# Tema 10: Programación lógica con restricciones

**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

# Sistemas de programación lógica con restricciones

- **Descripción del sistema que usaremos en el tema**
  - **ECLiPSe (ECLiPSe Common Logic Programming System)**
  - **es un sistema basado en Prolog**
  - **su objetivo es de servir como plataforma para integrar varias extensiones de la programación lógica, en particular la programación lógica con restricciones (CLP)**
  - **Llamada al sistema**

```
> eclipse
ECLiPSe Constraint Logic Programming System [kernel]
Copyright Imperial College London and ICL
Certain libraries copyright Parc Technologies Ltd
GMP library copyright Free Software Foundation
Version 5.7 #28, Mon Dec 22 00:13 2003
```

# CLP sobre números reales: CLP(R)

- **Uso de la biblioteca CLP(R)**

```
[eclipse 1]: lib(clpr).
Yes (0.32s cpu)
```

- **Diferencia entre objetivo y restricción:**

```
[eclipse 2]: 1+X=5.
No (0.00s cpu)

[eclipse 3]: {1+X=5}.
X = 4.0
Yes (0.00s cpu)
```

- **Restricciones aritméticas:**
  - <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 sobre números reales: CLP(R)

- **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**

```
[eclipse 4]: convierte_1(C,95).
C = 35
Yes (0.00s cpu)

[eclipse 5]: convierte_1(35,F).
instantiation fault in -(F, 32, _192) in module eclipse
Abort
```

# CLP sobre números reales: CLP(R)

- **Programa CLP(R)**

```
:- lib(clpr).

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

- **Sesión con el programa CLP(R)**

```
[eclipse 6]: convierte_2(C,95).
C = 35.0
Yes (0.00s cpu)

[eclipse 7]: convierte_2(35,F).
F = 95.0
Yes (0.00s cpu)

[eclipse 8]: convierte_2(C,F).
% Linear constraints:
{F = 32.0 + 1.7999999999999998 * C}
Yes (0.00s cpu)
```

# CLP sobre números reales: CLP(R)

- **Ejemplo de ecuaciones e inecuaciones:**

```
[eclipse 9]: {3*X-2*Y=6, 2*Y=X}.
Y = 1.5
X = 3.0
Yes (0.00s cpu)

[eclipse 10]: {Z=<X-2, Z=<6-X, Z+1=2}.
X = X
Z = 1.0
% Linear constraints:
{X =< 5.0, X >= 3.0}
Yes (0.00s cpu)

[eclipse 11]: {X>0, X+2<0}.
No (0.00s cpu)
```

# CLP sobre números reales: CLP(R)

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

```
[eclipse 12]: {X=<5}, maximize(X).
X = 5.0
Yes (0.00s cpu)

[eclipse 13]: {X=<5, 2=<X}, minimize(2*X+3).
X = 2.0
Yes (0.00s cpu)

[eclipse 14]: {3=<X, X+1=<Y+2, Y=<9, Z=X+Y}, minimize(Z).
X = 3.0
Y = 2.0
Z = 5.0
Yes (0.00s cpu)

[eclipse 15]: {X+Y=<4}, maximize(X+Y).
% Linear constraints: {Y = 4.0 - X}
Yes (0.00s cpu)

[eclipse 16]: {X=<5}, minimize(X).
No (0.00s cpu)
```

# CLP sobre números reales: CLP(R)

- **Optimización con `sup/2` e `inf/2`:**

```
[eclipse 17]: {2=<X, X=<5}, inf(X,I), sup(X,S).
I = 2.0
X = X
S = 5.0
% Linear constraints:
{X =< 5.0, X >= 2.0}
Yes (0.00s cpu)

[eclipse 18]: {3=<X, X+1=<Y+2, Y=<9, Z=X+Y}, inf(Z,I), sup(Z,S), minimize(Z).
X = 3.0
Y = 2.0
I = 5.0
S = 19.0
Z = 5.0
Yes (0.00s cpu)
```

# CLP sobre números reales: 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:**

```
[eclipse 19]: {Ta>=0, Ta+2=<Tb, Ta+2=<Tc, Tb+3=<Td, Tc+5=<Tf, Td+4=<Tf},
              minimize(Tf).
Ta = 0.0
Tb = 2.0
Tc = Tc
Td = 5.0
Tf = 9.0
% Linear constraints:
{Tc =< 4.0, Tc >= 2.0}
Yes (0.01s cpu)
```

# CLP sobre números reales: CLP(R)

- **Sucesión de Fibonacci:**
  - **Especificación:**
    `fib(+N,-F)` **se verifica si** `F` **es el** `N`**-ésimo término de la sucesión de Fibonacci; es decir, 0, 1, 1, 2, 3, 5, 8, 13, 21, ...**
  - **Programa Prolog**

```
fib_1(N,F) :-
   ( N=0, F=1
   ; N=1, F=1
   ; N>1,
     N1 is N-1, fib_1(N1,F1),
     N2 is N-2, fib_1(N2,F2),
     F is F1+F2 ).
```

  - **Sesión con el programa Prolog**

```
[eclipse 20]: fib_1(6,F).
F = 13
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 21]: fib_1(N,13).
instantiation fault in N > 1 in module eclipse
Abort
```

# CLP sobre números reales: CLP(R)

- **Programa CLP(R)**

```
:- lib(clpr).

fib_2(N,F) :-
   ( {N=0, F=1}
   ; {N=1, F=1}
   ; {N>1, F=F1+F2, N1=N-1, N2=N-2},
     fib_2(N1,F1),
     fib_2(N2,F2) ).
```

- **Sesión con el programa CLP(R)**

```
[eclipse 22]: fib_2(6,F).
F = 13.0
Yes (0.01s cpu, solution 1, maybe more) ?
[eclipse 23]: fib_2(N,13).
N = 6.0
Yes (0.02s cpu, solution 1, maybe more) ?
[eclipse 24]: fib_2(N,4).
interruption: type a, b, c, e, or h for help : ? a
abort
Aborting execution ...
Abort
```

# CLP sobre números reales: CLP(R)

- **Modificación de `fib_2` para determinar los números que no son términos de la sucesión**

```
fib_3(N,F) :-
   ( {N=0, F=1}
   ; {N=1, F=1}
   ; {N>1, F=F1+F2, N1=N-1, N2=N-2, F1>=N1, F2>=N2},
     fib_3(N1,F1),
     fib_3(N2,F2) ).
```

- **Sesión**

```
[eclipse 25]: fib_3(6,F).
F = 13.0
Yes (0.02s cpu, solution 1, maybe more) ?
[eclipse 25]: fib_3(N,13).
N = 6.0
Yes (0.04s cpu, solution 1, maybe more) ?
[eclipse 27]: fib_3(N,4).
No (0.01s cpu)
```

# CLP sobre números racionales: CLP(Q)

- **CLP sobre números racionales: CLP(Q)**

```
[eclipse 1]: lib(clpq).
Yes (0.66s cpu)

[eclipse 2]: {X=2*Y, Y=1-X}.
Y = 1 / 3
X = 2 / 3
Yes (0.00s cpu)
```

# Planificación de tareas con CLP(Q)

- **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).
    ```

# Planificación de tareas con CLP(Q)

- **Planificador**
  - **Biblioteca CLP(Q)**

```
:- lib(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,**

```
[eclipse 3]: planificación(P,TP).
P = [t1/0/5, t2/5/7, t3/12/10, t4/I/2, t5/I/9]
TP = 22
% Linear constraints:
{ I =< 13, I >= 5, I - I >= 2 }
Yes (0.01s cpu)
```

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

# Planificación de tareas con CLP(Q)

- `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).
```

- `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)

- **Restricciones de dominio y aritméticas**
  - **Ejemplos:**

```
[eclipse 1]: lib(fd).

[eclipse 2]: X :: 1..5, Y :: 0..4, X #< Y, Z #= X+Y+1.
Z = Z{[4..8]}    X = X{[1..3]}    Y = Y{[2..4]}
Delayed goals:  Y{[2..4]} - X{[1..3]}#>=1
                -1 - X{[1..3]} - Y{[2..4]} + Z{[4..8]}#=0

[eclipse 3]: [X,Y] :: 1..3, Z #=X+Y.
Z = Z{[2..6]}    X = X{[1..3]}    Y = Y{[1..3]}
Delayed goals:  0 - X{[1..3]} - Y{[1..3]} + Z{[2..6]}#=0
```

  - **Tipos de restricciones**

    * **de dominio de variables:** `<variable> :: <Mínimo>..<Máximo>`

    * **de dominio de lista:** `<lista> :: <Mínimo>..<Máximo>`

    * **aritmética:** `<expresión 1> <relación> <expresión 2>` **con** `<relación>` **en** `#=, #\=, #<, #>, #=<, #>=.`

# Restricciones sobre dominios finitos: CLP(FD)

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

```
[eclipse 4]: X :: 1..3, indomain(X).
X = 1   Yes (0.00s cpu, solution 1, maybe more) ? ;
X = 2   Yes (0.01s cpu, solution 2, maybe more) ? ;
X = 3   Yes (0.01s cpu, solution 3)
```

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

```
[eclipse 5]: L=[X,Y], L :: 1..2, labeling(L).
X = 1   Y = 1   L = [1, 1]   Yes (0.00s cpu, solution 1, maybe more) ? ;
X = 1   Y = 2   L = [1, 2]   Yes (0.00s cpu, solution 2, maybe more) ? ;
X = 2   Y = 1   L = [2, 1]   Yes (0.00s cpu, solution 3, maybe more) ? ;
X = 2   Y = 2   L = [2, 2]   Yes (0.00s cpu, solution 4)
```

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

```
[eclipse 5]: L=[X,Y], L :: 1..2, alldifferent(L), labeling(L).
X = 1   Y = 2   L = [1, 2]   Yes (0.00s cpu, solution 1, maybe more) ? ;
X = 2   Y = 1   L = [2, 1]   Yes (0.00s cpu, solution 2)
```

# Restricciones sobre dominios finitos: CLP(FD)

- **Problema de criptoaritmética**
  - **Especificación:** `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`**.**
  - **Programa**

```
:- lib(fd).

solución([S,E,N,D],[M,O,R,E],[M,O,N,E,Y]) :-
   Vars = [S,E,N,D,M,O,R,Y], Vars :: 0..9,
   alldifferent(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,
   labeling(Vars).
```

  - **Solución**

```
[eclipse 6]: solución(L1,L2,L3).
L1 = [9,5,6,7]   L2 = [1,0,8,5]   L3 = [1,0,6,5,2]
Yes (0.00s cpu, solution 1, maybe more) ? ;
No (0.00s cpu)
```

# Restricciones sobre dominios finitos: CLP(FD)

- **Problema de las N reinas**
  - `solución(N,L)` **se verifica si** `L` **es una solución del problema de las** `N` **reinas. Por ejemplo,**

```
[eclipse 7]: solución(4,L).
L = [2, 4, 1, 3]   Yes (0.00s cpu, solution 1, maybe more) ? ;
L = [3, 1, 4, 2]   Yes (0.00s cpu, solution 2, maybe more) ? ;
No (0.00s cpu)
[eclipse 8]: findall(L,solución(8,L),LS), length(LS,N).
N = 92
Yes (0.02s cpu)
```

```
:- lib(fd).

solución(N,L) :-
   length(L,N),              % Hay N reinas
   L :: 1..N,                % Las ordenadas están en el intervalo 1..N
   alldifferent(L),          % Las ordenadas son distintas (distintas filas)
   segura(L),                % No hay en más de una en las diagonales
   labeling(L)    .          % Buscar los valores de L
```

# Restricciones sobre dominios finitos: CLP(FD)

- `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).
```

- `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).
```

# Restricciones sobre dominios finitos: CLP(FD)

- **Optimización:**
  - `minimize(P,V)` **busca una solución del problema** `P` **que minimiza el valor de** `V`**. Por ejemplo,**

```
[eclipse 9]: X :: 1..6, V #= X*(X-6), minimize(indomain(X),V).
Found a solution with cost -5
Found a solution with cost -8
Found a solution with cost -9

X = 3
V = -9
Yes (0.00s cpu)

[eclipse 10]: X :: 1..6, V #= X*(6-X), minimize(indomain(X),V).
Found a solution with cost 5
Found a solution with cost 0

X = 6
V = 0
Yes (0.00s cpu)
```

# Restricciones sobre dominios finitos: CLP(FD)

- **Problemas de planificación óptima de tareas con CLP(FD):**
  - **Planificación**

```
[eclipse 11]: Tiempos_iniciales = [Ta,Tb,Tc,Td,Tf],
              Tiempos_iniciales :: 0..10,
              Ta+2 #=< Tb,
              Ta+2 #=< Tc,
              Tb+3 #=< Td,
              Tc+5 #=< Tf,
              Td+4 #=< Tf.

Tiempos_iniciales = [Ta{[0, 1]}, Tb{[2, 3]}, Tc{[2..5]}, Td{[5, 6]}, Tf{[9, 10]}]

Delayed goals:
 Tf{[9, 10]} - Tc{[2..5]}#>=5
 Tc{[2..5]} - Ta{[0, 1]}#>=2
 Tf{[9, 10]} - Td{[5, 6]}#>=4
 Td{[5, 6]} - Tb{[2, 3]}#>=3
 Tb{[2, 3]} - Ta{[0, 1]}#>=2
Yes (0.00s cpu)
```

# Restricciones sobre dominios finitos: CLP(FD)

- **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 | | |

# Restricciones sobre dominios finitos: CLP(FD)

- **Optimización**

```
[eclipse 11]: Tiempos_iniciales = [Ta,Tb,Tc,Td,Tf],
              Tiempos_iniciales :: 0..20,
              Ta+2 #=< Tb,
              Ta+2 #=< Tc,
              Tb+3 #=< Td,
              Tc+5 #=< Tf,
              Td+4 #=< Tf,
              minimize(labeling(Tiempos_iniciales),Tf).
Found a solution with cost 9
Ta = 0
Tb = 2
Tc = 2
Td = 5
Tiempos_iniciales = [0, 2, 2, 5, 9]
Tf = 9
Yes (0.00s cpu)
```

# Restricciones sobre dominios finitos: CLP(FD)

- **Reducción del espacio de búsqueda**
  - **Genera y prueba**

```
p(1).  p(3).  p(7).  p(16).  p(15).  p(14).
solución_1(X,Y,Z) :-
   p(X), p(Y), p(Z),
   q_1(X,Y,Z).
q_1(X,Y,Z) :-  Y =:= X+1, Z =:= Y+1.
```

  - **Estadísticas**

```
[eclipse 25]: lib(port_profiler).
[eclipse 26]: port_profile(solución_1(X,Y,Z),[]).
PREDICATE       call
p           /1    42
+           /3   218
=:=         /2   218
q_1         /3   208
solución_1  /3     1

X = 14   Y = 15   Z = 16
Yes (0.00s cpu)
```

# Restricciones sobre dominios finitos: CLP(FD)

- **Restringe y genera**

```
:- lib(fd).

solución_2(X,Y,Z) :-
   q_2(X,Y,Z),
   p(X), p(Y), p(Z).

q_2(X,Y,Z) :- Y #= X+1, Z #= Y+1.
```

- **Estadísticas**

```
[eclipse 27]: port_profile(solución_2(X,Y,Z),[]).
PREDICATE       call
p           /1     9
q_2         /3     1
solución_2  /3     1

X = 14   Y = 15   Z = 16
Yes (0.00s cpu)
```

# 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).