Tema 8: Aplicaciones de PD

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

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
```

Bases de datos

- **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)).
```

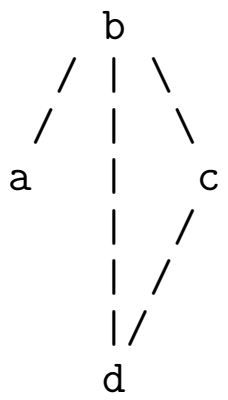
  - **Sesión**

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

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`.**

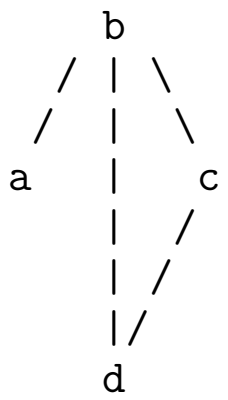
```
nodo(N,G) :-
   nodos(G,L),
   member(N,L).
```

- **`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)).
```

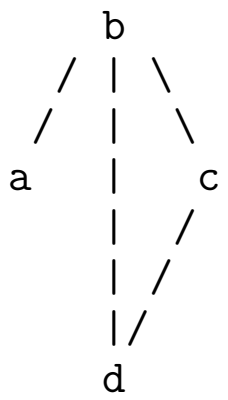
  - nodo

```
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).
```

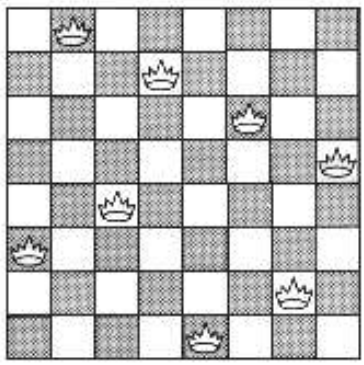
- **`cubre(H,G)` se verifica si `H` cubre el grafo `G` (es decir, todos los nodos de G pertenecen a `H`).**

```
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, 4], [2, 2], [3, 7], [4, 3],
         [5, 6], [6, 8], [7, 5], [8, 1]] ;
    ...
    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).
```

Bibliografía

- Bratko, I. *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
  - Cap. 4: "Using Structures: Example Programs"
  - Cap. 9: "Operations on Data Structures"
- Sterling,, L. y Shapiro, E. *The Art of Prolog (2nd edititon)* (The MIT Press, 1994)
  - Cap. 2 "Database programming"