DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

ECLiPSe
CLP - Constraint Logic Programming

ECLiPSe

Herramienta open-source de CLP [Constraint Logic Programming] útil para modelar problemas de satisfacción de restricciones:

http://www.eclipse-clp.org

Historia

1991 – European Computer-Industry Research Centre (ECRC), Munich
1999 – Parc Technologies (spin-off del Imperial College, Londres)
2001 – Biblioteca ic (propagación de restricciones)
2004 – Adquisición de Parc Technologies por parte de Cisco Systems

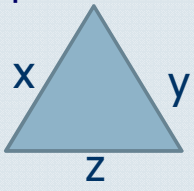
ECLiPSe

- **PROLOG**
  - Elementos del lenguaje
  - Búsqueda en PROLOG: Backtracking
  - Satisfacción de restricciones en PROLOG
- **ECLiPSe**
  - Satisfacción de restricciones en ECLiPSe
  - Elementos del lenguaje: arrays e iteradores.
  - La biblioteca suspend
  - Las bibliotecas sd & ic
  - Estrategias de búsqueda

PROLOG Satisfacción de restricciones

Los problemas de satisfacción de restricciones pueden representarse mediante programas lógicos:

Problema	PSR

• Desigualdad triangular: La suma de las longitudes de cualesquiera 2 lados no es menor que la del tercero. 	• Variables: X,Y,Z
	• Dominios: [0,inf)
	• Restricciones
	  • X+Y>=Z
	  • X+Z>=Y
	  • Y+Z>= X

PROLOG Satisfacción de restricciones

Los problemas de satisfacción de restricciones pueden representarse mediante programas lógicos:

PSR

- Variables: X,Y,Z
- Dominios: [0,inf)
- Restricciones
  - X+Y>=Z
  - X+Z>=Y
  - Y+Z>= X

Programa Lógico

```
triangular(X,Y,Z)
:-
  X>0 AND Y>0 AND Z>0
  AND
  X+Y>=Z AND X+Z>=Y AND Y+Z>=X
```


PROLOG Satisfacción de restricciones

Un programa lógico puede interpretarse como una base de conocimiento (hechos y reglas) sobre la que se realizan consultas

El programa lógico termina con

- Una respuesta "Sí"/"No" en función de si se puede deducir la consulta de la base de conocimiento.
- Una sustitución de variables que la hace cierta.

Programa Lógico

```
triangular(X,Y,Z):-
X>0 AND Y>0 AND Z>0 AND X+Y>=Z
AND X+Z>=Y AND Y+Z>=X
```

Consulta

- `triangular(3,4,5)?`
  Respuesta: Sí
- `triangular(8,1,2)?`
  Respuesta: No
- `triangular(3,4,Z)?`
  Respuesta: Z = [1..7]

PROLOG

Ejemplo

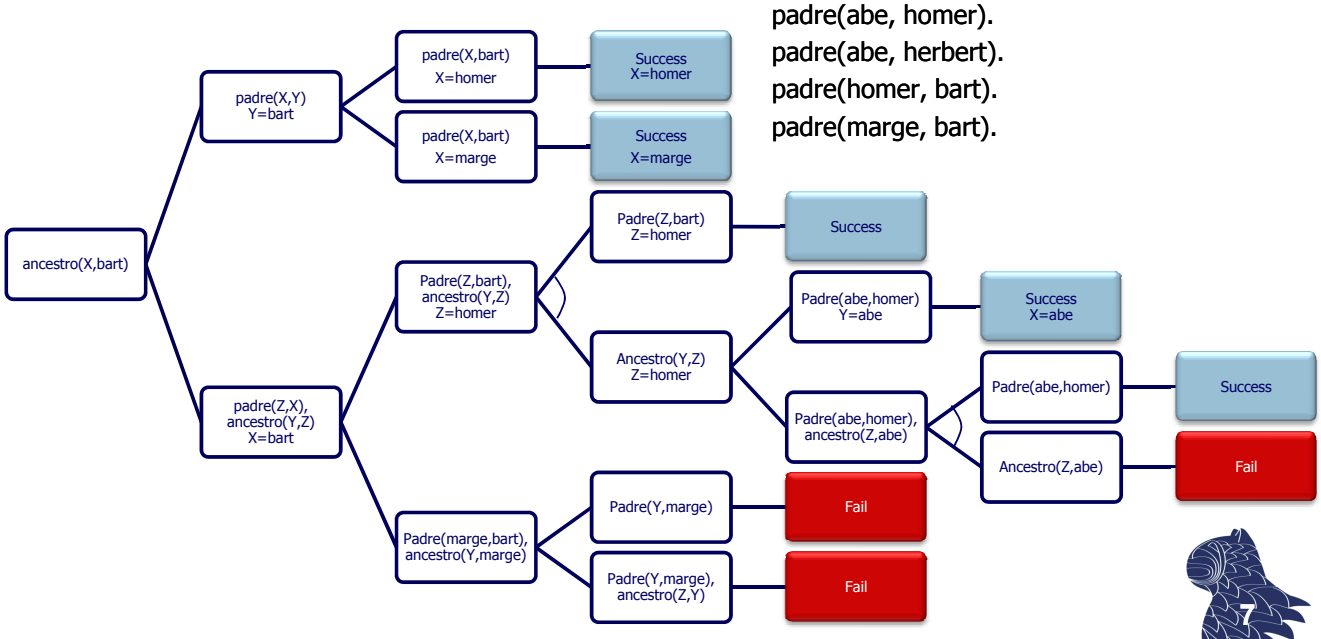
REGLAS
ancestro(X,Y) :- padre(X,Y).
ancestro(X,Y) :- padre(Z,Y), ancestro(X,Z).

HECHOS
padre(abe, homer).
padre(abe, herbert).
padre(homer, bart).
padre(marge, bart).

PROLOG

Consulta ancestro(X,bart).

ancestro(X,Y) :- padre(X,Y).
ancestro(X,Y) :- padre(Z,Y), ancestro(X,Z).
padre(abe, homer).
padre(abe, herbert).
padre(homer, bart).
padre(marge, bart).



PROLOG - Aritmética

Operadores predefinidos para aritmética básica:
+, -,*, div, mod, ^, - (unario), abs

- Si no se especifica lo contrario, los operadores son como cualquier otra relación (predicado)

  X = 1+2.
  Respuesta: X=1+2 % X unificado al término +(1,2)

- El operador "is" fuerza la evaluación de expresiones aritméticas: **A is B** evalúa B a un número y unifica el resultado con A.

  X is 1+2.
  Respuesta: X = 3

- Los operadores de comparación fuerzan también la evaluación

  145 * 34 > 100.
  Respuesta: Yes.

PROLOG - Comparación

Operadores

X > Y	X es mayor que Y
X < Y	X es menor que Y
X >= Y	X es mayor o igual que Y
X =< Y	X es menor o igual que Y
X =:= Y	Los valores de X e Y son iguales
X =\= Y	Los valores de X e Y no son iguales

1 + 2 =:= 2 + 1.
  > Yes
1 + 2 = 2 + 1.
  > No
1 + A = B + 2.
  > A = 2
  > B = 1
1+A =:= B+2.
  > Error (variables no instanciadas)

= vs. =:=

- **X=Y**
  causa la unificación de X e Y (y puede que la instanciación de variables)
- **X =:= Y**
  causa una evaluación aritmética de X e Y (pero no puede dar lugar a instanciación de variables)

X is Y + 1
  > Error (variable no instanciada)
Y=0, X is Y + 1.
  > X = 1
X = 0, X is X + 1
  > No.
  No se puede unificar X con X + 1.
X = 0, X1 is X + 1.
  > Yes. X = 0, X1 = 1.

PROLOG - Listas

[a, [1,2,3],tom, 1995, fecha(1,mayo,1995)]

- Lista vacía: []
- Lista = [Cabeza | Cola]
  - Cabeza: primer elemento de la lista.
  - Cola: Una lista formada por el resto de la lista.

[X|Y] = [a,b,c]
> X=a, Y=[b,c]

Representan a la misma lista:

[a,b,c] = [a| [b,c]] = [a, b |[c]] = [a,b,c | []]

PROLOG - Listas

Ejemplos

- Miembro de una lista

```
member(X, [X|Tail]).
member(X, [Head | Tail]) :- member(X,Tail).
```

- Concatenación de listas

```
append([],L,L).
append([X|L1], L2, [X|Result]) :- append(L1,L2,Result).
```

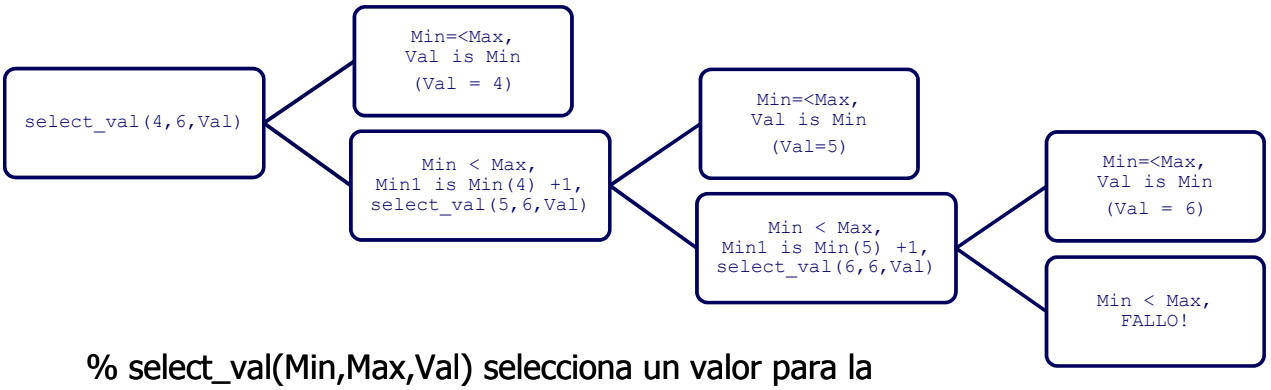
PROLOG - Listas

Ejemplos de uso

append([a,b,c], [1,2,3], L).
> L = [a,b,c,1,2,3]
append(L1, L2, [a,b,c]).
> L1 = [], L2 = [a,b,c];
> L1 = [a], L2 = [b,c];
> L1 = [a,b], L2 = [c];
> L1 = [a,b,c], L2 = [];
> no
append(Before, [4|After], [1,2,3,4,5,6,7]).
> Before = [1,2,3], After = [5,6,7]
append(_, [Pred, 4, Succ |_], [1,2,3,4,5,6,7]).
> Pred = 3, Succ = 5

PROLOG Ejemplos

Selección de valores dentro de un rango definido:




```
% select_val(Min,Max,Val) selecciona un valor para la
% variable Val en el rango [Min,Max], ambos incluidos

select_val(Min,Max,Val) :- Min =< Max, Val is Min.
select_val(Min,Max,Val) :-
        Min < Max, Min1 is Min + 1,
        select_val(Min1,Max,Val).
```

PROLOG Satisfacción de restricciones

Esquema para la resolución de problemas de satisfacción de restricciones en PROLOG:

```
solución(Lista) :-
        declaraciónDominios(Lista),
        búsquedaAsignación(Lista),
        comprobaciónRestricciones(Lista).
```

1. Asignar valores, dentro de sus dominios, a cada una de las variables de nuestro problema.
2. Comprobar si las asignaciones verifican todas las restricciones de nuestro problema.
3. Si no es así y quedan más valores, volver a 1.

PROLOG Satisfacción de restricciones

Ejemplo

- Variables: X,Y
- Dominios: [1..4]
- Restricciones: X<Y, Y>3, X<>2

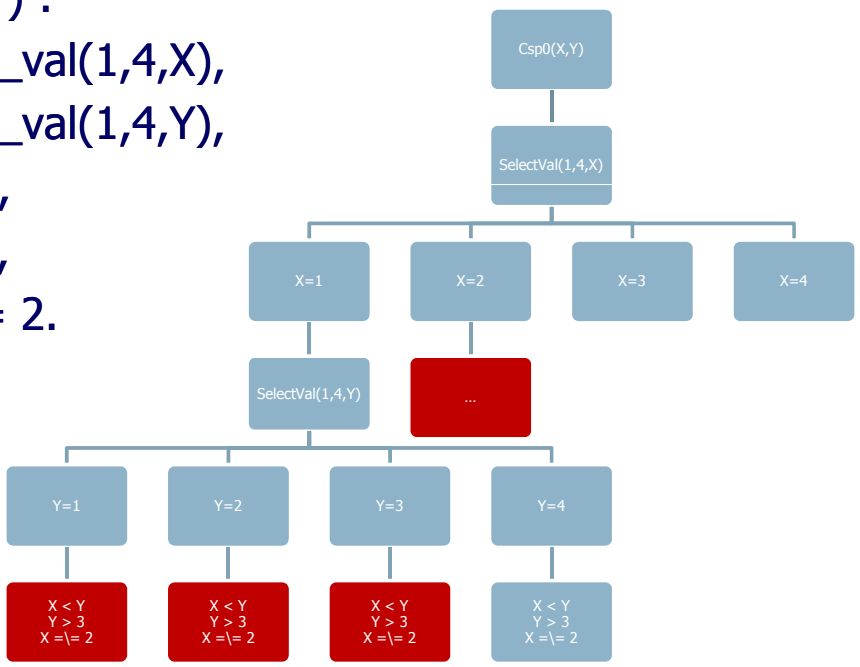
```
% Con select_val
csp0(X,Y) :-
        select_val(1,4,X),
        select_val(1,4,Y),
        X < Y,
        Y > 3,
        X =\= 2.
```

```
% Con listas PROLOG
csp0(X,Y) :-
        member(X, [1,2,3,4]),
        member(X, [1,2,3,4]),
        X < Y,
        Y > 3,
        X =\= 2.
```

PROLOG Satisfacción de restricciones

Ejemplo

```
csp0(X,Y) :-
    select_val(1,4,X),
    select_val(1,4,Y),
    X < Y,
    Y > 3,
    X =\= 2.
```



PROLOG Satisfacción de restricciones

solución(Lista) :-
declaraciónDominios(Lista),
búsquedaAsignación(Lista),
comprobaciónRestricciones(Lista).

Inconvenientes

- **Ineficiencia**: Las variables en las restricciones tienen que estar instanciadas para poder evaluarse.
- **Dificultad**: Definición de heurísticas para la selección de variables y valores, mecanismos de propagación de restricciones…

ECLiPSe Satisfacción de restricciones

Esquema para la resolución de problemas de satisfacción de restricciones en ECLiPSe:

```
:-lib(<biblioteca>).

solución(Lista) :-
        declaraciónDominios(Lista),
        especificaciónRestricciones(Lista),
        búsqueda(Lista).
```


ECLiPSe – suspend

Biblioteca suspend

PROLOG:
```
Y=3, 2 < Y + 1.
Yes
2 < Y + 1, Y=3.
Abort. Instantiation fault.
```

ECLiPSe:
```
suspend:(2<Y+1), Y=3.
Yes
```

- La comprobación de los objetivos anotados son se retrasa hasta que sus variables están instanciadas.
- Cambio en el control de la ejecución del programa: Ahora se selecciona el objetivo más a la izquierda que no sea una expresión aritmética no instanciada.

ECLiPSe – suspend

Restricciones definidas en la biblioteca suspend

- Declaración de rangos de variables

```
Var :: <rango>
Var $:: <rango>     (reales)
Var #:: <rango>     (enteros)      X::1..9
Lista :: <rango>                   [X,Y,Z]::1..9
```

- Operadores booleanos:

```
X or Y          % Disyunción booleana
X and Y         % Conjunción booleana
neg(X)          % Negación booleana
X=>Y            % Implicación booleana
```

ECLiPSe – suspend

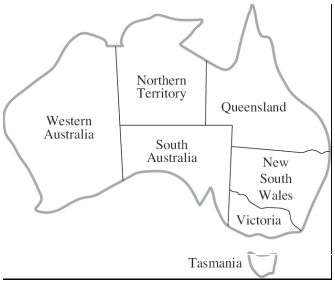
Restricciones definidas en la biblioteca suspend

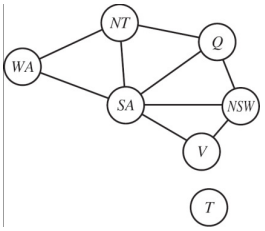
- Operadores de comparación

Prolog		suspend (reales)	suspend (enteros)
X > Y	X es mayor que Y	**X $> Y**	**X #> Y**
X < Y	X es menor que Y	**X $<Y**	**X #<Y**
X >= Y	X es mayor o igual que Y	**X $>= Y**	**X #>= Y**
X =< Y	X es menor o igual que Y	**X $=< Y**	**X #=< Y**
X =:= Y	Los valores de X e Y son iguales	**X $= Y**	**X #= Y**
X =\= Y	Los valores de X e Y no son iguales	**X $\= Y**	**X #\= Y**

ECLiPSe Ejemplo: Coloreado de mapas

```
:- lib(suspend).

coloreado(Vars) :-
  % Declaración de variables y dominios
  Vars = [WA,NT,Q,NSW,V,SA,T],
  Vars :: 1..3,
  % Especificación de restricciones
  SA $\= WA, SA $\= NT, SA $\= NSW, SA $\=V,
  WA $\= NT,
  NT $\= Q,
  Q $\= NSW,
  NSW $\= V,
  % Búsqueda (ciega)
  (foreach(Var, Vars) do
    select_val(1,3,Var)).

select_val(Min,Max,Val) :- Min =< Max, Val is Min.
select_val(Min,Max,Val) :- Min < Max, Min1 is Min +1, select_val(Min1,Max,Val).
```





ECLiPSe – Iteradores

- En Prolog, la única forma de expresar iteración es mediante recursividad.
- ECLiPSe proporciona estructuras iterativas con **do**:
  (<EstructuraIterativa> do <Objetivos>)

Ejemplos

- Elementos de una lista:

```
write_list(List) :-
  write("Lista: "), ( foreach(X,List) do write(X) ).
```

- Rango de números:

```
write_numbers(N) :-
  write("Números: "), ( for(I,1,N) do write(I) ).
```

ECLiPSe – Iteradores y arrays

ECLiPSe permite el uso de arrays con el functor [] :

```
A1 = [](1,2,3)
A2 = [](_, apple, orange, _)
Matriz = []([](1,2,3), [](4,5,6))
```

?- M = []([](2, 3, 5), [](1, 4, 7)), X is M[1,2] + M[2,3].
X = 10

?- dim(M, [2, 3]).
M = []([](_, _, _), [](_, _, _))

?- dim(M4, [3, 4, 2, 1]).
M4 = ...

?- M = []([](1, 2, 3), [](4, 5, 6)), dim(M, Dim).
Dim = [2, 3]

ECLiPSe – Iteradores y arrays

Acceso a submatrices

```
?- Matrix = []([](1,2,3),[](4,5,6),[](7,8,9)),
   Row2 is Matrix[2,1..3],
   SubRow2 is Matrix[2,2..3],
   Column1 is Matrix[1..3,1].
Row2 = [4, 5, 6]
SubRow2 = [5, 6]
Column1 = [1, 4, 7]
```

Uso de iteradores con arrays:

```
write_array(A) :- dim(A,[5]),  ( foreachelem(El, A) do write(El)).

write_array(A) :- dim(A,[5]),  ( for(I,1,5), param(A) do
                                    write(A[I])
                               ).
```

ECLiPSe Ejemplo: Coloreado de mapas

```
colorea(Regiones) :-
  regiones(Count),
  dim(Regiones,[Count]),
  (multifor ([I,J],1,Count),
    param(Regiones)
   do % if-then-else
     (vecino(I,J) ->
        Regiones[I] $\= Regiones[J]
        ;
        true
     ) ),
 colores(NumColors),
 ( foreacharg(R,Regiones),
   param(NumColors) do
     select_val(1,NumColors,R)).
```

```
% Configuración
% Mapa particular

regiones(7).
colores(3).
vecino(1,2).
vecino(1,3).
vecino(2,3).
vecino(2,4).
vecino(4,3).
vecino(4,5).
vecino(5,3).
vecino(5,6).
vecino(6,3).
```


ECLiPSe Propagación de restricciones

- La biblioteca **suspend** sólo tiene en cuenta las restricciones de forma pasiva: difiere su comprobación hasta el momento en que se instancian las variables.
- ECLiPSe también incluye bibliotecas en las que se incorpora un mecanismo de propagación de restricciones (**sd** para restricciones sobre valores simbólicos e **ic** para restricciones sobre valores intervalares, como enteros o reales):
  - Cada vez que una variable X se instancia a algún valor, se revisan automáticamente los dominios de las variables Y que estén relacionadas con X
  - ECLiPSe aplica nodo-consistencia y arco-consistencia, AC-3, en cada etapa de asignación de variables.

ECLiPSe – sd

Biblioteca sd

[symbolic domains]

Restricciones sobre valores simbólicos

(internamente representados como enteros)

- Declaración de variables **&::**
- Restricciones **&=** **&\=**

- get_domain_as_list(X,Dom) devuelve en Dom la lista de valores del dominio asociado a X.
- indomain(X) instancia X a un valor de su dominio.

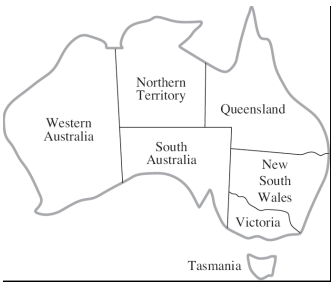
ECLiPSe – sd

Ejemplo

```
:-lib(sd).
csp(List):-
  List=[X,Y,Z,U],
  X&::[a,b,c],
  Y&::[a,b,d],
  Z&::[a,b],
  U&::[b],
  X&=Y,
  Y&\=Z,
  Z&\=U,
  (foreach(V,List) do
    get_domain_as_list(V,D)
    writeln(D)).
```

ECLiPSe Ejemplo: Coloreado de mapas

```
:- lib(sd).

coloreado(Vars) :-
  % Variables y dominios
  Vars = [WA,NT,Q,NSW,V,SA,T],
  Nombres=["WA","NT","Q","NSW",
     "V","SA","T"],
  Vars &:: [rojo, verde, azul],
  % Especificación de restricciones
  SA &\= WA,
  SA &\= NT,
  SA &\= NSW,
  SA &\= V,
  WA &\= NT,
  NT &\= Q,
  Q &\= NSW,
  NSW &\= V,
```




```
  % Propagación de restricciones
  writealldomains(Vars,Nombres),
  indomain(SA),
  writealldomains(Vars,Nombres),
  indomain(NSW),
  writealldomains(Vars,Nombres).

writealldomains(List,Nombres):-
  writeln("====="),
  (foreach(L,List),foreach(N,Nombres) do
    write(N),write("  "),
    writedomain(L)).

writedomain(X):-
  get_domain_as_list(X,Dom),
  writeln(Dom).
```


ECLiPSe Ejemplo: Coloreado de mapas

```
:- lib(sd).
coloreado(Vars) :-
  % Variables y dominios
  Vars = [WA,NT,Q,NSW,V,SA,T],
  Nombres=["WA","NT","Q","NSW",
     "V","SA","T"],
  Vars &:: [rojo, verde, azul],
  % Especificación de restricciones
  SA &\= WA,
  SA &\= NT,
  SA &\= NSW,
  SA &\= V,
  WA &\= NT,
  NT &\= Q,
  Q &\= NSW,
  NSW &\= V,
 % Búsqueda de soluciones
 (foreach(V,Vars) do indomain(V)).
```

NOTA:
El dominio de las variables cambia (se va reduciendo) cada vez que se hace una llamada a **indomain(X)**.

ECLiPSe – ic

Biblioteca ic

[interval constraints]

Dominios discretos finitos o infinitos

(booleanos, enteros, reales y restricciones "reificadas").

- Las restricciones en **ic** utilizan la misma sintaxis que en **suspend** (# para enteros, $ para reales).
- Declaración de dominios:

  X::[1..13]

  X::[3,4,5,7]

  X::[1..3,5,7..9]

  Una variable con dominio sin especificar: [-1.0inf .. 1.0inf].

ECLiPSe – ic

Ejemplo

% Restricciones boolenas

ic:(X or Y), X = 0.

Resultado: X=0, Y=1.

% Restricciones lineales

[X,Y]::[1..4], X/2 - Y/2 #=1.

Resultado: X=4, Y=2.

(excluye 3/2 y 1/2 porque no son enteros).

ECLiPSe – ic

- **labeling(L)** proporciona el mecanismo estándar de búsqueda y propagación en cualquier lenguaje de resolución de problemas de satisfacción de restricciones:

  Para una lista de variables L:
  - Se selecciona una variable X de L.
  - Se instancia a un valor de su dominio (indomain(X)).
  - Se realiza una búsqueda con backtracking.

- **alldifferent(X)** se asegura de que todas las variables de una lista toman valores diferentes:

```
alldifferent(L):-
  (fromto(L,[X|Tail],Tail,[]) do
    (foreach(Y,Tail), param(X) do
      X$\=Y)).
```


ECLiPSe Ejemplo: Criptoaritmética

```
:-lib(ic).

sendmore(Digits) :-
    % Variables y dominios
    Digits = [S,E,N,D,M,O,R,Y],
    Digits :: [0..9],
    % Especificación de restricciones
    alldifferent(Digits),
    S #\= 0,
    M #\= 0,
    1000*S + 100*E + 10*N + D
     + 1000*M + 100*O + 10*R + E
     #= 10000*M + 1000*O
     + 100*N + 10*E + Y,
    % Búsqueda de soluciones
    (foreach(Var,Digits) do
      indomain(Var)).
```

```
  SEND
+MORE
-----
 MONEY
```

Variables: [S,E,N,D,M,O,R,Y]
Dominios: [0..9]
Restricciones:

$$1000*S+100*E+10*N+D+1000*M+100*O+10*R+E = 10000*M+1000*O+100*N+10*E+Y$$

S y M <> 0

28 desigualdades X<>Y, para todo X,Y diferentes

ECLiPSe Ejemplo: Criptoaritmética

```
:-lib(ic).

sendmore(Digits) :-
    % Variables y dominios
    Digits = [S,E,N,D,M,O,R,Y],
    Digits :: [0..9],
    % Especificación de restricciones
    alldifferent(Digits),
    S #\= 0,
    M #\= 0,
    1000*S + 100*E + 10*N + D
     + 1000*M + 100*O + 10*R + E
     #= 10000*M + 1000*O
     + 100*N + 10*E + Y,
    % Búsqueda de soluciones
    labeling(Digits).
```

```
  SEND
+MORE
------
 MONEY
```

Variables: [S,E,N,D,M,O,R,Y]

Dominios: [0..9]

Restricciones:

$$1000*S+100*E+10*N+D+1000*M+100*O+10*R+E = 10000*M+1000*O+100*N+10*E+Y$$

S y M <> 0

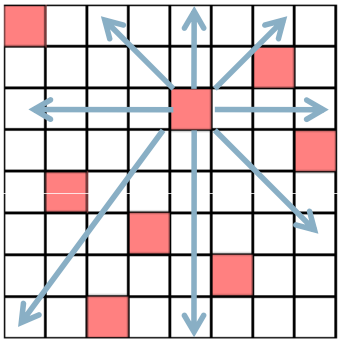
28 desigualdades X<>Y, para todo X,Y diferentes

ECLiPSe El problema de las N reinas

Problema

Situar n reinas

- en un tablero de n x n
- de forma que no se ataquen mutuamente



NOTA: Una reina ataca a todas las celdas en dirección horizontal, vertical y diagonal.

ECLiPSe El problema de las N reinas

Problema

Situar n reinas

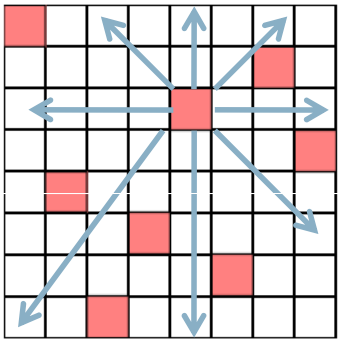
- en un tablero de n x n
- de forma que no se ataquen mutuamente

Variables: X1,..,Xn

Xi=j establece la fila j en que se coloca la reina en la columna i.

Restricciones:

- Xi = [1..N],
- Xi <> Xj, i < j
- Xi <> Xj + j-i
- Xi <> Xj + i-j



NOTA: Una reina ataca a todas las celdas en dirección horizontal, vertical y diagonal.

ECLiPSe El problema de las N reinas

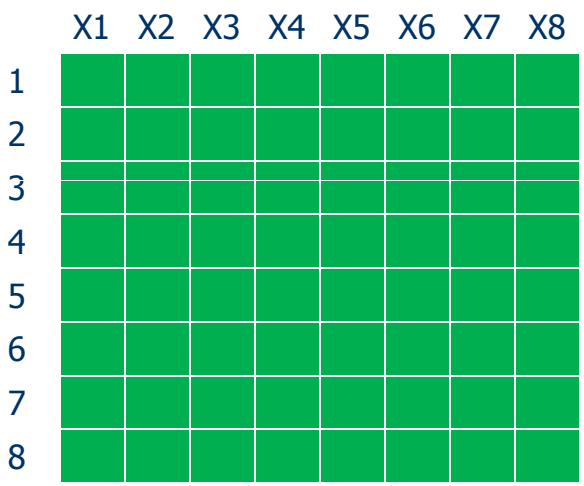
```
:-lib(suspend).

queens(Xs,Number) :-
  dim(Xs,[Number]),
  restricciones(Xs,Number),
  busqueda(Xs,Number).

restricciones(Xs,Number):-
  (for(I,1,Number),
     param(Xs,Number) do
    Xs[I]:: 1..Number,
    (for(J,1,I-1), param(I,Xs) do
      Xs[I] $\= Xs[J],
      Xs[I] - Xs[J] $\= I-J,
      Xs[I] - Xs[J] $\= J-I
      )
   ).
```

```
% Búsqueda sin información
% usando backtracking

busqueda(Xs,N):-
  (foreachelem(Col,Xs),
     param(N)
     do select_val(1,N,Col)).

select_val(Min,Max,Val) :-
   Min =< Max, Val is Min.
select_val(Min,Max,Val) :-
   Min < Max, Min1 is Min+1,
   select_val(Min1,Max,Val).
```

ECLiPSe El problema de las N reinas

```
:-lib(ic).

nqueens(N,Xs):-
  dim(Xs,[N]),
  Xs[1..N]::1..N,
  alldifferent(Xs[1..N]),
  (for(I,1,N-1), param(Xs,N) do
    (for(J,I+1,N), param(Xs,I) do
     Xs[I] #\= Xs[J] + I-J,
     Xs[I] #\= Xs[J] + J-I)),
  labeling(Xs[1..N]).
```

% Propagación de restricciones

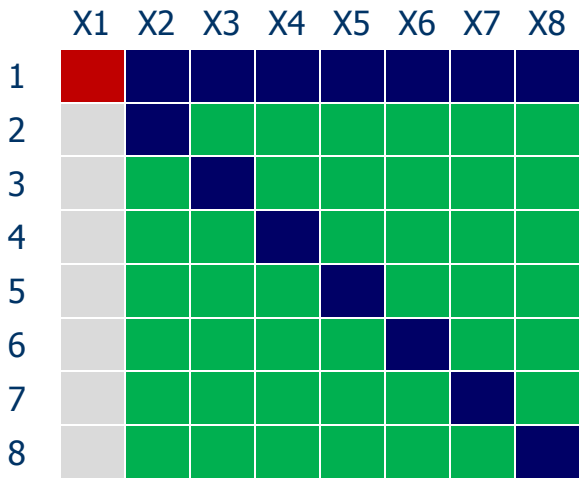


Estadio inicial

ECLiPSe El problema de las N reinas

```
:-lib(ic).

nqueens(N,Xs):-
  dim(Xs,[N]),
  Xs[1..N]::1..N,
  alldifferent(Xs[1..N]),
  (for(I,1,N-1), param(Xs,N) do
    (for(J,I+1,N), param(Xs,I) do
     Xs[I] #\= Xs[J] + I-J,
     Xs[I] #\= Xs[J] + J-I)),
  labeling(Xs[1..N]).
```

% Propagación de restricciones

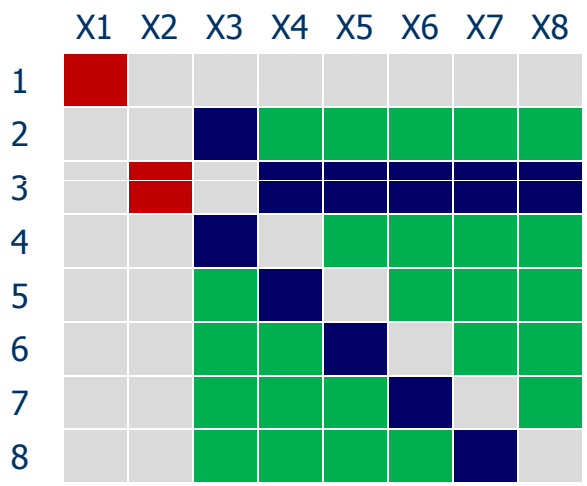


Propagación de restricciones sobre filas, columnas y diagonales:
Al asignar un valor a X1 (rojo), se prohíben los valores incompatibles con las restricciones asociadas a X1 (azul).

ECLiPSe El problema de las N reinas

```
:-lib(ic).

nqueens(N,Xs):-
  dim(Xs,[N]),
  Xs[1..N]::1..N,
  alldifferent(Xs[1..N]),
  (for(I,1,N-1), param(Xs,N) do
    (for(J,I+1,N), param(Xs,I) do
      Xs[I] #\= Xs[J] + I-J,
      Xs[I] #\= Xs[J] + J-I)),
  labeling(Xs[1..N]).
```

% Propagación de restricciones

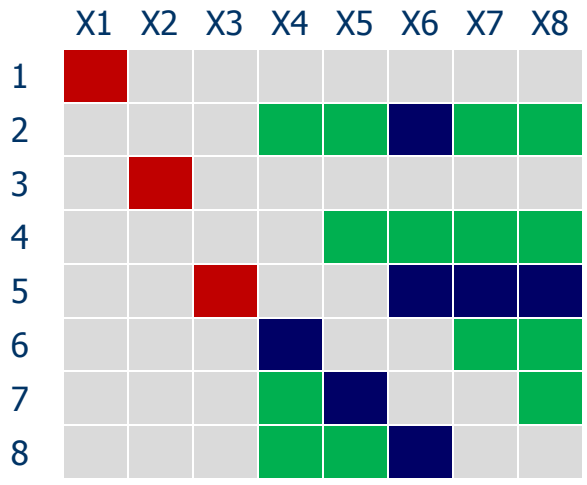


Propagación de restricciones sobre filas, columnas y diagonales: El dominio de X2 pasó a ser [3..8], se prueba con 3 y se propagan las restricciones asociadas a este valor de X2.

ECLiPSe El problema de las N reinas

```
:-lib(ic).

nqueens(N,Xs):-
  dim(Xs,[N]),
  Xs[1..N]::1..N,
  alldifferent(Xs[1..N]),
  (for(I,1,N-1), param(Xs,N) do
    (for(J,I+1,N), param(Xs,I) do
      Xs[I] #\= Xs[J] + I-J,
      Xs[I] #\= Xs[J] + J-I)),
  labeling(Xs[1..N]).
```

% Propagación de restricciones

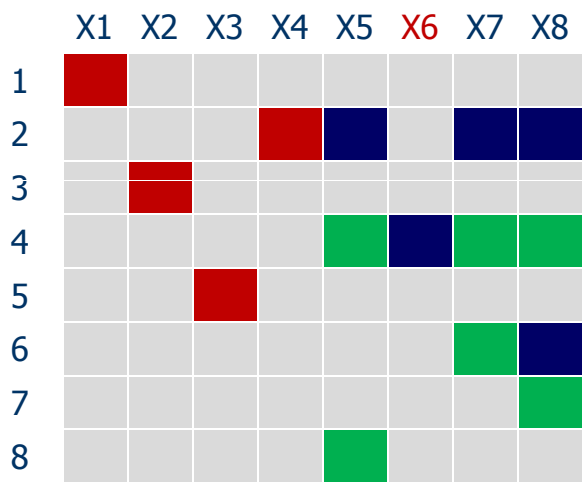


Propagación de restricciones sobre filas, columnas y diagonales: X3::[5..8]. Obsérvese que X6 tiene un único valor posible, pero en la búsqueda con labeling no nos aprovechamos de ello.

ECL^{i}PS^e El problema de las N reinas

```
:-lib(ic).

nqueens(N,Xs):-
  dim(Xs,[N]),
  Xs[1..N]::1..N,
  alldifferent(Xs[1..N]),
  (for(I,1,N-1), param(Xs,N) do
    (for(J,I+1,N), param(Xs,I) do
     Xs[I] #\= Xs[J] + I-J,
     Xs[I] #\= Xs[J] + J-I)),
  labeling(Xs[1..N]).
```

% Propagación de restricciones



Propagación de restricciones sobre filas, columnas y diagonales:
Al poner X4=2, X6 se queda sin valores posibles => backtracking.

ECL^{i}PS^e Estrategias de búsqueda

- **labeling(L)** selecciona las variables por el orden en el que aparecen en el programa y selecciona sus valores de menor a mayor.

- ECL^{i}PS^e nos permite utilizar heurísticas para la selección de variables y valores, lo que nos permite alterar el orden de elección (puntos de decisión) y resolver problemas de una forma más eficiente, p.ej.
  - Primero las variables con un dominio más reducido (reduce el número de nodos internos del árbol de búsqueda).
  - Primero los valores centrales (para aplicar más restricciones).

ECLiPSe Estrategias de búsqueda

- El predicado **search/6**, incluido en la biblioteca **ic**, nos proporciona distintas estrategias de búsqueda:
  search(L,0,select_var,select_val,método,opcional)
  - L Lista de varibles (pueden ser otras estructuras)
  - 0 si L es una lista de variables (sólo veremos lista)
  - Selección de variable: first_fail, input_order, occurrence...
  - Selección de valor: indomain, indomain-middle,...
  - Método de búsqueda: complete, incomplete, user-defined
  - Argumentos opcionales: contar backtrackings - backtrack(N), ...
- **labeling(L)** se puede expresar como
  search(L,0,input_order,indomain,complete,[])

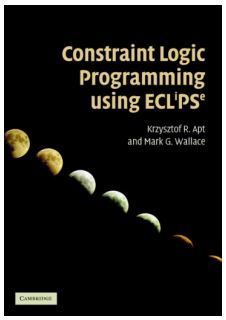
ECLiPSe El problema de las N reinas

```
:-lib(ic).

nqueens(N,Xs):-
  dim(Xs,[N]),
  Xs[1..N]::1..N,
  alldifferent(Xs[1..N]),
  (for(I,1,N-1), param(Xs,N) do
    (for(J,I+1,N), param(Xs,I) do
     Xs[I] #\= Xs[J] + I-J,
     Xs[I] #\= Xs[J] + J-I)),
  search(Xs[1..N],
    0,input_order,indomain,
    complete,[backtrack(B)]),
write("Número backtrackings: "),
writeln(B).
```

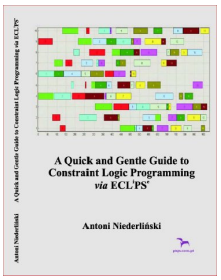
Estrategia de búsqueda	B
Para N=16	
`search(Xs[1..N], 0, input_order, indomain, complete, [backtrack(B)]`	542
`search(Xs[1..N], 0, first_fail, indomain, complete, [backtrack(B)]`	3
`search(Xs[1..N], 0, occurrence, indomain, complete, [backtrack(B)]`	542
`search(Xs[1..N], 0, most_constrained, indomain, complete, [backtrack(B)]`	3
Para N=160	
`search(Xs[1..N], 0, most_constrained, indomain, complete, [backtrack(B)]`	3
`search(Xs[1..N], 0, occurrence, indomain, complete, [backtrack(B)]`	--
`search(Xs[1..N], 0, first_fail, indomain, complete,[backtrack(B)]`	0

Bibliografía

- Krzysztof Apt & Mark Wallace:
  Constraint Logic Programming using ECL^{i}PS^e,
  Cambridge University Press, 2006.
  ISBN 978-0-521-86628-6



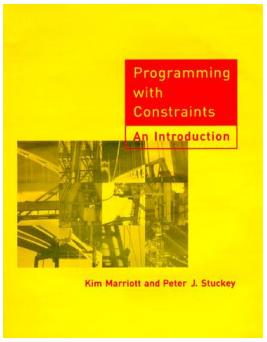
- Antoni Niederliński:
  A Quick and Gentle Guide to Constraint Logic Programming via ECL^{i}PS^e, pkjs.com.pl, 2011.
  ISBN 978-83-62652-08-2
  http://www.anclp.pl/



Bibliografía

Bibliografía complementaria

- Kim Marriott & Peter J. Stuckey:
  Programming with Constraints. An Introduction, MIT Press, 1998.
  ISBN 0-262-13341-5



- William F. Clocksin
  & Christopher S. Mellish:
  Programming in Prolog,
  Springer, 5th edition, 2003.
  ISBN 3-540-00678-8