Tema 3: Estructuras

Grupo de Programación Declarativa
{José A. Alonso, Andrés Cordón y Miguel A. Gutiérrez}

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Objetos estructurados

- **Segmentos verticales y horizontales**
  - **Representación:**

    ```
    punto(X,Y)
    seg(P1,P2)
    ```

  - `horizontal(S)` **se verifica si el segmento** `S` **es horizontal**
  - `vertical(S)` **se verifica si el segmento** `S` **es vertical. Por ejemplo,**

    ```
    vertical(seg(punto(1,2),punto(1,3)))  =>  Sí
    vertical(seg(punto(1,2),punto(4,2)))  =>  No
    horizontal(seg(punto(1,2),punto(1,3)))  =>  No
    horizontal(seg(punto(1,2),punto(4,2)))  =>  Sí
    ```

  - **Programa:** `ver_hor.pl`

    ```
    horizontal(seg(punto(X,Y),punto(X1,Y))).
    vertical(seg(punto(X,Y),punto(X,Y1))).
    ```

Objetos estructurados

- **¿Es vertical el segmento que une los puntos** `(1,1)` **y** `(1,2)`**?.**

```
?- vertical(seg(punto(1,1),punto(1,2))).
Yes
```

- **¿Es vertical el segmento que une los puntos** `(1,1)` **y** `(2,2)`**?.**

```
?- vertical(seg(punto(1,1),punto(2,2))).
No
```

- **¿Hay algún** `Y` **tal que el segmento que une los puntos** `(1,1)` **y** `(2,Y)` **sea vertical?.**

```
?- vertical(seg(punto(1,1),punto(2,Y))).
No
```

- **¿Hay algún** `X` **tal que el segmento que une los puntos** `(1,2)` **y** `(X,3)` **sea vertical?.**

```
?- vertical(seg(punto(1,2),punto(X,3))).
X = 1 ;
No
```

- **¿Hay algún** `Y` **tal que el segmento que une los puntos** `(1,1)` **y** `(2,Y)` **sea horizontal?.**

```
?- horizontal(seg(punto(1,1),punto(2,Y))).
Y = 1 ;
No
```

Objetos estructurados

- **¿Para qué puntos el segmento que comienza en `(2,3)` es vertical?.**

```
?- vertical(seg(punto(2,3),P)).
P = punto(2, _G459) ;
No
```

- **¿Hay algún segmento que sea horizontal y vertical?.**

```
?- vertical(S),horizontal(S).
S = seg(punto(_G444, _G445),
        punto(_G444, _G445)) ;
No
?- vertical(_),horizontal(_).
Yes
```

- `no_degenerado(S)` **se verifica si** `S` **es un segmento no degenerado (es decir, que no se reduce a un punto). Por ejemplo,**

```
no_degenerado(seg(punto(1,2),punto(2,1)) => Yes
no_degenerado(seg(punto(1,2),punto(1,2)) => No
```

- **Programa:**

```
no_degenerado(seg(P1,P2)) :- P1 \== P2.
```

- **¿Todos los segmentos que son horizontales y verticales son degenerados?.**

```
?- vertical(S),horizontal(S),no_degenerado(S).
No
```

Recuperación de la información

- **Descripción de la familia 1**
  - **el padre es Tomás García Pérez, nacido el 7 de Mayo de 1950, trabaja de profesor y gana 75 euros diarios**
  - **la madre es Ana López Ruiz, nacida el 10 de marzo de 1952, trabaja de médica y gana 100 euros diarios**
  - **el hijo es Juan García López, nacido el 5 de Enero de 1970, estudiante**
  - **la hija es María García López, nacida el 12 de Abril de 1972, estudiante**

```
familia(
  persona([tomas,garcia,perez],
          fecha(7,mayo,1950),
          trabajo(profesor,75)),
  persona([ana,lopez,ruiz],
          fecha(10,marzo,1952),
          trabajo(medica,100)),
  [ persona([juan,garcia,lopez],
            fecha(5,enero,1970),
            estudiante),
    persona([maria,garcia,lopez],
            fecha(12,abril,1972),
            estudiante) ]).
```

Recuperación de la información

- **Descripción de la familia 2**
  - **el padre es José Pérez Ruiz, nacido el 6 de Marzo de 1953, trabaja de pintor y gana 150 euros/día**
  - **la madre es Luisa Gálvez Pérez, nacida el 12 de Mayo de 1954, es médica y gana 100 euros/día**
  - **un hijo es Juan Luis Pérez Pérez, nacido el 5 de Febrero de 1980, estudiante**
  - **una hija es María José Pérez Pérez, nacida el 12 de Junio de 1982, estudiante**
  - **otro hijo es José María Pérez Pérez, nacido el 12 de Julio de 1984, estudiante**

```
familia(
  persona([jose,perez,ruiz],
          fecha(6,marzo,1953),
          trabajo(pintor,150)),
  persona([luisa,galvez,perez],
          fecha(12,mayo,1954),
          trabajo(medica,100)),
  [ persona([juan_luis,perez,perez],
            fecha(5,febrero,1980),
            estudiante),
    persona([maria_jose,perez,perez],
            fecha(12,junio,1982),
            estudiante),
    persona([jose_maria,perez,perez],
            fecha(12,julio,1984),
            estudiante) ]).
```

Recuperación de la información

- **¿Existe alguna familia sin hijos?**

```
?- familia(_,_,[]).
No
```

- **¿Existe alguna familia con un hijo?**

```
?- familia(_,_,[_]).
No
```

- **¿Existe alguna familia con dos hijos?**

```
?- familia(_,_,[_,_]).
Yes
```

- **¿Existe alguna familia con tres hijos?**

```
?- familia(_,_,[_,_,_]).
Yes
```

- **¿Existe alguna familia con cuatro hijos?**

```
?- familia(_,_,[_,_,_,_]).
No
```

- **Buscar los nombres de los padres de familia con tres hijos**

```
?- familia(persona(NP,_,_),_,[_,_,_]).
NP = [jose, perez, ruiz] ;
No
```

Recuperación de la información

- **Hombres casados**
  - `casado(X)` **se verifica si** `X` **es un hombre casado**

```
casado(X) :- familia(X,_,_).
```

  - **Consulta**

```
?- casado(X).
X = persona([tomas, garcia, perez],
            fecha(7, mayo, 1950),
            trabajo(profesor, 75)) ;
X = persona([jose, perez, ruiz],
            fecha(6, marzo, 1953),
            trabajo(pintor, 150)) ;
No
```

- **Mujeres casadas**
  - `casada(X)` **se verifica si** `X` **es una mujer casada**

```
casada(X) :- familia(_,X,_).
```

  - **Consulta**

```
?- casada(X).
X = persona([ana, lopez, ruiz],
            fecha(10, marzo, 1952),
            trabajo(medica, 100)) ;
X = persona([luisa, galvez, perez],
            fecha(12, mayo, 1954),
            trabajo(medica, 100)) ;
No
```

Recuperación de la información

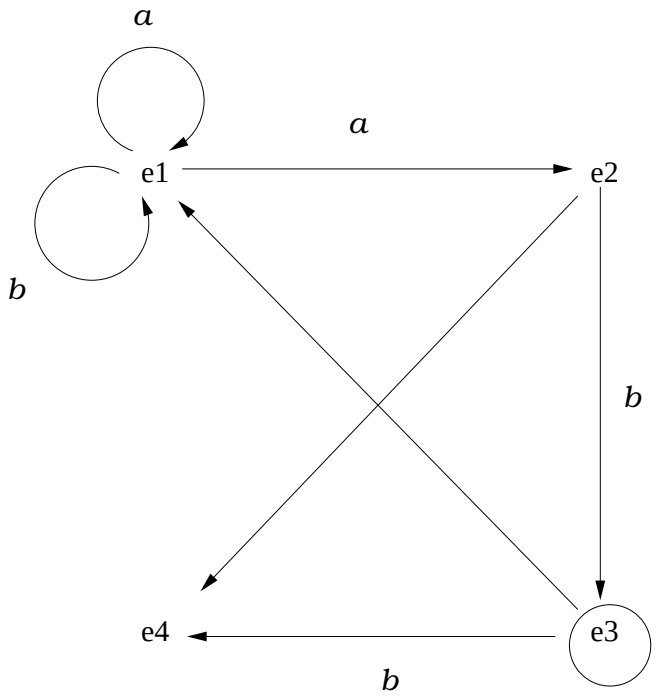
- **Nombres de las mujeres casadas que trabajan**

```
?- casada(persona([N,_,_],_,trabajo(_,_))).
N = ana ;
N = luisa ;
No
```

- **Conceptos**
  - **Uso de objetos estructurados**
  - **Prolog como lenguaje de consulta de bases de datos.**
  - **Bases de datos como conjuntos de hechos**
  - **Selectores**
  - **Abstracción de datos**

Simulación de un autómata

- **Autómata no determinista (con estado final e3)**



- **Representación del autómata (`automata.pl`):**
  - `final(E)` **se verifica si** `E` **es el estado final.**

```
final(e3).
```

  - `trans(E1,X,E2)` **se verifica si se puede pasar del estado** `E1` **al estado** `E2` **usando la letra** `X`

```
trans(e1,a,e1).
trans(e1,a,e2).
trans(e1,b,e1).
trans(e2,b,e3).
trans(e3,b,e4).
```

Simulación de un autómata

- `nulo(E1,E2)` **se verifica si se puede pasar del estado** `E1` **al estado** `E2` **mediante un movimiento nulo.**

```
nulo(e2,e4).
nulo(e3,e1).
```

- `acepta(E,L)` **se verifica si el autómata, a partir del estado** `E` **acepta la lista** `L`
  - **Ejemplo:**

```
acepta(e1,[a,a,a,b]) => Sí
acepta(e2,[a,a,a,b]) => No
```

  - **Definición:**

```
acepta(E,[]) :-
   final(E).
acepta(E,[X|L]) :-
   trans(E,X,E1),
   acepta(E1,L).
acepta(E,L) :-
   nulo(E,E1),
   acepta(E1,L).
```

Simulación de un autómata

- **Determinar si el autómata acepta la lista `[a,a,a,b]`**

```
?- acepta(e1,[a,a,a,b]).
Yes
```

- **Determinar los estados a partir de los cuales el autómata acepta la lista `[a,b]`**

```
?- acepta(E,[a,b]).
E=e1 ;
E=e3 ;
No
```

- **Determinar las palabras de longitud 3 aceptadas por el autómata a partir del estado `e1`**

```
?- acepta(e1,[X,Y,Z]).
X = a
Y = a
Z = b ;
X = b
Y = a
Z = b ;
No
```

Problema del mono

- **Descripción:**
  - **Un mono se encuentra en la puerta de una habitación. En el centro de la habitación hay un plátano colgado del techo. El mono está hambriento y desea coger el plátano, pero no lo alcanza desde el suelo. En la ventana de la habitación hay una silla que el mono puede usar. El mono puede realizar las siguientes acciones: pasear de un lugar a otro de la habitación, empujar la silla de un lugar a otro de la habitación (si está en el mismo lugar que la silla), subirse en la silla (si está en el mismo lugar que la silla) y coger el plátano (si está encima de la silla en el centro de la habitación).**
- **Representación:**
  - **Representación:** `estado(PM,AM,PS,MM)`
    `PM` **posición del mono (puerta, centro o ventana)**
    `AM` **apoyo del mono (suelo o silla)**
    `PS` **posición de la silla (puerta, centro o ventana)**
    `MM` **mano del mono (con o sin) plátano**
  - `movimiento(E1,A,E2)` **se verifica si** `E2` **es el estado que resulta de realizar la acción** `A` **en el estado** `E1`

Problema del mono

```
movimiento(estado(centro,silla,centro,sin),
           coger,
           estado(centro,silla,centro,con)).
movimiento(estado(X,suelo,X,U),
           subir,
           estado(X,silla,X,U)).
movimiento(estado(X1,suelo,X1,U),
           empujar(X1,X2),
           estado(X2,suelo,X2,U)).
movimiento(estado(X,suelo,Z,U),
           pasear(X,Z),
           estado(Z,suelo,Z,U)).
```

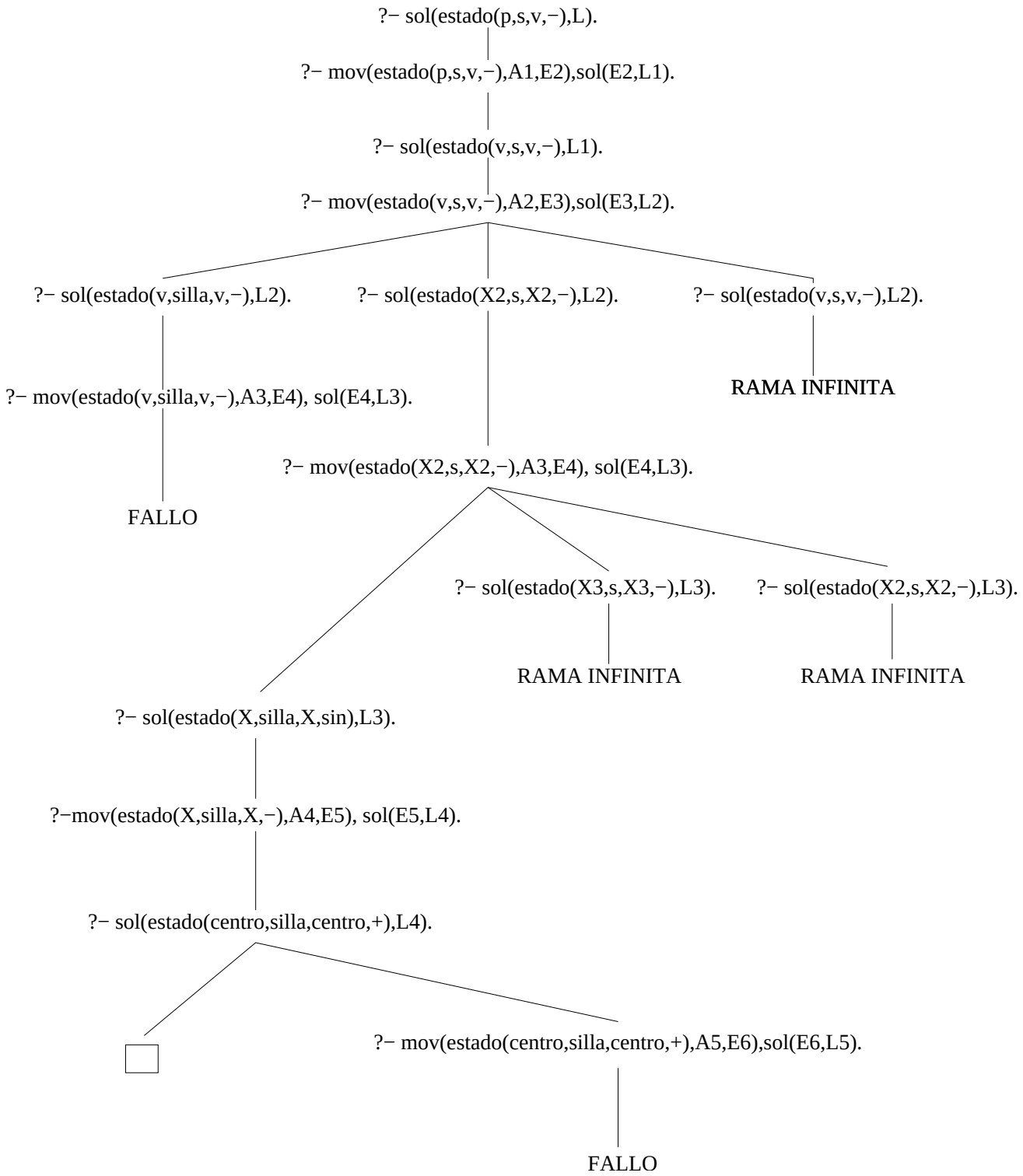
- **Solución:**
  - `solución(E,S)` **se verifica si** `S` **es una sucesión de acciones que aplicadas al estado** `E` **permiten al mono coger el plátano.**
  - **Ejemplo:**

```
?- solucion(estado(puerta,suelo,ventana,sin),L).
L = [pasear(puerta, ventana),
     empujar(ventana, centro),
     subir,
     coger] ;
No
```

  - **Definición**

```
solucion(estado(_,_,_,con),[]).
solucion(E1,[A|L]) :-
   movimiento(E1,A,E2),
   solucion(E2,L).
```

Problema del mono



Bibliografía

- Alonso, J.A. y Borrego, J. *Deducción automática (Vol. 1: Construcción lógica de sistemas lógicos)* (Ed. Kronos, 2002)
  `www.cs.us.es/~jalonso/libros/da1-02.pdf`
  - Cap. 2: Introducción a la programación lógica con Prolog.
- Bratko, I. *Prolog Programming for Artificial Intelligence (3 ed.)* (Addison–Wesley, 2001)
  - Cap. 2: "Syntax and meaning of Prolog programs"
  - Cap. 4: "Using Structures: Example Programs"
- Van Le, T. *Techniques of Prolog Programming* (John Wiley, 1993)
  - Cap. 2: "Declarative Prolog programming".
- Clocksin, W.F. y Mellish, C.S. *Programming in Prolog (Fourth Edition)* (Springer Verlag, 1994)
  - Cap. 3: "Using Data Structures"