



## Tema 6

# Búsqueda con espacios estructurados “Restricciones”

*Año Académico 2018/19*

*Profesores:*

*Holger Billhardt, Alberto Fernández, Sascha Ossowski, Miguel Ángel Rodríguez*

# Módulo II: Agentes basados en Búsqueda

## Resumen:

2. Agentes basados en búsqueda
  - 2.1. Búsqueda en espacios de estados
  - 2.2. Búsqueda no-informada
  - 2.3. Búsqueda heurística
  - 2.4. Búsqueda multiagente
  - 2.5. Búsqueda con espacios estructurados
    - Búsqueda basada en restricciones

# Problemas de representaciones “monolíticas” de estados

- *representación* de los conocimientos a priori del agente
  - estado inicial: símbolo ( $s_0$ )
  - estados sucesores: función *expandir* (  $s_7 \mapsto \{s_8, s_{10}, s_{12}, s_{27}, s_{112}\}$  )
  - estados meta: función *meta?* (  $s_{112} \mapsto \text{true}$  )
  - coste de operador: función (  $c(s_7, s_{112}) \mapsto 5$  )
  - heurísticas: función  $h^*$  (  $s_7 \mapsto 235$  )
- se abstrae completamente de las características de los estados
  - ¿bloque *A* encima de bloque *B*? , ¿el agente se encuentra en *Radovac*? , ...
- Problema:
  - complejidad de la representación del conocimiento (p.e. *expandir*)
  - mantenibilidad del conocimiento
- Solución:
  - tomar en cuenta la “estructura” de los estados
  - definir un estado no por su nombre sino por el conjunto de sus *propiedades*

# Problemas de Satisfacción de Restricciones

## Estructura de estados:

- Un estado está compuesto por un conjunto de  $n$  variables que pueden tomar diferentes valores
- Un estado es un estado meta si los valores que tienen sus variables cumplen una serie de *restricciones*
- Muchos dominios reales se pueden describir de esta forma: juegos unipersonales (Sudoku), generación de horarios en institutos, planificación de la producción, etc.

## Definición:

- Un **problema de satisfacción de restricciones** (*Constraint Satisfaction Problem, CSP*) es una tripleta  $(X, D, R)$ 
  - $X = \{x_1, \dots, x_n\}$  es un conjunto finito de *variables*
  - $D: X \rightarrow V$  es una función total que asigna un *dominio* (conjunto finito de valores de  $V_i$ ) a cada variable. Frecuentemente se escribe  $D_i$  en vez de  $D(x_i)$  para referirse al dominio de la variable  $x_i$
  - $R = \{R_1, \dots, R_k\}$  es un conjunto finito de *restricciones* tal que cada  $R_i$  es un predicado sobre un subconjunto de las variables de  $X$ . Formalmente:  $R_i(x_1, \dots, x_l) \subseteq D_1 \times \dots \times D_l$

# Ejemplo: $n$ -reinas

## Problema de los $n$ reinas:

- Posicionar  $n$  reinas en un tablero  $n \times n$ , tal que ninguna de ellas está amenazada por otra

## 4-reinas como CSP:

- $X = \{x_1, \dots, x_4\}$  (número de las filas)
- $D = \{D_1, \dots, D_4\}$  siendo  $D_i = \{1, 2, 3, 4\}$  (nº de las columnas)
- $R = \{R_1\}$  siendo  $R_1(x_1, x_2, x_3, x_4) = \{(2, 4, 1, 3), (3, 1, 4, 2)\}$

## Ejemplo: 4 reinas

	1	2	3	4
$X_1$			♛	
$X_2$				♛
$X_3$	♛			
$X_4$			♛	

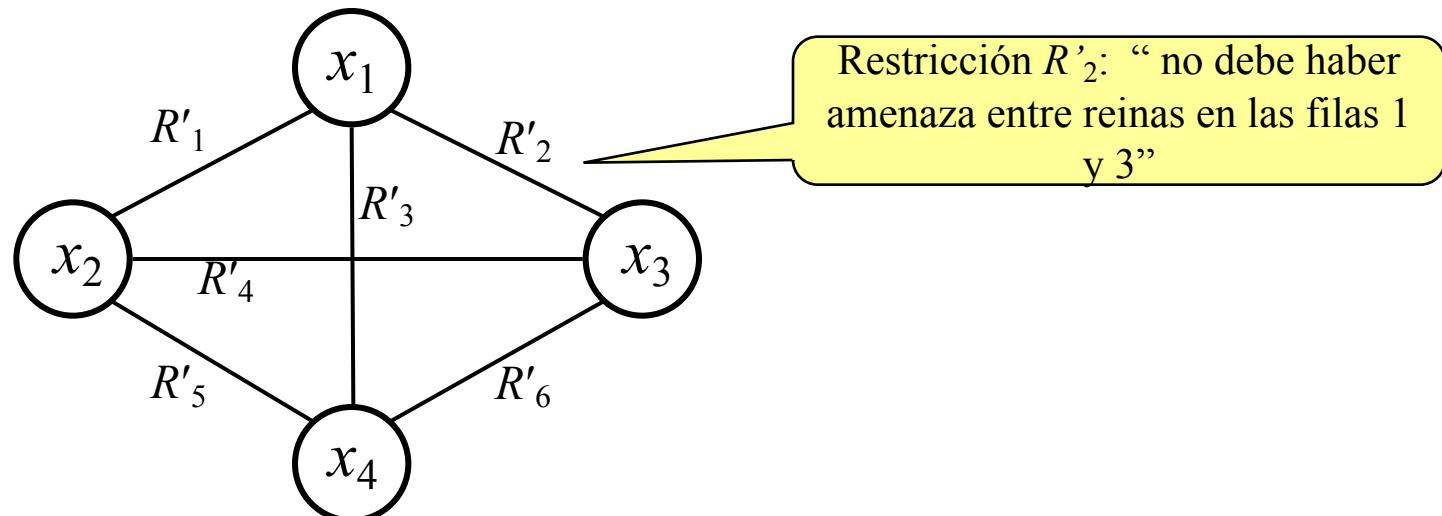
## Nótese:

- Normalmente  $R$  es un coto de *restricciones binarias* (involucrando sólo 2 variables)
- En el ejemplo:  $R' = \{R'_1, \dots, R'_6\}$  refiriéndose cada  $R'_i$  a la amenaza entre dos filas
  - Ejemplo:  $R'_2$  expresa que no debe haber amenaza entre reinas en las filas 1 y 3
  - $R'_2(x_1, x_3) = \{(1, 2), (1, 4), (2, 1), (2, 3), (3, 2), (3, 4), (4, 1), (4, 3)\}$
- Cualquier CSP puede expresarse sobre la base de restricciones binarias

# CSP como grafo

Los CSPs de restricciones binarias se suelen representar como grafos

- Cada variable  $x_i$  del CSP es representada por un nodo
- Cada restricción binaria  $R_i(x_j, x_k)$  se representa por un arco entre los nodos  $x_j$  y  $x_k$
- Ejemplo: grafo correspondiente al problema de las 4-reinas



- CSPs con restricciones  $n$ -áreas se pueden representar mediante *hipergrafos*

# Problemas de Satisfacción de Restricciones

## Solución a un CSP

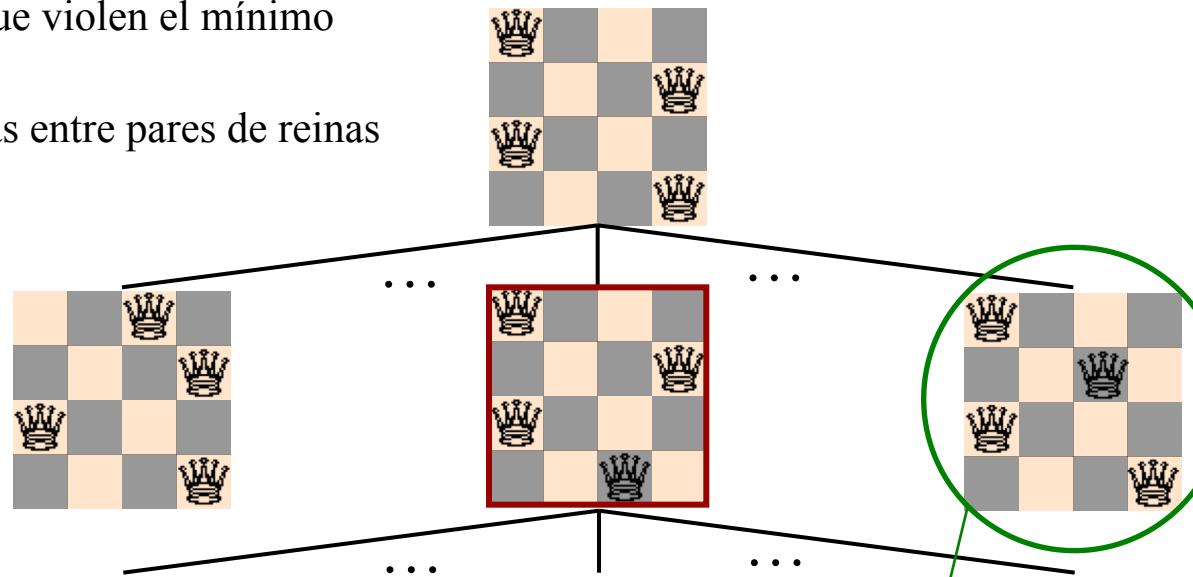
- Una *asignación* es una función parcial  $\sigma: X \rightarrow D(X)$ 
  - Habitualmente se escribe “ $x_1 = v_a$ ” por “ $\sigma(x_1) = v_a$ ” y se describe una asignación de forma extensiva, es decir  $\sigma = \{x_1 = v_a, x_3 = v_b, x_7 = v_c, \dots\}$
  - Una asignación es *completa* si está definida para todas las variables  $X$  del CSP
  - Una asignación  $\sigma$  *cumple* con una restricción  $R_i$  si
- ( $\sigma(x_1), \dots, \sigma(x_l)\right) \in R_i(x_1, \dots, x_l)$
- Una asignación completa  $\sigma$  es una *solución* si cumple todas las restricciones en  $R_i \in R$
- *Ejemplos* para el caso de las 4 reinas
  - $\sigma = \{x_1 = 2, x_2 = 4, x_3 = 3\}$  es una asignación parcial que cumple  $R'_2$
  - $\sigma = \{x_1 = 1, x_2 = 2, x_3 = 3, x_4 = 4\}$  es una asignación total que no cumple  $R'_2$
  - $\sigma = \{x_1 = 2, x_2 = 4, x_3 = 1, x_4 = 3\}$  es una solución al problema de las 4 reinas

# Solución con métodos de búsqueda

## Búsqueda con asignaciones completas

- *Estado*: asignación completa (valores para *todas* las variables)
- *Operador*: modificar el valor de una variable en la asignación
- *Meta?*: asignación que *cumple* todas las restricciones
- *Coste*: cero (la longitud de un camino hasta un nodo meta es irrelevante)
- *Heurística del conflicto mínimo*:
  - Preferir los sucesores que violen el mínimo número de restricciones
  - 4-reinas: n° de amenazas entre pares de reinas

columna 3			
fila 2	3	1	3
4	5	2	3
3	2	5	4
3	3	1	3



Significado de la tabla. Ejemplo:

“En el estado que resulta de colocar la reina de la fila 2 en la columna 3 hay 2 pares de reinas que se amenazan”

# Solución con métodos de búsqueda

Nótese:

- A cada una de las  $n$  variables se pueden asignar  $d$  valores: cada nodo tiene  $n \cdot d$  sucesores
- En cada camino, a cada una de las  $n$  variable se cambia el valor sólo 1 vez. Por tanto, se genera un árbol con aprox.  $(n \cdot d)^n$  hojas – pero sólo hay  $d^n$  posibles asignaciones!
- Equivalencia de caminos por *comutatividad*: el *orden* en el se cambian los valores de las variables es irrelevante

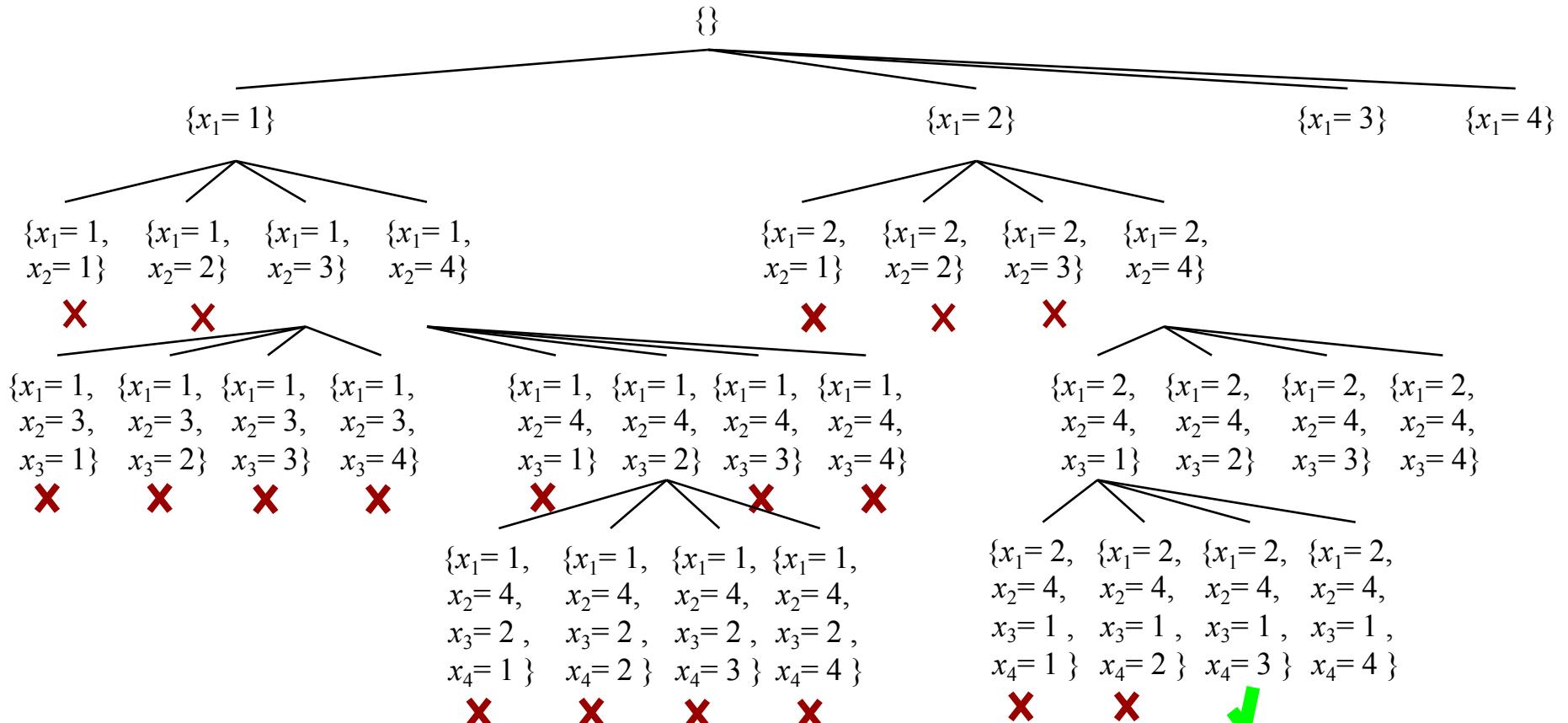
## Búsqueda con asignaciones parciales

- *Estado*: asignación parcial (valores asignados a *algunas* variables)
- *Operador*: elegir un valor para una variable *no asignada*  
(el posible estado sucesor ha de cumplir  $R$ )
- *Meta?*: solución (asignación *completa* que cumple *todas* las restricciones)
- *Coste*: cero (la longitud de un camino hasta un nodo meta es irrelevante)

# Satisfacción de restricciones con vuelta atrás cronológica

**Algoritmo:** Vuelta atrás cronológica (*chronological backtracking*)

- Búsqueda en profundidad en el espacio de asignaciones parciales
- Ejemplo: 4-reinas



# Satisfacción de restricciones con vuelta atrás cronológica

Función vuelta-atrás-cronológica(CSP,  $\sigma$ ) **devuelve**  $\sigma'$  / *fallo*

**Si**  $|\sigma| = n$  **entonces** % la asignación es completa (i.e. una solución)  
    **devolver**( $\sigma$ )

$x_i \leftarrow \text{elegir-variable-no-asignada}(X, \sigma)$  % orden de elección no afecta la completitud  
dominio  $\leftarrow D_{x_i}$

**Mientras** dominio  $\neq \{\}$  **hacer** % en el dominio quedan valores por probar  
     $v \leftarrow \text{elegir-valor}(\text{dominio})$

$\sigma' \leftarrow \sigma \cup \{x_i = v\}$  % generar nueva asignación parcial

**Si**  $\sigma' \in R$  **entonces** % la nueva asignación parcial es consistente  
        resultado  $\leftarrow$  vuelta-atrás-cronológica(CSP,  $\sigma'$ ) % completarla

**Si** resultado  $\neq \text{fallo}$  **entonces**

**devolver**(resultado)

    dominio  $\leftarrow \text{dominio} \setminus \{v\}$

**Fin** {Mientras}

**devolver**(*fallo*)

**Fin** {vuelta-atrás-cronológica}

# Mejoras: Heurísticas

## Nótese

- La selección de variables (*elegir-variable-no-asignada*) and y de valores (*elegir-valor*) no afecta la completitud (el espacio de búsqueda es finito)
- pero *sí* afecta la complejidad del algoritmo (el tamaño del árbol de búsqueda)

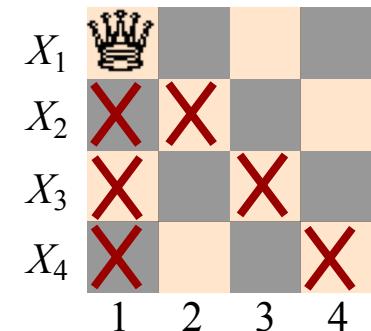
## Heurísticas (para búsqueda con asignaciones parciales)

- Elegir variables: “encontrar ‘callejones sin salida’ cuanto antes”
  - Preferir la variable de *mínimos valores restantes*
  - Preferir la variable con mayor número de restricciones (*grado heurístico*)
- Elegir valores: “generar cuanto antes una solución”
  - Preferir el *valor menos restringido*
  - ...

# Propagación de restricciones

## Propagación de restricciones:

- Idea: usar el conjunto de restricciones para reducir el espacio de búsqueda
- Método:
  - “propagar” valores a través de las restricciones que afectan a sus variables,
  - intentando eliminar de los dominios los valores que no podrán formar parte de una solución
- Ejemplo de las 4 reinas: Si  $\{x_1 = 1\}$  se puede
  - eliminar 1 y 2 de  $D_2$ , i.e  $D_2 = \{3,4\}$  (debido a  $R'_1(x_1, x_2)$ )
  - eliminar 1 y 3 de  $D_3$ , i.e  $D_3 = \{2,4\}$  (debido a  $R'_2(x_1, x_3)$ )
  - eliminar 1 y 4 de  $D_4$ , i.e  $D_4 = \{2,3\}$  (debido a  $R'_3(x_1, x_4)$ )



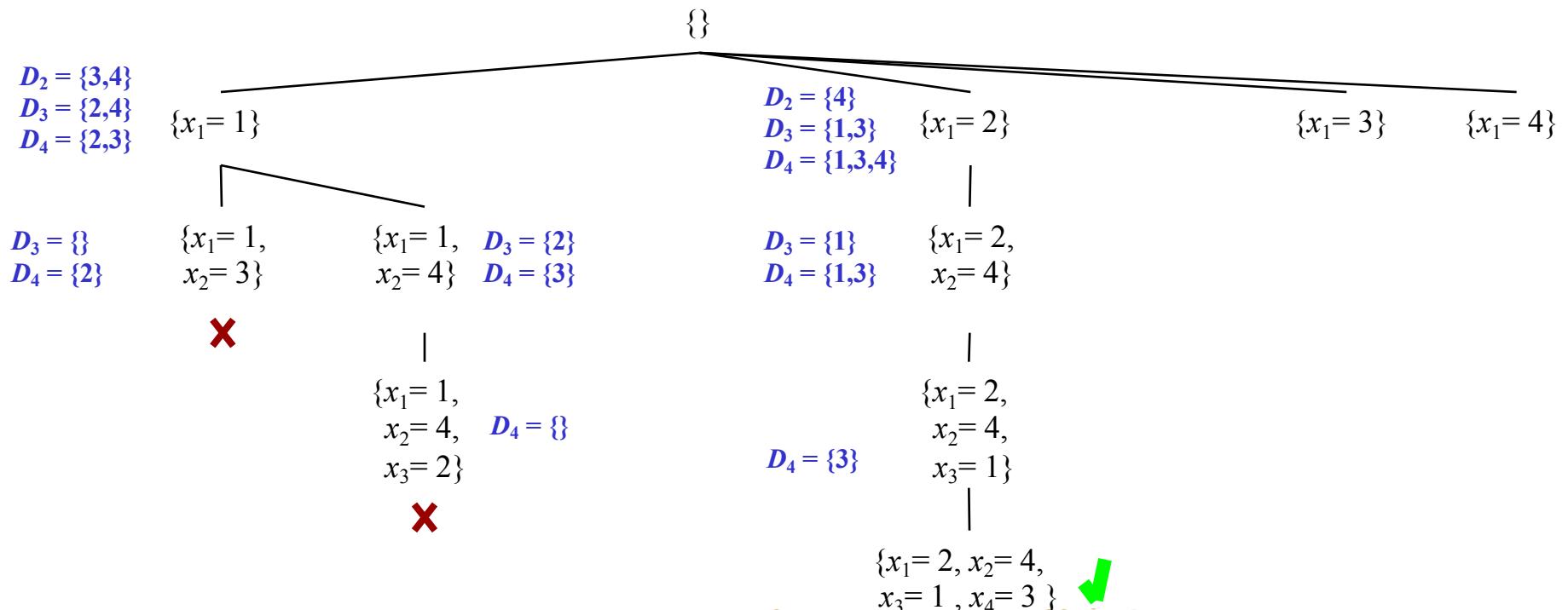
## Aplicación:

- a) *Antes de la búsqueda*, para reducir la cardinalidad de los dominios
- b) *Intercalado con la búsqueda*: filtrar dominios sobre la base de una asignación parcial

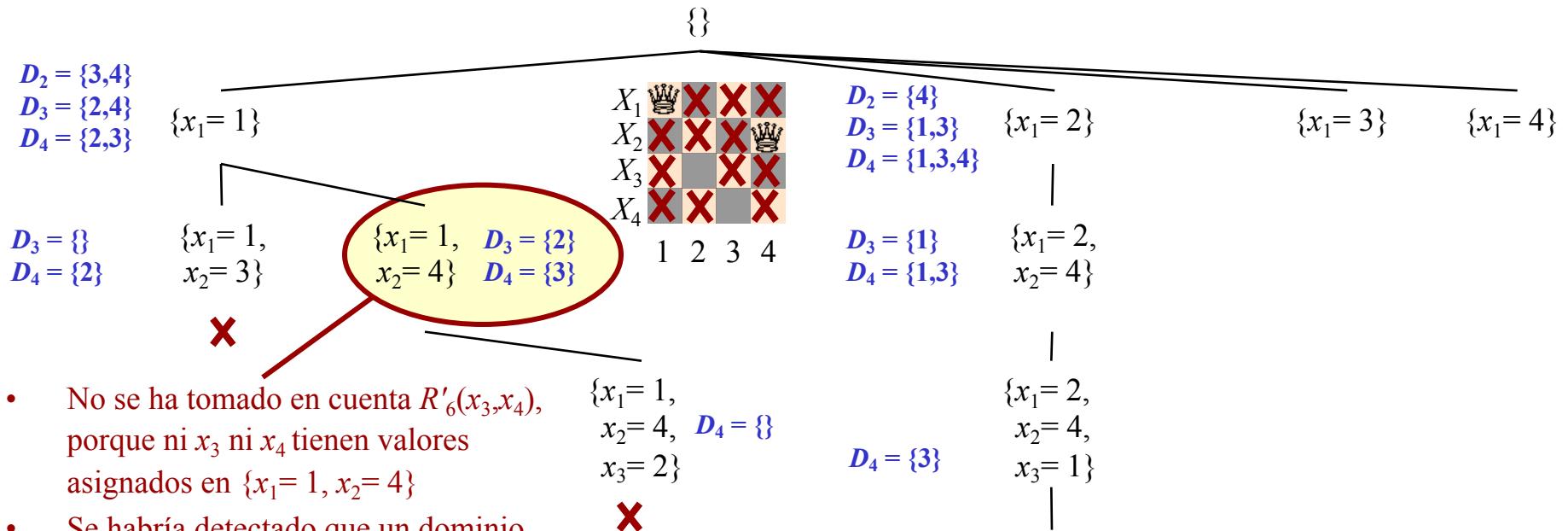
# Propagación de restricciones: comprobación hacia adelante

Búsqueda con vuelta atrás cronológica con comprobación hacia delante (*forward checking*):

- Filtrar dominios después de cada ampliación de la asignación parcial  $\sigma$ .
  - Para todas las restricciones binarias  $R_k$  que involucran una variable  $x_i$  asignada en  $\sigma$  y otra  $x_j$  no asignada en  $\sigma$
  - Para todo valor  $v$  de  $D_j$ : Si  $(\sigma(x_i), v)$  viola  $R_k(x_i, x_j)$  entonces eliminar  $v$  de  $D_j$
- Marcha atrás si algún dominio se queda sin valores



# Transitividad de Restricciones



- No se ha tomado en cuenta  $R'_6(x_3, x_4)$ , porque ni  $x_3$  ni  $x_4$  tienen valores asignados en  $\{x_1 = 1, x_2 = 4\}$
- Se habría detectado que un dominio se queda sin valores
- Y, por tanto, que no se puede ampliar la asignación para que forme una solución
- Hay que ir más allá del *forward checking* y tomar en cuenta la *transitividad* de las restricciones...

# Consistencia de arco

Definición.:

- Un CSP es **arco consistente** si para toda restricción binaria  $R_k(x_i, x_j)$ , se cumple que

$$\forall v \in D_i \exists v' \in D_j (v, v') \text{ cumple } R(x_i, x_j)$$

Ejemplo:

- El CSP de las cuatro reinas con  $D_1 = \{1\}$ ,  $D_2 = \{3,4\}$ ,  $D_3 = \{2,4\}$ , y  $D_4 = \{2,3\}$  no es arco consistente porque (entre otros)
  - Para  $x_3=2$  no existe  $v \in D_4$  tal que  $(2,v) \in R'_7(x_3,x_4)$ )
  - Para  $x_3=4$  no existe  $v' \in D_2$  tal que  $(4,v') \in R'_4(x_3,x_2))$
- Por tanto, para alcanzar arco consistencia,  $D_3$  tendría que quedarse vacío...

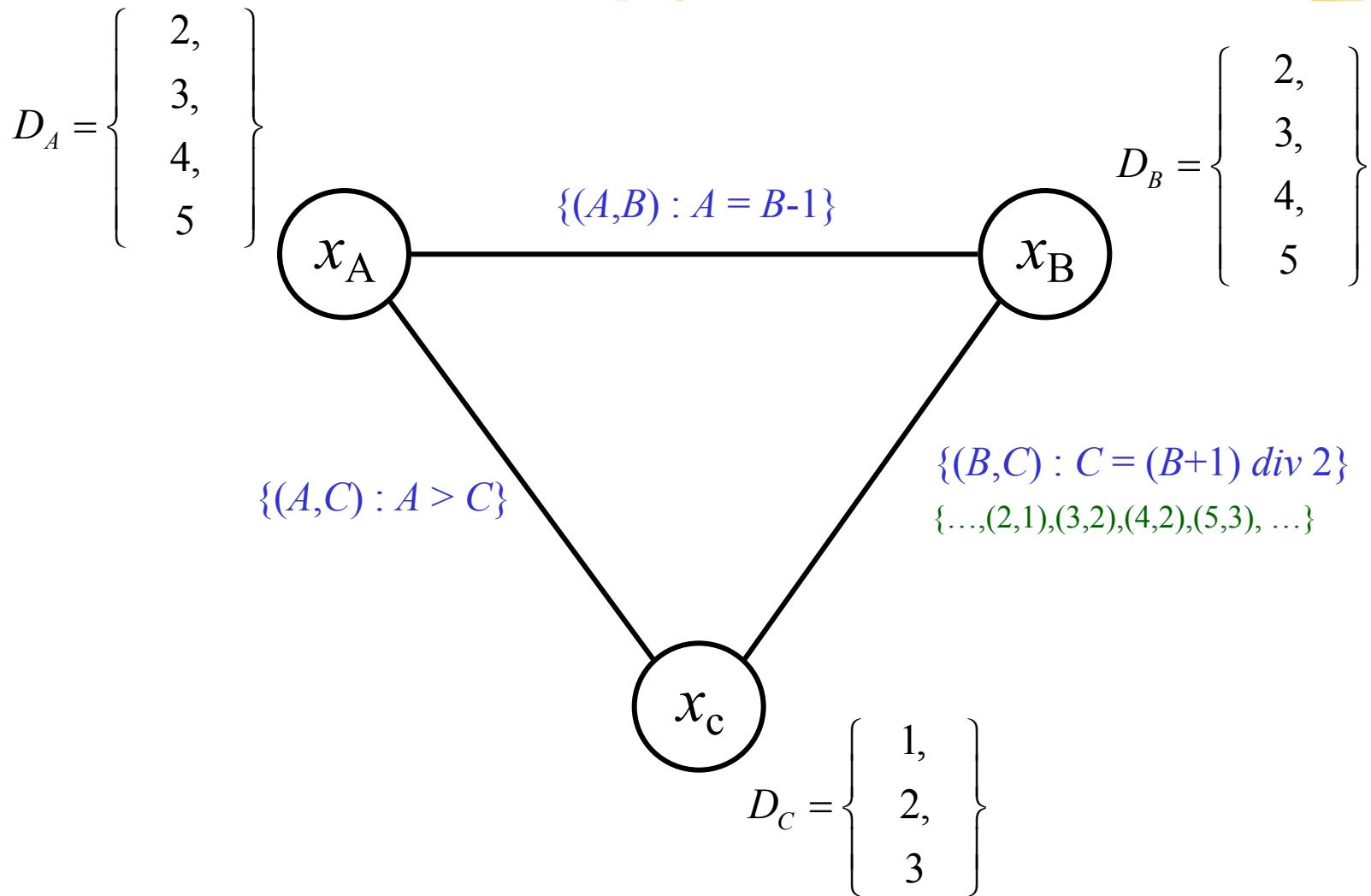
$X_1$				
$X_2$				
$X_3$				
$X_4$				

1    2    3    4

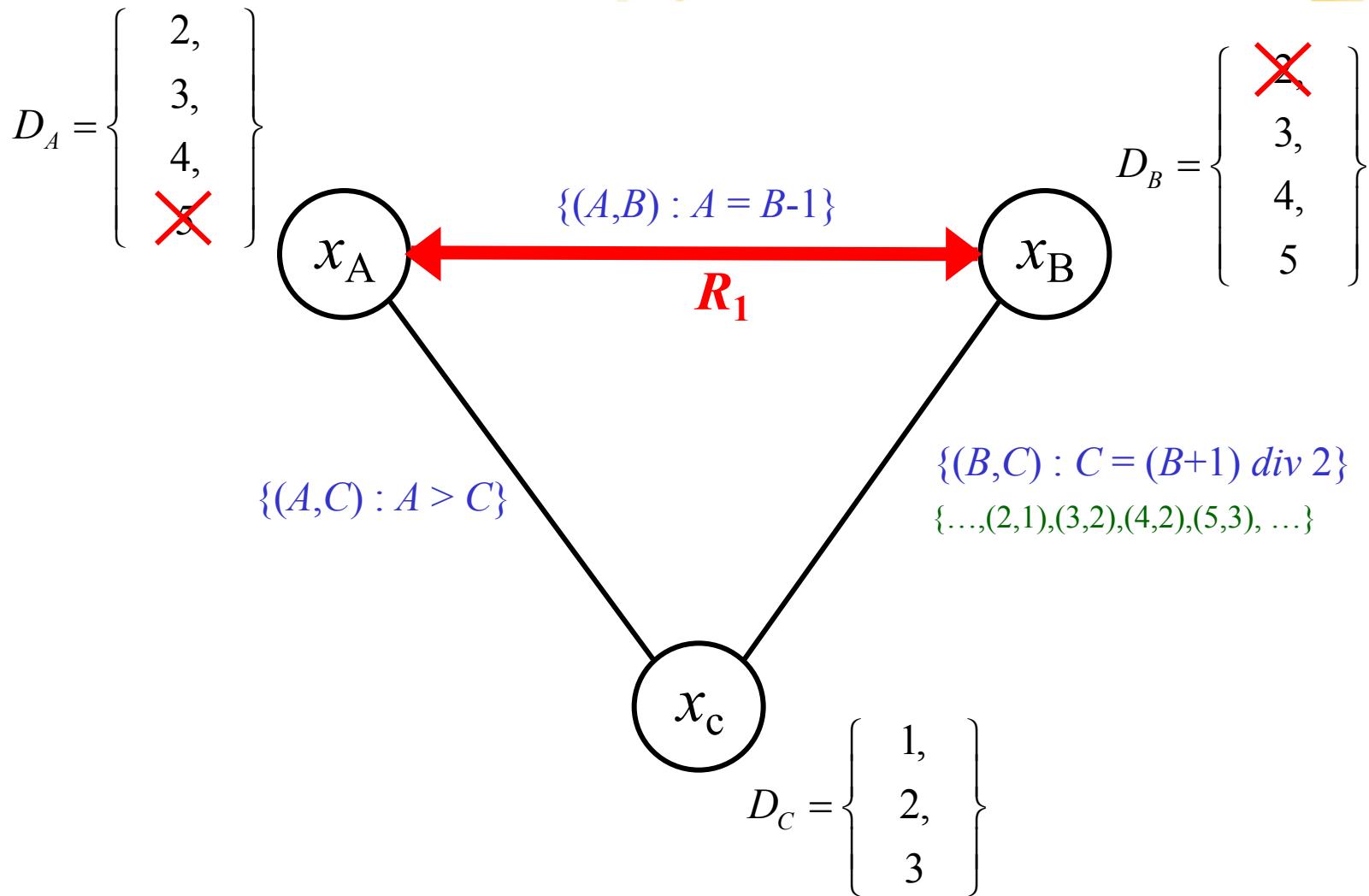
Propagación de restricciones para establecer consistencia de arco:

- Comprobar las restricciones una tras otra, eliminando los valores de los dominios de las variables involucradas en la restricción actual que no cumplen la condición de arco-consistencia
- Al eliminar un valor del dominio  $D_i$  de una variable  $x_i$  involucrada en la restricción  $R_y(x_i, x_j)$ , se puede “estropear” la condición de arco consistencia de otra restricción  $R_z(x_i, x_k)$  que involucra también a esta variable  $x_i$ , por lo que puede ser necesario “propagar” valores varias veces por la misma restricción

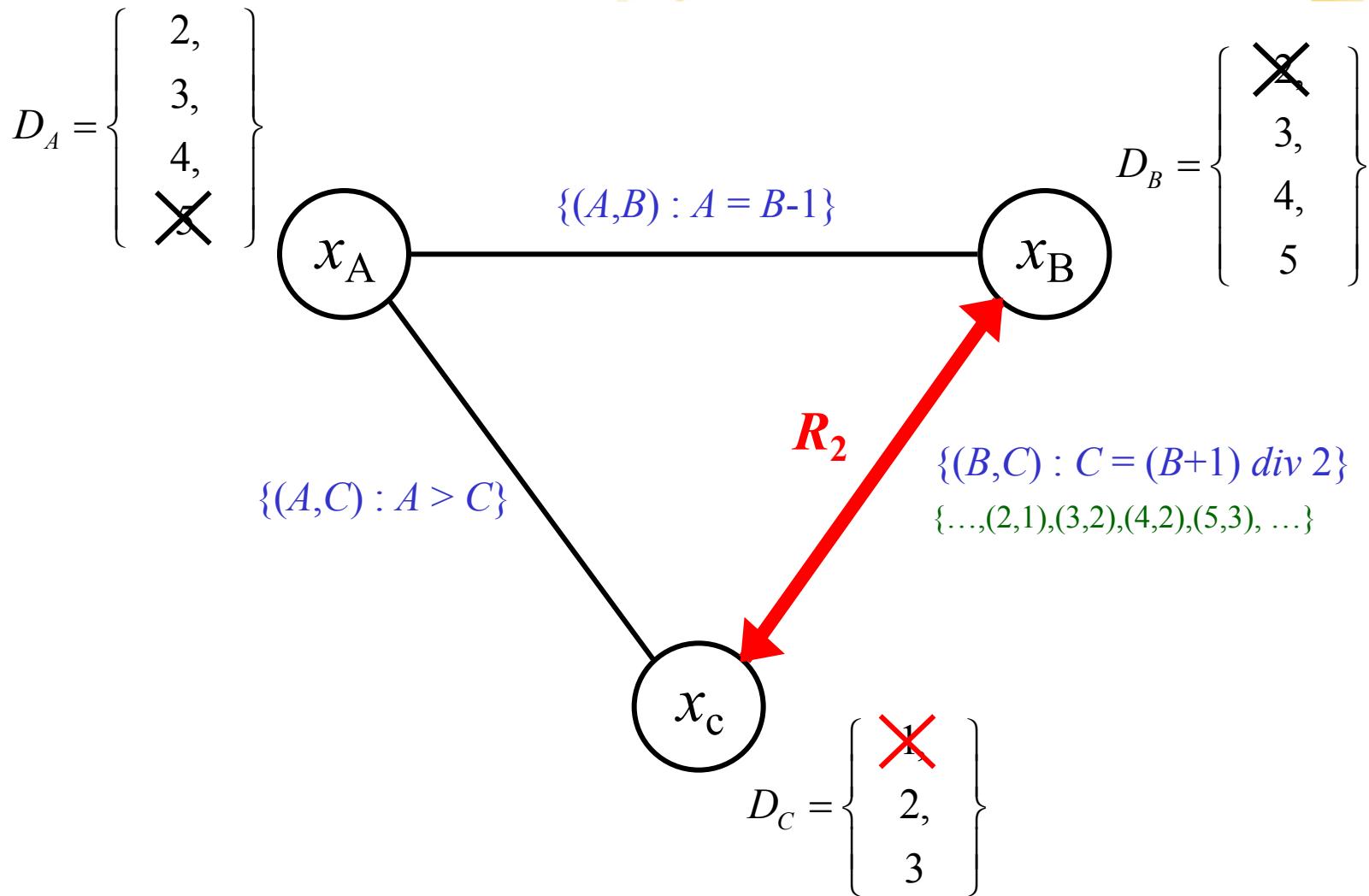
# Algoritmo de arco consistencia



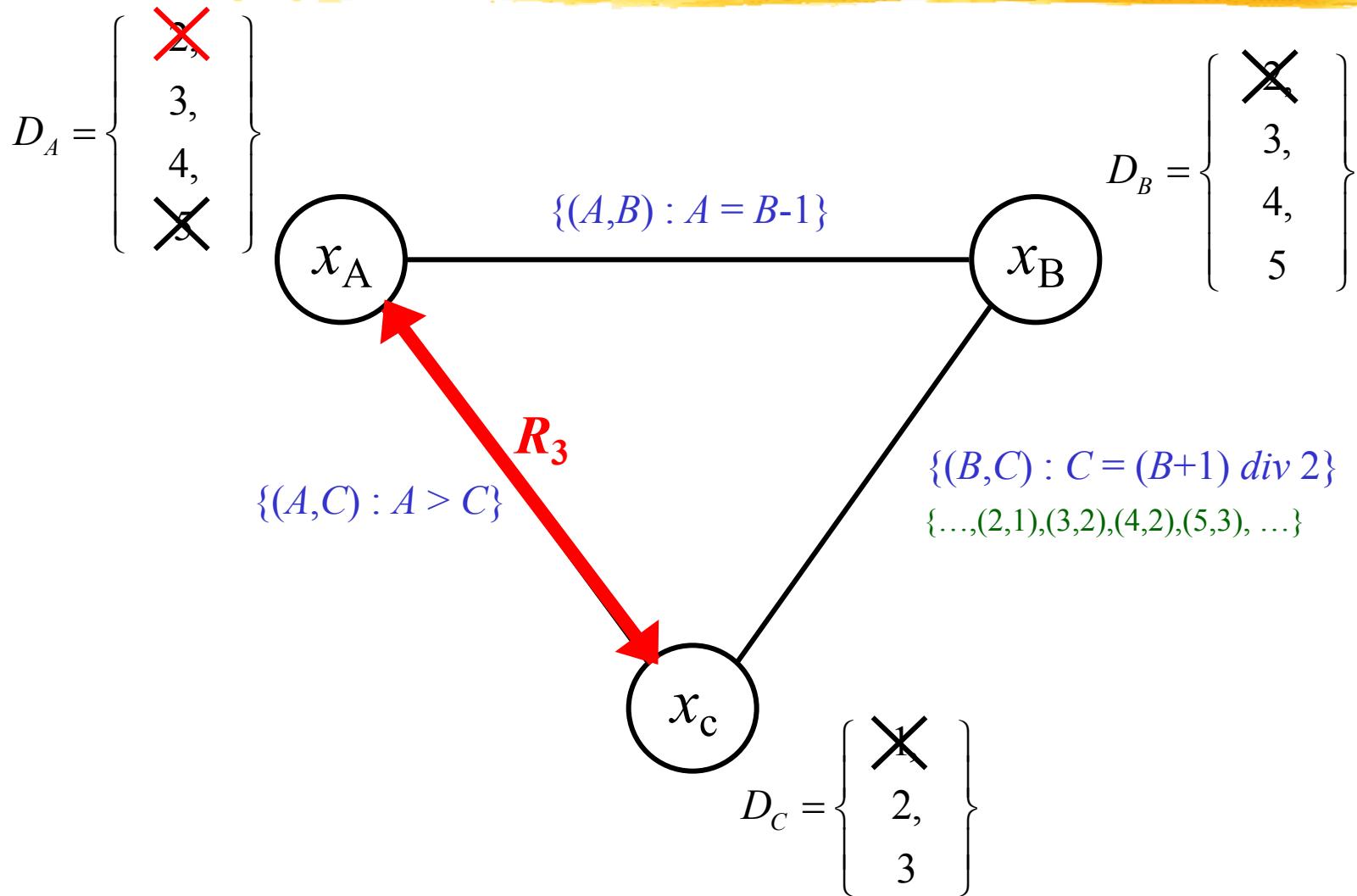
# Algoritmo de arco consistencia



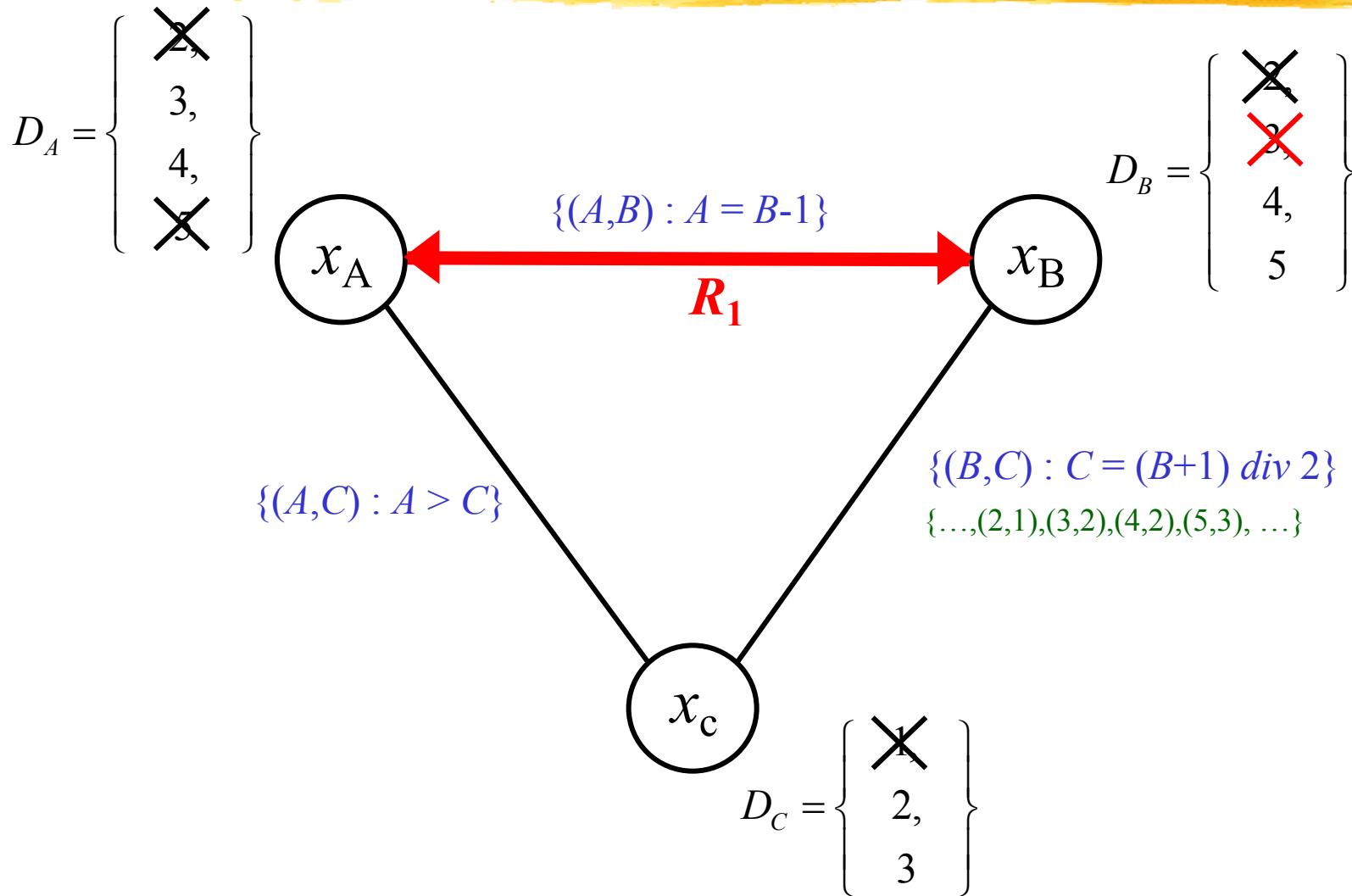
# Algoritmo de arco consistencia



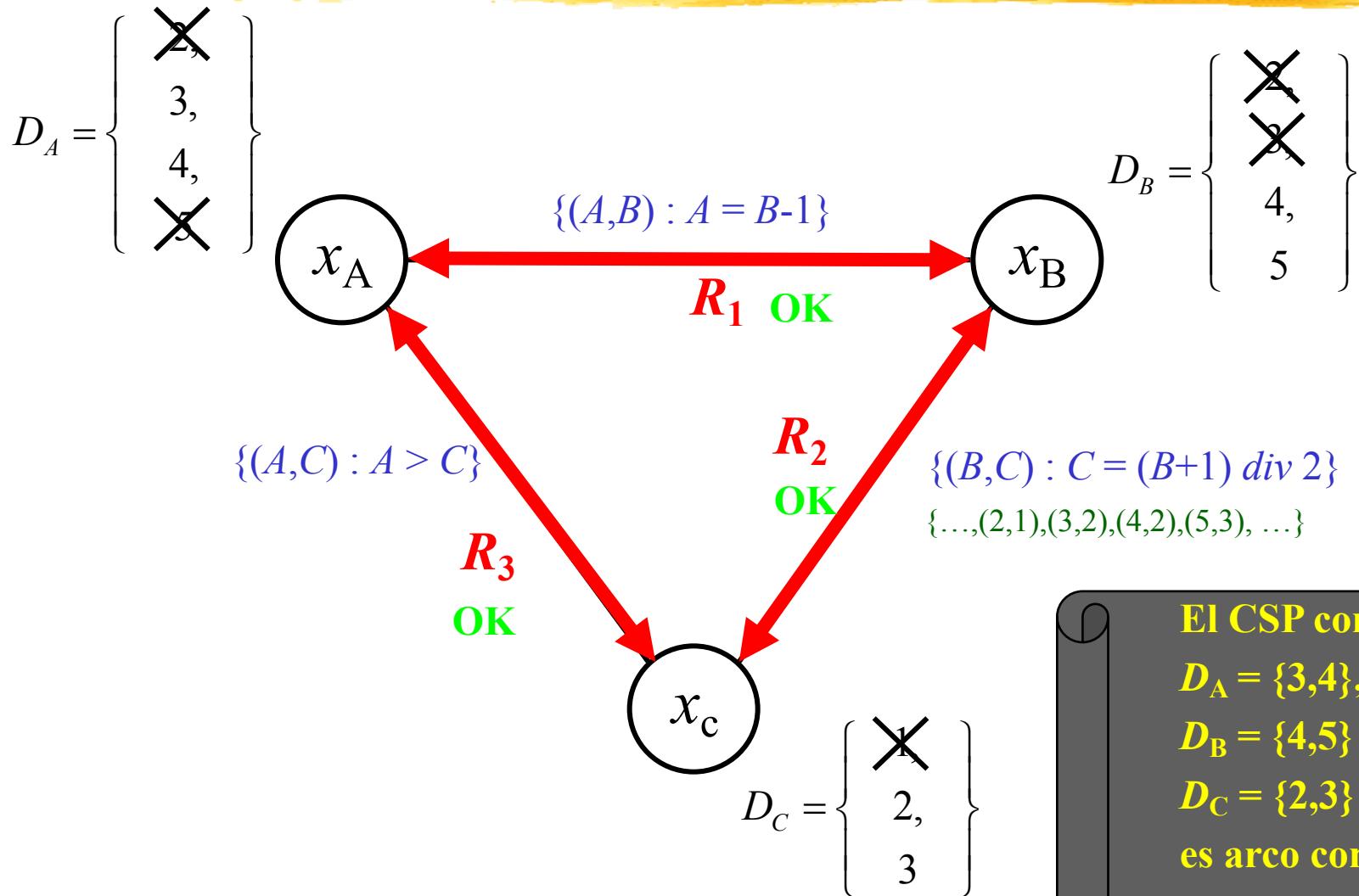
# Algoritmo de arco consistencia



# Algoritmo de arco consistencia



# Algoritmo de arco consistencia



# Algoritmo de arco consistencia

**Función** consistencia-de-arcos(CSP) **devuelve** CSP' % con dominios reducidos  
abierta  $\leftarrow R$  % inicialmente todas las restricciones binarias dirigidas

**Mientras**  $\neg$ vacio?(abierta) **hacer**

$R_y(x_i, x_j) \leftarrow$  primero(abierta)

**Si** borrar-valores-inconsistentes( $R_y(x_i, x_j)$ ) **entonces**

**Para cada**  $x_k \in$  vecinos( $x_i$ ) **hacer**

            abierta  $\leftarrow$  abierta  $\cup \{R_z(x_k, x_i)\}$

**Fin** {consistencia-de-arcos}

**Función** borrar-valores-inconsistentes( $R_y(x_i, x_j)$ ) **devuelve** si / no

borrado  $\leftarrow$  no % variable booleana – iniciar con false

**Para cada**  $v \in D_i$  **hacer** % borrar valores inconsistentes (sólo de  $D_i$ )

**Si**  $\neg \exists v' \in D_j$  tal que  $(v, v') \in R_y(x_i, x_j)$  **entonces**



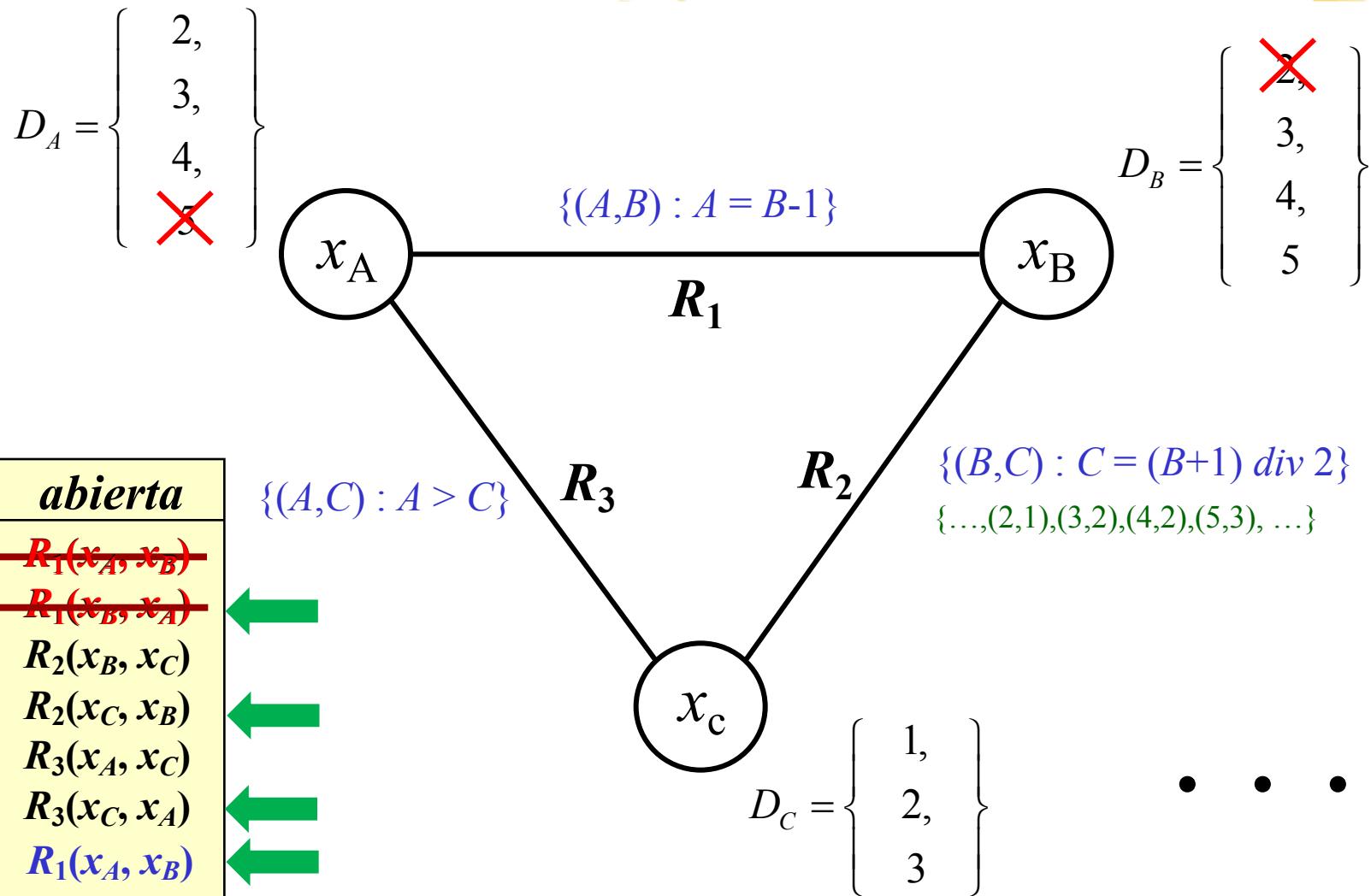
$D_i \leftarrow D_i \setminus \{v\}$

        borrado  $\leftarrow$  si

**Devolver** (borrado)

**Fin** {borrar-valores-inconsistentes}

# Algoritmo de arco consistencia



# Algoritmo de arco consistencia

## Resultado:

- El algoritmo de consistencia de arco reduce un CSP en un CSP' *equivalente* (es decir: con las mismas soluciones)
- Si al aplicar el algoritmo el dominio de una variable se queda vacío, el CSP es *inconsistente* (es decir: no tiene solución)

## Complejidad:

- Un grafo que representa el CSP binario tiene como mucho  $n \cdot (n-1)$  ( i.e.,  $O(n^2)$ ) arcos (dirigidos)
- Cada arco puede insertarse en *abierta* a lo sumo  $d$  veces
- El borrado de valores inconsistentes se realiza en  $O(d^2)$  pasos
- Por tanto, la complejidad en tiempo en el peor caso es  $O(n^2 \cdot d^3)$

## Análisis:

- No se puede garantizar que el algoritmo detecta *cualquier* CSP inconsistente (esto requeriría *k-consistencia fuerte*)
- La reducción en el tamaño de los dominios (y el ahorro correspondiente en el tiempo de ejecución del algoritmo de satisfacción de restricciones) suele compensar el coste adicional de aplicar el algoritmo de propagación de restricciones para alcanzar arco-consistencia

# Algoritmo MAC

**Algoritmo:** Mantenimiento de consistencia de arcos

- Inglés: *Maintaining Arc Consistency* (MAC)
- Intercalar satisfacción y propagación de restricciones
  - Realizar búsqueda con vuelta atrás cronológica
  - Después de aumentar una asignación, construir un CSP equivalente que sea arco-consistente
- MAC es de los algoritmos básicos más conocidos para CSPs

# Algoritmo MAC

Función **MAC**(CSP,  $\sigma$ ) devuelve  $\sigma'$  / *fallo*

```
Si  $|\sigma| = n$  entonces      % la asignación es completa (i.e. una solución)
    devolver( $\sigma$ )
 $x_i \leftarrow$  elegir-variable-no-asignada( $X, \sigma$ ) % orden de elección no afecta la completitud
dominio  $\leftarrow D_{x_i}$ 
Mientras dominio  $\neq \{\}$  hacer          % en el dominio quedan valores por probar
     $v \leftarrow$  elegir-valor(dominio)
     $\sigma' \leftarrow \sigma \cup \{x_i = v\}$            % generar nueva asignación parcial
    Si  $\sigma' \in R$  entonces          % la nueva asignación parcial es consistente
        CSP'  $\leftarrow$  consistencia-de-arcos( $\sigma'(CSP)$ )
        resultado  $\leftarrow$  MAC(CSP',  $\sigma'$ )    % completarla
        Si resultado  $\neq$  fallo entonces
            devolver(resultado)
        dominio  $\leftarrow$  dominio  $\setminus \{v\}$ 
    Fin {Mientras}
    devolver(fallo)
Fin {MAC}
```

# CSPs avanzados

## Temas avanzados:

- Algoritmos de satisfacción y propagación más sofisticados
- Heurísticas de selección de variables y/o valores más sofisticadas
- CSPs con dominios *infinitos* y *continuos*
- Estructuración (jerárquica) de CSPs
- Preferencias sobre soluciones (*optimización* de restricciones, COPs)
- Satisfacción /optimización *distribuida* de restricciones (DCSP/DCOP)
- *Relajación* de restricciones
- ...