G` que pasa por todos sus nodos).**
- **Ejemplo**

```
?- hamiltoniano(g1,H).
H = [a, b, c, d] ;
H = [d, c, b, a] ;
H = [c, d, b, a] ;
H = [a, b, d, c] ;
No
```

- **Definición de `hamiltoniano`**

```
hamiltoniano(G,H) :-
   camino(_,_,G,H),
   cubre(H,G).
```

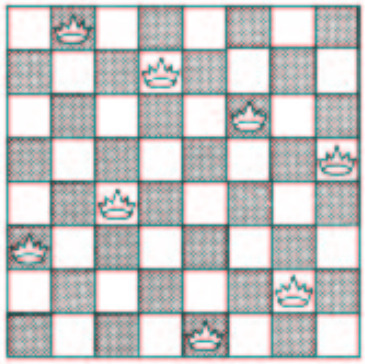
- **`cubre(H,G)` se verifica si `H` cubre el grafo `G` (es decir, todos los nodos de G pertenecen a `H`).**

```
cubre(H,G) :-
   igual_medida(H,G).

igual_medida(H,G) :-
   length(H,MH),
   nodos(G,N),
   length(N,MH).
```

# El problema de las 8 reinas

- **El problema de las ocho reinas**
  - **Colocar 8 reinas en un tablero rectangular de dimensiones 8 por 8 de forma que no se encuentren más de una en la misma línea: horizontal, vertical o diagonal.**



- **Representación 1:**
  - **Sesión:**

```
?- tablero(S), solucion(S).
    S = [[1, 3], [2, 8], [3, 4], [4, 7],
         [5, 1], [6, 6], [7, 2], [8, 5]] ;
    ...
    Yes
```

  - `tablero(L)` **se verifica si** `L` **es una lista de posiciones que representan las coordenadas de 8 reinas en el tablero.**

```
tablero([[1,_],[2,_],[3,_],[4,_],
         [5,_],[6,_],[7,_],[8,_]]).
```

# El problema de las 8 reinas

- **`solucion1(?L)` se verifica si `L` es una lista de pares de números que representan las coordenadas de una solución del problema de las 8 reinas.**

```
solucion1([]).
solucion1([[X,Y]|L]) :-
   solucion1(L),
   member(Y,[1,2,3,4,5,6,7,8]),
   no_ataca([X,Y],L).
```

- **`no_ataca([X,Y],L)` se verifica si la reina en la posición `(X,Y)` no ataca a las reinas colocadas en las posiciones correspondientes a los elementos de la lista `L`.**

```
no_ataca(_,[]).
no_ataca([X,Y],[[X1,Y1]|L]) :-
   X =\= X1,
   Y =\= Y1,
   X-X1 =\= Y-Y1,
   X-X1 =\= Y1-Y,
   no_ataca([X,Y],L).
```

# El problema de las 8 reinas

- **Representación 2:**
  - `solucion2(L)` **se verifica si** `L` **es una lista de 8 números,** `[n1, ..., n8]` **, de forma que si las reinas se colocan en las casillas** `(1, n1), ..., (8, n8)`**, entonces no se atacan entre sí.**

```
solucion2(L) :-
   permutacion([1,2,3,4,5,6,7,8],L),
   segura(L).
```

  - `segura(L)` **se verifica si** `L` **es una lista de** `m` **números** `[n_1,...,n_m]` **tal que las reinas colocadas en las posiciones** `(x, n_1), ..., (x + m, n_m)` **no se atacan entre sí.**

```
segura([]).
segura([X|L]) :-
   segura(L),
   no_ataca(X,L,1).
```

  - `no_ataca(Y,L,D)` **se verifica si** `Y` **es un número,** `L` **es una lista de números** `[n_1,...,n_m]` **y** `D` **es un número tales que las reinas colocada en la posición** `(X,Y)` **no ataca a las colocadas en las posiciones** `(X+D,n_1),...,(X+D+m,n_m)`**.**

```
no_ataca(_,[],_).
no_ataca(Y,[Y1|L],D) :-
   Y1-Y =\= D,
   Y-Y1 =\= D,
   D1 is D+1,
   no_ataca(Y,L,D1).
```

# El problema de las 8 reinas

- **Representación 3:**
  - `solucion3(?L)` **se verifica si** `L` **es una lista de 8 números,** `[n1, ..., n8]`**, de forma que si las reinas se colocan en las casillas** `(1, n1), ..., (8, n8)`**, entonces no se atacan entre sí.**

```
solucion3(L) :-
   sol_aux(L,
           [1,2,3,4,5,6,7,8],
           [1,2,3,4,5,6,7,8],
           [-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7],
           [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]).
```

  - `sol_aux(?L,+Dx,+Dy,+Du,+Dv)` **se verifica si** `L` **es una permutación de los elementos de** `Dy` **de forma que si** `L` **es** `[y_1,...,y_n]` **y** `Dx` **es** `[1,...,n]`**, entonces** `y_j-j (1<=j<=n)` **son elementos distintos de** `Du` **e** `y_j+j (1<=j<=n)` **son elementos distintos de** `Dv`**.**

```
sol_aux([],[],Dy,Du,Dv).
sol_aux([Y|Ys],[X|Dx1],Dy,Du,Dv) :-
   select(Dy,Y,Dy1),
   U is X-Y,
   select(Du,U,Du1),
   V is X+Y,
   select(Dv,V,Dv1),
   sol_aux(Ys,Dx1,Dy1,Du1,Dv1).
```

# El problema de las 8 reinas

- **Comparación**

```
?- time((findall(_S,(tablero(_S), solucion1(_S)),_L),
   length(_L,N))).
% 211,330 inferences in 0.19 seconds (1112263 Lips)
N = 92
Yes

?- time((findall(_S,solucion2(_S),_L),
   length(_L,N))).
% 1,139,741 inferences in 0.82 seconds (1389928 Lips)
N = 92
Yes

?- time((findall(_S,solucion(_S),_L),
   length(_L,N))).
% 120,542 inferences in 0.10 seconds (1205420 Lips)
N = 92
Yes
```

# Bases de datos

- **Relaciones como tablas**
  - **Atributos:** $D_1, \ldots, D_n$
  - **Valores:** $d_1 \in D_1, \ldots, d_n \in D_n$
  - **Relaciones:** $\langle d_1, \ldots, d_n \rangle \in R$
  - **Ejemplos:**

Tabla: CLIENTES

| NOMBRE | ESTADO CIVIL | PROFESION | HIJOS |
|---|---|---|---|
| paco | soltero | médico | 2 |
| ana | soltero | estudiante | 0 |
| maría | casado | médico | 3 |
| luisa | soltero | estudiante | 5 |

Tabla: AFICIONES

| NOMBRE | VACACIONES | DEPORTE | OCIO |
|---|---|---|---|
| ana | playa | voley | cine |
| maria | playa | natacion | tv |
| andres | campo | voley | tv |

# Bases de datos

- **Programa (Datalog)**

```
clientes(paco,soltero,medico,2).
clientes(ana,soltero,estudiante,0).
clientes(maria,casado,medico,3).
clientes(jose,viudo,ebanista,1).
clientes(luisa,soltero,estudiante,5).

aficiones(ana,playa,voley,cine).
aficiones(maria,playa,natacion,tv).
aficiones(andres,campo,voley,tv).
```

- **Selección**
  - **En Bases de datos:**
    **Selecciona de** `CLIENTES` **los** `NOMBRES` **de los clientes tales que** `PROFESION = Estudiante`
  - **En Prolog:**

```
?- clientes(N,_,estudiante,_).
   ana ;
   luisa;
   No

?- findall(_N,clientes(_N,_,estudiante,_),L).
L = [ana, luisa] ;
No
```

# Bases de datos

- **En Bases de datos:**
  **Selecciona de** `AFICIONES` **todas las** `VACACIONES`**.**
- **En Prolog:**

```
?- setof(_V,_N^_D^_O^aficiones(_N,_V,_D,_O),L).
L = [campo, playa] ;
No
```

- **En Bases de datos:**
  **Selecciona de** `CLIENTES` **las entradas tales que** `PROFESION` `= Estudiante` **y crea la tabla** `CLIENTES_EST`

- **En Prolog:**

```
clientes_est(N,E,estudiante,H):-
   clientes(N,E,estudiante,H).
```

- **Sesión**

```
?- clientes_est(N,E,P,H).
N = ana   E = soltero  P = estudiante  H = 0 ;
N = luisa E = soltero  P = estudiante  H = 5 ;
No
```

# Bases de datos

- **Proyección**
  - **En Bases de datos:**
    **Selecciona de `CLIENTES` los pares `NOMBRE`–`PROFESION`.**
  - **En Prolog:**

```
?- clientes(Nombre,_,Profesion,_).
Nombre = paco   Profesion = medico ;
Nombre = ana    Profesion = estudiante ;
Nombre = maria  Profesion = medico ;
Nombre = jose   Profesion = ebanista ;
Nombre = luisa  Profesion = estudiante ;
No

?- findall(_N-_P,clientes(_N,_,_P,_),L).
L = [paco-medico,  ana-estudiante,  maria-medico,
    jose-ebanista, luisa-estudiante] ;
No
```

- **Intersección**
  - **En Bases de datos:**
    **Selecciona de** `CLIENTES` **los** `NOMBRES` **de tales que** `ESTADO CIVIL = soltero` **y** `PROFESION = medico`**.**
  - **En Prolog:**

```
?- clientes(Nombre,soltero,medico,_).
Nombre = paco ;
No
```

  - **En Bases de datos:**
    **Selecciona de** `CLIENTES` **y** `AFICIONES` **los** `NOMBRES` **de tales que** `ESTADO CIVIL = soltero` **y** `VACACIONES = playa`**.**
  - **En Prolog:**

```
?- clientes(N,soltero,_,_),aficiones(N,playa,_,_).
N = ana ;
No
```

# Bases de datos

- **Bases de datos con los mismos atributos**

Tabla: MATRICULA 1999

| ALUMNO | ASIGNATURA | CALIFICACION |
|---|---|---|
| Paco | DBD | Notable |
| Maria | PD | Notable |
| Ana | IA1 | N.P. |
| Jose | EATP | Suspenso |
| Laura | PD | Sobresaliente |

Tabla: MATRICULA 2000

| ALUMNO | ASIGNATURA | CALIFICACION |
|---|---|---|
| Luis | TCO | Notable |
| Maria | IA1 | Aprobado |
| Ana | IA1 | N.P. |
| Jose | EATP | Aprobado |
| Jaime | PD | Notable |

```
matricula_99(paco,dbd,notable).
matricula_99(maria,pd,notable ).
matricula_99(ana,ia1,np).
matricula_99(jose,eatp,suspenso).
matricula_99(laura,pd,sobresaliente).

matricula_00(luis,tco,notable).
matricula_00(maria,ia1,aprobado).
matricula_00(ana,ia1,np).
matricula_00(jose,eatp,aprobado).
matricula_00(jaime,pd,notable).
```

# Bases de datos

## • Unión

### • Programa

```
n_union(Nombre, Asignatura, Calificacion):-
   matricula_99(Nombre, Asignatura, Calificacion).
n_union(Nombre, Asignatura, Calificacion):-
   matricula_00(Nombre, Asignatura, Calificacion).
```

### • Sesión:

```
?- n_union(N,A,C).
N = paco     A = dbd     C = notable ;
N = maria    A = pd      C = notable ;
N = ana      A = ia1     C = np ;
N = jose     A = eatp    C = suspenso ;
N = laura    A = pd      C = sobresaliente ;
N = luis     A = tco     C = notable ;
N = maria    A = ia1     C = aprobado ;
N = ana      A = ia1     C = np ;
N = jose     A = eatp    C = aprobado ;
N = jaime    A = pd      C = notable ;
```

# Bases de datos

## • Intersección

### • Programa

```
n_interseccion(Nombre, Asignatura, Calificacion):-
   matricula_99(Nombre, Asignatura, Calificacion),
   matricula_00(Nombre, Asignatura, Calificacion).
```

### • Sesión:

```
?- n_interseccion(N,A,C).
N = ana    A = ia1   C = np ;
No
```

## • Diferencia

### • Programa

```
n_diferencia(Nombre, Asignatura, Calificacion):-
   matricula_99(Nombre, Asignatura, Calificacion),
   not(matricula_00(Nombre, Asignatura, Calificacion)).
```

### • Sesión:

```
?- n_diferencia(N,A,C).
N = paco   A = dbd   C = notable ;
N = maria  A = pd    C = notable ;
N = jose   A = eatp  C = suspenso ;
N = laura  A = pd    C = sobresaliente ;
No
```

# Algebra relacional

- **Combinación de operaciones básicas sobre bases de datos**
- **Ejemplo**
  - **Relaciones iniciales**

Tabla R

| A1 | A2 | A3 |
|---|---|---|
| a | b | c |
| f | d | h |
| f | e | h |

Tabla S

| B1 | B2 |
|---|---|
| d | e |
| g | d |
| f | m |

Tabla T

| B1 | B2 |
|---|---|
| b | e |
| d | e |
| g | m |

  - **Programa**

```
r(a,b,c).
r(f,d,h).
r(f,e,h).

s(d,e).
s(g,d).
s(f,m).

t(b,e).
t(d,e).
t(g,m).
```

# Algebra relacional

- **A partir de la tabla `R` crea la relación binaria `R1` con los atributos `A1` y `A3`. Sólo tomaremos aquellos individuos tales que el valor correspondiente a `A2` sea un valor de `B2` en la tabla `S`, pero no sea un valor de `B2` en la tabla `T`.**

- **En Base de datos:**
  - **Seleccionar `B2` en `S`**
  - **Seleccionar `B2` en `T`**
  - **Hacer la diferencia (si es posible)**
  - **Seleccionar `A2` en `R`**
  - **Hacer la insersección**
  - **Hacer la proyección**

- **En Prolog:**
  - **Programa:**

```
r1(A,B):-
   r(A,C,B),
   s(_,C),
   not(t(_,C)).
```

  - **Sesión**

```
?- r1(A,B).
   A = f  B = h ;
No
```

# Bibliografía


- Bratko, I. *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
  - Cap. 4: "Using Structures: Example Programs"
  - Cap. 8: "Programming Style and Technique"
  - Cap. 9: "Operations on Data Structures"
- 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`
- Sterling,, L. y Shapiro, E. *The Art of Prolog (2nd edititon)* (The MIT Press, 1994)
  - Cap. 2 "Database programming"
- Van Le, T. *Techniques of Prolog Programming* (John Wiley, 1993)
  - Cap. 5: "Development of Prolog Programs"