PRÁCTICA 2:
PROBLEMAS DE SATISFACCIÓN DE RESTRICCIONES.
SESIÓN 2

Técnicas de los Sistemas Inteligentes
Curso 2014-2015
3er. Curso Grado en Informática

Indice

- Conocer
  - arrays en ECLiPSe,
    - Antes conoceremos “estructuras”
  - cómo recorrer arrays
- Necesarios para
  - Modelar problemas que usan índices y conjuntos de variables.
  - Modelar el problema de la N-Reinas
- Nos servirá para
  - Entender la necesidad de heurísticas generales
  - Conocer algunas heurísticas generales
- Usarlas
  - Con el problema de las N-Reinas.

Estructuras

- In ECLiPSe, structure fields can be given names. This makes it possible to write structures in a more readable and maintainable way. Such structures first need to be declared by specifying a template like:

  - :- local struct(book(author, title, year, publisher)).

- Structures with the functor book/4 can then be written as

  - book{}
  - book{title:'tom sawyer'}
  - book{title:'tom sawyer', year:1876, author:twain}

- which, in canonical syntax, correspond to the following:

  - book(_, _, _, _)
  - book(_, 'tom sawyer', _, _)
  - book(twain, 'tom sawyer', 1876, _)

- There is absolutely no semantic difference between the two syntactical forms. The special struct-syntax with names has the advantage that

  - the arguments can be written in any order
  - “dummy” arguments with anonymous variables do not need to be written
  - the arity of the structure is not implied (and can be changed by changing the declaration and recompiling the program)

- Sometimes it is necessary to refer to the numerical position of a structure field within the structure, e.g. in the arg/3 predicate:

  - arg(3, B, Y)

- When the structure has been declared as above, we can write instead:

  - arg(year of book, B, Y)

- Declared structures help readability, and make programs easier to modify. In order not to lose these benefits, one should always use curly-bracket and of-syntax when working with them, and never write them in canonical syntax or referring to argument positions numerically.

Arrays en ECLiPSe

- Array "es como un" vector: Vector(a,b,c,d)
- Usa el functor "[]" para representar arrays.
  - `A1 = [](1,2,3)`
  - `A2 = [](_, apple, orange, _)`
  - Array multidimensional ⇒ estructuras array anidados
  - `M = []([](1,2,3), [](4,5,6))`
- Acceso a elementos vía subíndices:
  - `X is M[1,2] + M[2,3].`
- Ejemplo
  - `?- M = []([](2, 3, 5), [](1, 4, 7)),`
  - `X is M[1,2] + M[2,3].`
  - `X = 10`
  - `Yes`

- Creación (o consulta de la dimensión):

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

?- dim(M4, [3, 4, 2, 1]).
M4 = …
Yes

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

- ¡Cuidado porque ECLiPSe no evalúa subíndices como en lenguajes imperativos!

  Se hace sólo en el contexto de expresiones

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

  - prints: `[](1, 2, 3), [](4, 5, 6), [](7, 8, 9))[2, 3]`
    `[2,3]` `!!!`

  - La forma correcta de

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

  - prints: `6`

- **`subscript(Matrix,[2,3],X).`**

Ejemplo (secuencia)

- Convertir un vector en una lista:
  - Dim(Vector, [N]).
  - Vector es un array unidimensional
  - Vector[1..N] es una lista

- Variables: Vector[10]
- Dominios: Vector[i] ={0..9}
- Restricciones: Vector[i] es el numero de veces que aparece i en Vector.

- Acceso a sub-matrices:
  - *(Notar que las submatrices se "convierten" en listas)*

```
?- 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].
   Lista is Matrix[1..3,1..3]

Row2 = [4, 5, 6]
SubRow2 = [5, 6]
Column1 = [1, 4, 7]
Yes
```

Iteración en ECLIPSE

- En Prolog, la única forma de expresar iteración es mediante recursión, lo que puede ser terrible en determinados casos
- ECLiPSe suministra bucles do :

```
( EstructIterativa do
Objetivos)
```

- Refresco de iteración
-
```
(foreach (X,Lista) do
    <body>
)
```
-
```
(for  (I, 1,N) do
    <body>
)
```
- Uso de **param:** para hacer paso de una variable global al cuerpo de un bucle

```
(for (I, 1, N) do
   (for (J,1,N), param(I) do
        K is I*J,
        write(K), write(' ')))
```

Ejemplo cuadrado mágico

- Variables: Matriz(N,N).
- Dominios: Matriz[i,j] = {1..N}
- Restricciones:
  - Todos los valores de cada Fila(i), i =1..N son diferentes
  - Todos los valores de cada Col(j), j = 1..N son diferentes.

Iteración sobre un array

foreachelem

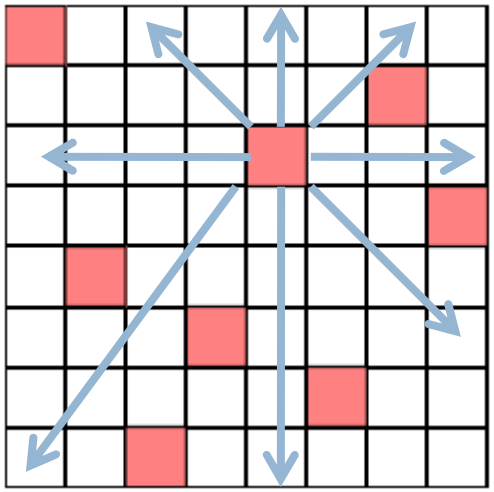
```
write_array(A) :-
   dim(A,[5]).
   (foreachelem(El, A) do
        write(El)).
```

for

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

N-reinas

- Situar n reinas en un tablero de n x n
- De forma que no se ataquen mutuamente
- Una reina ataca a todas las celdas en dirección horizontal, vertical y diagonal.



N-reinas

X1,..,Xn variables que representan las columnas

Cada Xi establece la posición de una reina en la columna i.

- Xi = j
  SI en la columna i, fila j hay una reina

Asignar valores a Xi tal que

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

	X1	X2	X3	X4	X5	X6	X7	X8
1							X7≠ X4--3	
2						X6≠ X4-2		
3					X5≠ X4-1			
4				X4=4	X5≠X4	X6≠X4	X7≠X4	X8≠X4
5					X5≠ X4+1			
6						X6≠ X4+2		
7							X7≠ X4+3	
8								X8≠ X4+8

n-reinas con ic

```
:-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]).
```

- Hacer pruebas con N=4, 8,16,24,32.

- Antes de llevar a cabo las restricciones, todas las posiciones están libres.

	X1	X2	X3	X4	X5	X6	X7	X8
1								
2								
3								
4								
5								
6								
7								
8								

n-reinas: propagación

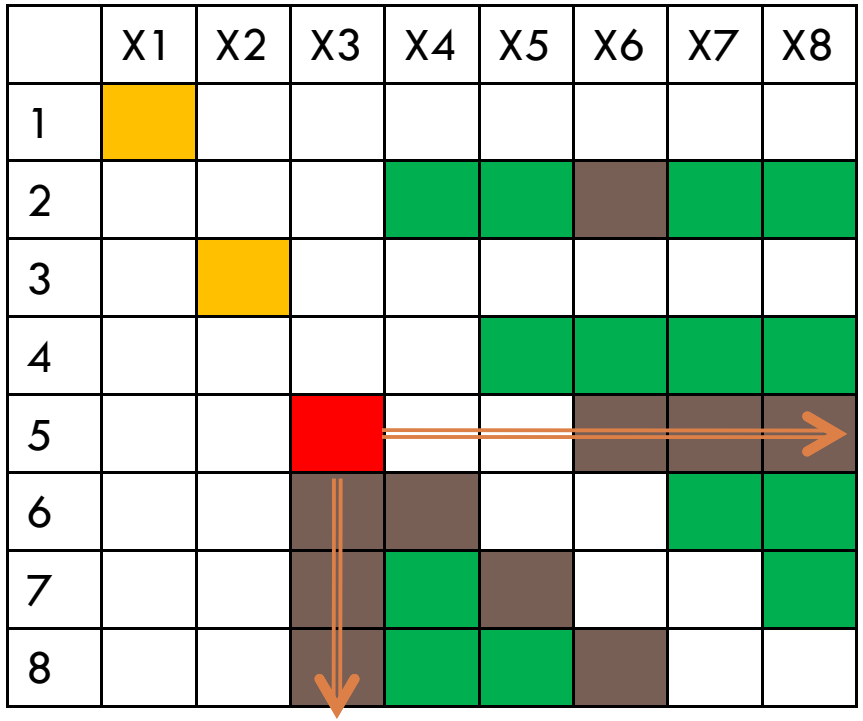
- Cada vez que se asigna un valor a Xi (en este caso X1) se propagan las restricciones sobre filas, columnas y diagonales.
- En azul las casillas "prohibidas" por las restricciones asociadas a X1
- En blanco las casillas que quedan descartadas por asignar un valor a X1

	X1	X2	X3	X4	X5	X6	X7	X8
1								
2								
3								
4								
5								
6								
7								
8								

- El dominio de X2 se actualizó a [3..8].
- Prueba con 3 y propaga restricciones

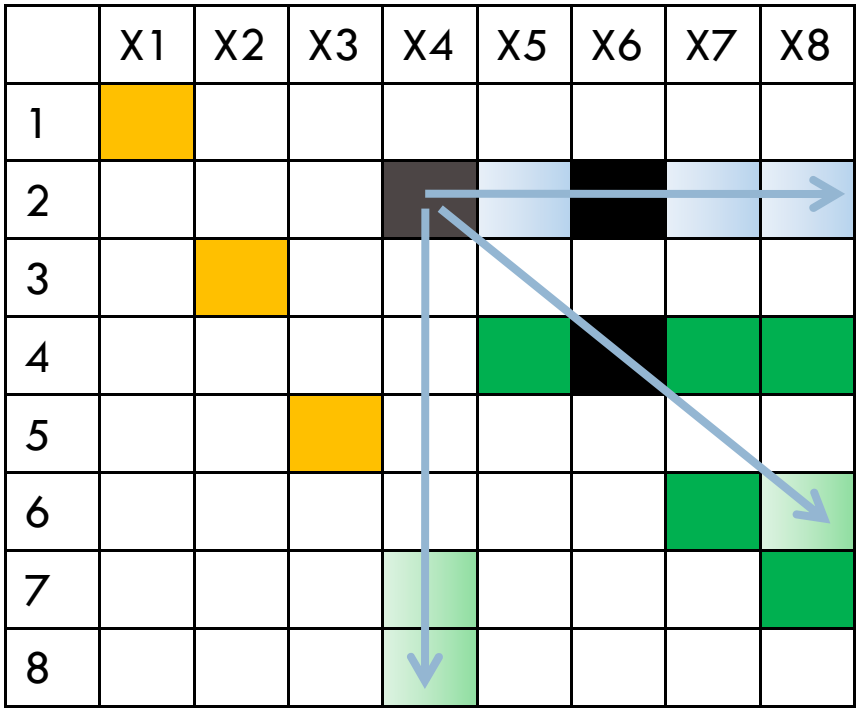
	X1	X2	X3	X4	X5	X6	X7	X8
1								
2								
3								
4								
5								
6								
7								
8								

- X3 ::[5,8]
- Si X3 = 5
  - Observar que X6 sólo tiene un único valor posible
- Pero con la técnica que se está usando ahora (labeling) no puede seleccionarse este valor.
  - Porque selecciona las variables en el orden en que se han definido.



n-reinas: propagación

- Prueba con X4= 2
- X6={Ø} => inconsistente
- Backtracking.



Heurísticas para la selección de variables y valores

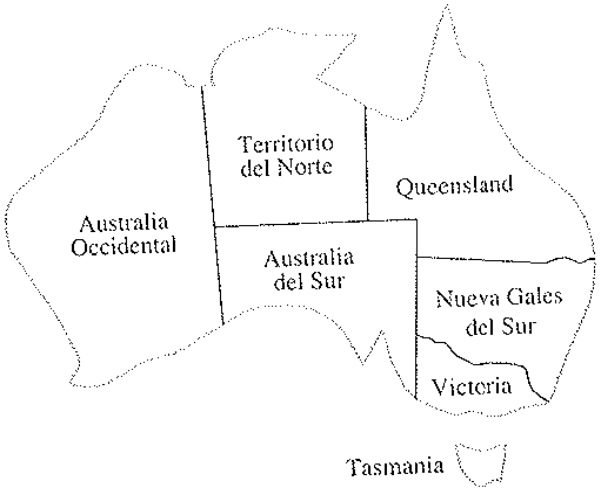
- `labeling(Lista)`
  - selecciona variables por el orden introducido en el programa y selecciona valores de menor a mayor
- Solucionar problemas computacionalmente costosos require usar heurísticas que alteren el orden de elección(puntos de decisión)
  - Sobre variables
  - Sobre valores
- Por ejemplo:
  - Seleccionar primero las variables con un dominio más pequeño
  - Seleccionar valores centrales para poder aplicar más restricciones.

Variable y ordenamiento de valor

- Algunas de las heurísticas generales más comunes son:
  - Mínimos valores restantes
  - Grado heurístico
  - Valor menos restringido

Mínimos valores restantes

- En principio no hay ningún criterio en la selección de la variable.
- Uso de la heurística de mínimos valores restantes (MVR) o de la variable más restringida.
  - Se selecciona la variable con el dominio más pequeño
  - También llamada
    - Primera en fallar (first-fail)



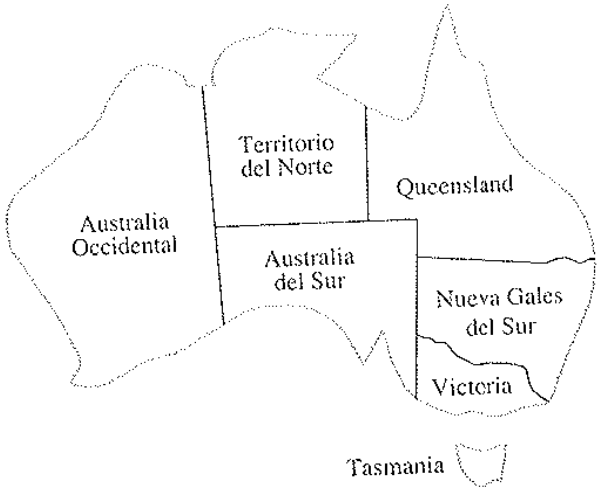
Mínimos valores restantes

Problema	Vuelta atrás	VA + MVR	Comprobación hacia delante	CD + MVR	Mínimo conflicto
EE.UU.	(> 1.000K)	(> 1.000K)	2K	60	64
n-reinas	(> 40.000K)	13.500K	(> 40.000K)	817K	4K
Zebra	3.859K	1K	35K	0,5K	2K
Aleatorio 1	415K	3K	26K	2K	
Aleatorio 2	942K	27K	77K	15K	

Grado heurístico

- Selecciona la variable, entre las no asignadas, que esté implicada en el mayor número de restricciones

La heurística MVR es más poderosa, pero el grado heurístico es útil en casos de empate



Valor menos restringido

- Una vez seleccionada una variable debe decidirse un orden para examinar sus valores.
- Heurística del valor menos restringido: se prefiere el valor que excluye el menor número de alternativas en las variables vecinas del grafo.

AO=roja, TN=verde, Q=¿?
Q puede ser azul o roja

Si azul entonces AS no tiene alternativa

La heurística opta por Q=roja

- Aporta diferentes métodos de búsqueda
- Permite declarar distintas heurísticas de selección de variable y valor.
- Sólo configurando sus argumentos
- Extensible por el usuario
- En la librería ic

parámetros de search

- Documentación en
  - http://eclipseclp.org/doc/bips/lib/ic/search-6.html
- search(L,0,select_var,select_val,método,optional)
  1. L Lista de varibles (pueden ser otras estructuras)
  2. 0 si L es una lista de variables (sólo veremos lista)
  3. Selección de variable: first_fail, input_order,ocurrence ...
  4. Selección de valor: indomain, indomain-middle,...
  5. Método de búsqueda: complete, incomplete, user-defined
  6. Argumentos opcionales: contar backtrackings - backtrkack(N), ...
- labeling(L)se puede expresar como
  - search(L,0,input_order,indomain,complete,[])

- Selección de variable
  - Mínimos valores restantes: first_fail
    - Se selecciona la variable con el dominio más pequeño
  - Grado heurístico:occurrence
    - Se selecciona la variable con el mayor número de restricciones asociadas

Ejemplos y resultados

```
:-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("Numero backtrackings: "),
writeln(B).
================================
Numero de Backtrackings: 542
```

Modo 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