# *Programación declarativa (2006–07)*

# *Tema 9: Programación lógica con restricciones*

José A. Alonso Jiménez

Andrés Cordón Franco

Grupo de Lógica Computacional

Dpto. Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

# Restricciones sobre números reales: CLP(R)

- Uso de la biblioteca CLP(R)

```
?- use_module(library(clpr)).
Yes
```

- Diferencia entre objetivo y restricción:

```
?- 1+X=5.
No
?- {1+X=5}.
X = 4.0
Yes
```

- Restricciones aritméticas en CLP(R):
  - *restricciones* := *{restricción_1, restricción_2, . . .}*
  - *restricción* := *expresión_1 operador expresión_2*
  - *expresión_1* y *expresión_2* son expresiones aritméticas
  - *operador* es uno de los siguientes: `=`, `=\=`, `<`, `=<`, `>`, `>=`

# CLP(R): ecuaciones e inecuaciones

- Ejemplo de ecuaciones e inecuaciones:

```
?- {3*X-2*Y=6, 2*Y=X}.
X = 3.0
Y = 1.5

?- {Z=<X-2, Z=<6-X, Z+1=2}.
{X=<5.0}
{X>=3.0}
Z = 1.0

?- {X>0, X+2<0}.
No
```

## CLP(R): Programa

- Ejemplo de programa en Prolog y en CLP(R)
  - Especificación:
    `convierte(-C,+F)` se verifica si `C` son los grados centígrados correspondientes a `F` grados Fahrenheit; es decir, $C = \frac{(F-32)*5}{9}$.
  - Programa Prolog

```
convierte_1(C,F) :-
    C is (F-32)*5/9.
```

  - Sesión con el programa Prolog

```
?- convierte_1(C,95).
C = 35

?- convierte_1(35,F).
ERROR: is/2: Arguments are not sufficiently
instantiated
```

# CLP(R): Programa

- Ejemplo de programa en Prolog y en CLP(R)
  - Programa CLP(R)

```
:- use_module(library(clpr)).

convierte_2(C,F) :-
    {C = (F-32)*5/9}.
```

  - Sesión con el programa CLP(R)

```
?- convierte_2(C,95).
C = 35.0

?- convierte_2(35,F).
F = 95.0

?- convierte_2(C,F).
{F=32.0+1.8*C}
```

# CLP(R): Optimización

- Optimización con `minimize/1` y `maximize/1`:

```
?- {X=<5}, maximize(X).
X = 5.0

?- {X=<5, 2=<X}, minimize(2*X+3).
X = 2.0

?- {3=<X, X+1=<Y+2, Y=<9, Z=X+Y}, minimize(Z).
X = 3.0     Y = 2.0     Z = 5.0

?- {X+Y=<4}, maximize(X+Y).
{Y=4.0-X}

?- {X=<5}, minimize(X).
No
```

## CLP(R): Optimización

- Optimización con `sup/2` e `inf/2`:

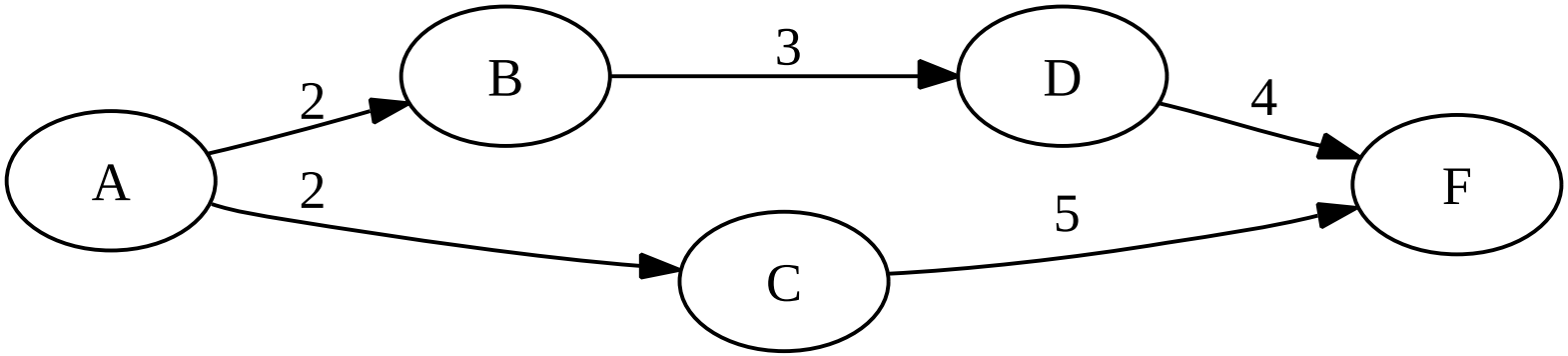
```
?- {2=<X, X=<5}, inf(X,I), sup(X,S).
{X>=2.0}
{X=<5.0}
I = 2.0
S = 5.0

?- {3=<X, X+1=<Y+2, Y=<9, Z=X+Y},
   inf(Z,I), sup(Z,S).
{X-Y=<1.0}
{Y=<9.0}
{X>=3.0}

I = 5.0
S = 19.0
```

# CLP(R): Planificación de tareas

- Problema: Calcular el menor tiempo necesario para realizar las tareas A, B, C y D teniendo en cuenta que los tiempos empleados en cada una son 2, 3, 5 y 4, respectivamente, y además que A precede a B y a C y que B precede a D.




- Plan óptimo:

```
?- {Ta>=0, Ta+2=<Tb, Ta+2=<Tc, Tb+3=<Td,
    Tc+5=<Tf, Td+4=<Tf},
   minimize(Tf).
{Tc=<4.0}  {Tc>=2.0}
Ta=0.0   Tb=2.0   Td=5.0   Tf=9.0
```

# Restricciones sobre números racionales: CLP(Q)

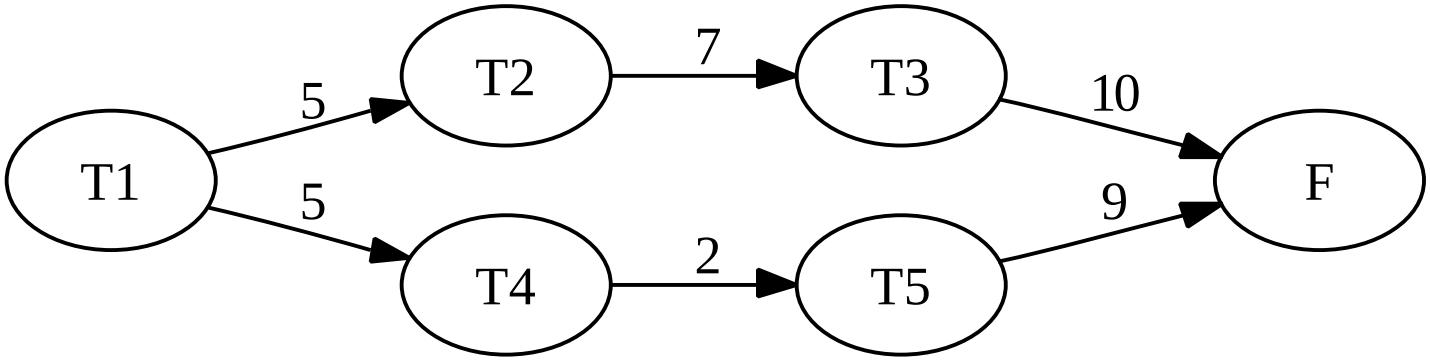
- CLP sobre números racionales: CLP(Q)

```
?- use_module(library(clpq)).
Yes

?- {X=2*Y, Y=1-X}.
X = 2 rdiv 3
Y = 1 rdiv 3
```

# CLP(Q): Planificación de tareas

- Especificación de un problema mediante `tareas` y `precede`




- `tareas(+LTD)` se verifica si `LTD` es la lista de los pares `T/D` de las tareas y sus duraciones.

```
tareas([t1/5,t2/7,t3/10,t4/2,t5/9]).
```

- `precede(+T1,+T2)` se verifica si la tarea `T1` tiene que preceder a la `T2`.

```
precede(t1,t2).
precede(t1,t4).
precede(t2,t3).
precede(t4,t5).
```

# CLP(Q): Planificación de tareas

Planificador

- Biblioteca CLP(Q)

```
:- use_module(library(clpq)).
```

- `planificación(P,TP)` se verifica si `P` es el plan (esto es una lista de elementos de la forma tarea/inicio/duración) para realizar las tareas en el menor tiempo posible y `TP` es dicho tiempo. Por ejemplo,

```
?- planificación(P,TP).
P = [t1/0/5,t2/5/7,t3/12/10,t4/_G3235/2,t5/_G3710/9]
TP = 22
?- planificación([_,_,_,_/X/_,_/Y/_],TP).
{X>=5}
{X-Y=< -2}
{Y=<13}
TP = 22
```

## CLP(Q): Planificación de tareas

- Definición

```
planificación(P,TP) :-
    tareas(LTD),
    restricciones(LTD,P,TP),
    minimize(TP).
```

- `restricciones(LTD,P,TP)` se verifica si `P` es un plan para realizar las tareas de `LTD` cumpliendo las restricciones definidas por `precedencia/2` y `TP` es el tiempo que se necesita para ejecutar el plan `P`.

```
restricciones([],[],_TP).
restricciones([T/D | RLTD],[T/I/D | RTID],TP) :-
    {I >= 0, I + D =< TP},
    restricciones(RLTD,RTID,TP),
    restricciones_aux(T/I/D,RTID).
```

## CLP(Q): Planificación de tareas

- `restricciones_aux(TID,LTID)` se verifica si el triple tarea–inicio–duración `TID` es consistente con la lista de triples tarea–inicio–duración `LTID`.

```
restricciones_aux(_,[]).
restricciones_aux(T/I/D, [T1/I1/D1 | RTID]) :-
    ( precede(T,T1) -> {I+D =< I1}
    ; precede(T1,T) -> {I1+D1 =< I}
    ; true ),
    restricciones_aux(T/I/D,RTID).
```

## Restricciones sobre dominios finitos: CLP(FD)

- Módulo de restricciones de dominios finitos.

```
?– use_module(library('clp/bounds')).
Yes
```

- `indomain(X)` asigna un valor a la variable de dominio acotado `X`, en orden creciente. Por ejemplo,

```
?– X in 1..2, Y in 1..10, X+7 #< Y,
   indomain(X), indomain(Y).
X = 1
Y = 9 ;
X = 1
Y = 10 ;
X = 2
Y = 10 ;
No
```

## CLP(FD): Instanciación y diferentes

- `label(L)` se verifica si existe una asignación que verifica las restricciones de las variables de la lista `L`. Por ejemplo,

```
?- L=[X,Y], L in 1..2, label(L).
L = [1, 1]   X = 1   Y = 1 ;
L = [1, 2]   X = 1   Y = 2 ;
L = [2, 1]   X = 2   Y = 1 ;
L = [2, 2]   X = 2   Y = 2 ;
No
```

- `all_different(L)` se verifica si todos las variables de la lista `L` tienen valores distintos. Por ejemplo,

```
?- L=[X,Y], L in 1..2, all_different(L), label(L).
L = [1, 2]   X = 1   Y = 2 ;
L = [2, 1]   X = 2   Y = 1 ;
No
```

# CLP(FD): Restricciones de dominio y aritméticas

- Restricciones de dominio y aritméticas:

```
?- X in 1..5, Y in 0..4, X #< Y, Z #= X+Y+1,
     findall(X-Y-Z,label([X,Y,Z]),L).
L = [1-2-4, 1-3-5, 1-4-6, 2-3-6, 2-4-7, 3-4-8]

?- [X,Y] in 1..2, Z #=X+Y,
     findall(X-Y-Z,label([X,Y,Z]),L).
L = [1-1-2, 1-2-3, 2-1-3, 2-2-4]
```

- Tipos de restricciones en CLP(FD):
  - de dominio de variables: <variable> in <Mínimo>..<Máximo>
  - de dominio de lista: <lista> in <Mínimo>..<Máximo>
  - aritmética: <expresión 1> <relación> <expresión 2> con <relación> en #=, #\=, #<, #>, #=<, #>=.

## CLP(FD): Criptoaritmética

- `solución([S,E,N,D],[M,O,R,E],[M,O,N,E,Y])` se verifica si cada una de las letras se sustituye por un dígito distinto de forma que `SEND+MORE=MONEY`. Por ejemplo,

```
?- solución(L1,L2,L3).
L1 =   [9,5,6,7]
L2 =   [1,0,8,5]
L3 = [1,0,6,5,2] ;
No
```

## CLP(FD): Criptoaritmética

- Programa

```
:- use_module(library('clp/bounds')).

solución([S,E,N,D],[M,O,R,E],[M,O,N,E,Y]) :-
    Vars = [S,E,N,D,M,O,R,Y],
    Vars in 0..9,
    all_different(Vars),
            1000*S+100*E+10*N+D
          + 1000*M+100*O+10*R+E #=
    10000*M+1000*O+100*N+10*E+Y,
    M #\= 0,
    S #\= 0,
    label(Vars).
```

## CLP(FD): Problema de las N reinas

- El problema de las N reinas consiste en colocar N reinas en un tablero rectangular de dimensiones N por N de forma que no se encuentren más de una en la misma línea: horizontal, vertical o diagonal.
- `solución(N,L)` se verifica si `L` es una solución del problema de las `N` reinas. Por ejemplo,

```
?- solución(4,L).
L = [2, 4, 1, 3] ;
L = [3, 1, 4, 2] ;
No
```

- Uso de dominios finitos:

```
:- use_module(library('clp/bounds')).
```

## CLP(FD): Problema de las N reinas

- Definición de `solución`:

```
solución(N,L) :-
    length(L,N),
    L in 1..N,
    all_different(L),
    segura(L),
    label(L).
```

- `segura(L)` se verifica si las reinas colocadas en las ordenadas `L` no se atacan diagonalmente.

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

## CLP(FD): Problema de las N reinas

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

```
no_ataca(_Y,[],_).
no_ataca(Y1,[Y2|L],D) :-
    Y1-Y2 #\= D,
    Y2-Y1 #\= D,
    D1 is D+1,
    no_ataca(Y1,L,D1).
```

## CLP(FD): Optimización

- `mejor(X)` restringe los valores de `X` a su menor valor. Por ejemplo,

```
?- X in 1..6, V #= X*(X-6), mejor(V).
X = 3   V = -9
?- X in 1..6, V #= X*(6-X), mejor(V).
X = 6   V = 0
```

- Definición de `mejor(X)`

```
mejor(X) :-
    findall(X, indomain(X), L),
    member(Y, L),
    not((member(Z, L), Z < Y)),
    X #= Y.
```

# CLP(FD): Planificación de tareas

- Problema: Calcular el tiempo necesario para realizar las tareas A, B, C y D teniendo en cuenta que los tiempos empleados en cada una son 2, 3, 5 y 4, respectivamente, y además que A precede a B y a C y que B precede a D.

```
?- [Ta,Tb,Tc,Td,Tf] in 0..10,
   Ta+2 #=< Tb,   Ta+2 #=< Tc,   Tb+3 #=< Td,
   Tc+5 #=< Tf,   Td+4 #=< Tf,
   findall(Ta, indomain(Ta), Da),
   findall(Tb, indomain(Tb), Db),
   findall(Tc, indomain(Tc), Dc),
   findall(Td, indomain(Td), Dd),
   findall(Tf, indomain(Tf), Df).

Da=[0,1]    Db=[2,3]    Dc=[2,3,4,5]
Dd=[5,6]    Df=[9,10]
```

# CLP(FD): Planificación de tareas

- Traza:

| $Paso$ | | $Ta$ | $Tb$ | $Tc$ | $Td$ | $Tf$ |
|---|---|---|---|---|---|---|
| | | $0..10$ | $0..10$ | $0..10$ | $0..10$ | $0..10$ |
| $1$ | $Ta + 2 \leq Tb$ | $0..8$ | $2..10$ | | | |
| $2$ | $Tb + 3 \leq Td$ | | $2..7$ | | $5..10$ | |
| $3$ | $Td + 4 \leq Tf$ | | | | $5..6$ | $9..10$ |
| $4$ | $Tb + 3 \leq Td$ | | $2..3$ | | | |
| $5$ | $Ta + 2 \leq Tb$ | $0..1$ | | | | |
| $6$ | $Ta + 2 \leq Tc$ | | | $2..10$ | | |
| $7$ | $Tc + 5 \leq Tf$ | | | $2..5$ | | |

# CLP(FD): Planificación óptima de tareas

```
?- [Ta,Tb,Tc,Td,Tf] in 0..10,
   Ta+2 #=< Tb,   Ta+2 #=< Tc,   Tb+3 #=< Td,
   Tc+5 #=< Tf,   Td+4 #=< Tf,
   mejor(Tf),
   findall(Ta, indomain(Ta), Da),
   findall(Tb, indomain(Tb), Db),
   findall(Tc, indomain(Tc), Dc),
   findall(Td, indomain(Td), Dd).

Ta=0   Tb=2   Tc=_G176   Td=5   Tf=9
Da = [0]
Db = [2]
Dc = [2, 3, 4]
Dd = [5]
```

# CLP(FD): Problema de los músicos

- Problema de los músicos: Se sabe que
  1. Una banda está compuesta por tres músicos de distintos paises y que tocan distintos instrumentos.
  2. El pianista toca primero.
  3. Juan toca el saxo y toca antes que el cubano.
  4. Marcos es ruso y toca antes que el flautista.
  5. Hay un músico coreano.
  6. Un músico se llama Luis.

  Determinar el nombre, el país y el instrumento que toca cada uno de los músicos de la banda.

- Solución:

| | Nombre | País | Instrumento |
|---|---|---|---|
| 1 | Marco | Rusia | Piano |
| 2 | Juan | Corea | Saxo |
| 3 | Luis | Cuba | Flauta |

## CLP(FD): Problema de los músicos

```
:- use_module(library('clp/bounds')).
músicos(Juan,Marco,Luis,Cuba,Rusia,
        Corea,Piano,Saxo,Flauta) :-
    [Juan,Marco,Luis,Cuba,Rusia,
     Corea,Piano,Saxo,Flauta] in 1..3,
    all_different([Juan,Marco,Luis]),
    all_different([Cuba,Rusia,Corea]),
    all_different([Piano,Saxo,Flauta]),
    Piano #=1,          % 2
    Juan #= Saxo,       % 3a
    Juan #< Cuba,       % 3b
    Marco #= Rusia,     % 4a
    Marco #< Flauta,    % 4b
    label([Juan,Marco,Luis,Cuba,Rusia,
           Corea,Piano,Saxo,Flauta]).
```

## CLP(FD): Problema de los músicos

```
?- músicos(Juan, Marco, Luis, Cuba,
           Rusia, Corea, Piano, Saxo, Flauta).
Juan = 2
Marco = 1
Sal = 3
Cuba = 3
Rusia = 1
Corea = 2
Piano = 1
Saxo = 2
Flauta = 3 ;

No
```

# CLP(FD): Problema de los músicos

Segunda representación

```
:- use_module(library('clp/bounds')).

músicos(S) :-
    % Variables:
    S = [Nombre, País, Instrumento],
    Nombre = [Juan, Marco, Luis],
    País = [Cuba, Rusia, Corea],
    Instrumento = [Piano, Saxo, Flauta],
    flatten(S, L),

    % Dominios:
    L in 1..3,
```

# CLP(FD): Problema de los músicos

```
% Restricciones:
 all_different(Nombre),
 all_different(País),
 all_different(Instrumento),
 Piano #=1,                 % 2
 Juan #= Saxo,              % 3a
 Juan #< Cuba,              % 3b
 Marco #= Rusia,            % 4a
 Marco #< Flauta,           % 4b

% Instanciación:
 label(L).
```

# CLP(FD): Problema de los músicos

```
?- músicos(S).
S = [[2, 1, 3], [3, 1, 2], [1, 2, 3]] ;
No
```

Por tanto,

```
Juan   = 2
Marco  = 1
Luis   = 3
Cuba   = 3
Rusia  = 1
Corea  = 2
Piano  = 1
Saxo   = 2
Flauta = 3
```

# CLP(FD): Problema de la cebra

- Problema de la cebra (Lewis Carroll): Cinco hombres de distintas nacionalidades viven en las cinco primeras casas de una calle. Cada uno tiene una profesión, un animal favorito y una bebida favorita (todas distintas de la de los otros). Sabemos que
  1. El inglés vive en la casa roja.
  2. El español tiene un perro.
  3. El japonés es pintor.
  4. El italiano bebe té.
  5. El Noruego vive en la primera casa de la izquierda.
  6. El propietario de la casa verde bebe café.
  7. La casa verde está a la derecha de la blanca.
  8. El escultor cria caracoles.
  9. El diplomático vive en la casa amarilla.

# CLP(FD): Problema de la cebra

- Problema de la cebra (continuación):

  10. En la casa central beben leche.
  11. La casa del noruego está al lado de la casa azul.
  12. El violinista bebe zumo de fruta.
  13. El zorro está en la casa vecina de la del médico.
  14. El caballo está en la casa vecina de la del diplomático.

  Determinar dónde está la cebra y quién bebe agua.

- La solución es

| | Nacionalidad | Color | Profesión | Animal | Bebida |
|---|---|---|---|---|---|
| 1 | Noruego | Amarilla | Diplomático | Zorro | **Agua** |
| 2 | Italiano | Azul | Doctor | Caballo | Te |
| 3 | Inglés | Roja | Escultor | Caracol | Leche |
| 4 | Espagnol | Blanca | Violinista | Perro | Zumo |
| 5 | Japones | Verde | Pintor | **Cebra** | Cafe |

## CLP(FD): Problema de la cebra

Modelización:

```
:- use_module(library('clp/bounds')).

cebra(Cebra, Agua, S) :-
    % Variables:
    S = [Nacionalidad,Color,Profesión,Animal,Bebida],
    Nacionalidad = [Inglés,Español,Japones,Italiano,Noruego
    Color = [Roja,Verde,Blanca,Amarilla,Azul],
    Profesión = [Pintor,Escultor,Diplomático,Violinista,Doc
    Animal = [Perro,Caracol,Zorro,Caballo,Cebra],
    Bebida = [Te,Cafe,Leche,Zumo,Agua],
    flatten(S, L),

    % Dominios:
    L in 1..5,
```

## CLP(FD): Problema de la cebra

```
% Restricciones:
 all_different(Nacionalidad),
 all_different(Color),
 all_different(Profesión),
 all_different(Animal),
 all_different(Bebida),
 Inglés = Roja,                    % 1
 Español = Perro,                  % 2
 Japones = Pintor,                 % 3
 Italiano = Te,                    % 4
 Noruego = 1,                      % 5
 Verde = Cafe,                     % 6
 Verde #= Blanca + 1,              % 7
 Escultor = Caracol,               % 8
 Diplomático = Amarilla,           % 9
```

## CLP(FD): Problema de la cebra

```
    Leche = 3,                          % 10
    vecino(Noruego, Azul),              % 11
    Violinista = Zumo,                  % 12
    vecino(Zorro, Doctor),              % 13
    vecino(Caballo, Diplomático),       % 14

    % Instanciación:
    label(L).

vecino(X,Y) :-
    (X #= Y+1) #\/ (X #= Y-1).
```

# CLP(FD): Problema de la cebra

Solución:

```
?- cebra(Cebra, Agua, S).
Cebra = 5
Agua = 1
S = [[3, 4, 5, 2, 1],
     [3, 5, 4, 1, 2],
     [5, 3, 1, 4, 2],
     [4, 3, 1, 2, 5],
     [2, 5, 3, 4, 1]] ;
No
```

# Bibliografía


- I. Bratko *Prolog Programming for Artificial Intelligence (Third ed.)* (Prentice–Hall, 2001)
  - Cap 14: “Constraint logic programming”
- A.M. Cheadle, W. Harvey, A.J. Sadler, J. Schimpf, K. Shen y M.G. Wallace *ECLiPSe: An Introduction* (Imperial College London, Technical Report IC–Parc–03–1, 2003)
- K. Marriott y P.J. Stuckey *Programming with Constraints. An Introduction.* (The MIT Press, 1998).
- J. Wielemaker *SWI–Prolog 5.6 (Reference Manual)*
  - A.13: “clp/bounds: Integer Bounds Constraint Solver”
  - A.14: “clpqr: Constraint Logic Programming over Rationals and Reals”.