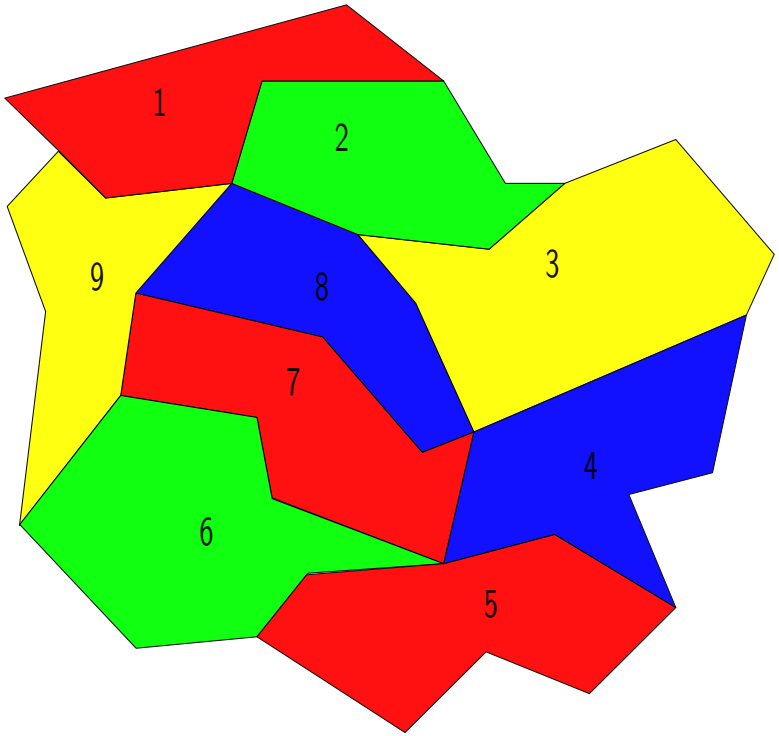
Satisfacción de Restricciones

- Componentes del estado:
  - Variables
  - Dominios (valores posibles para las variables)
  - Restricciones binarias entre las variables
- **Objetivo:** Encontrar un estado que satisface las restricciones (Asignación de valores a las variables, que satisfaga las restricciones)
- Ejemplos:
  - Colorear mapas, crucigramas, 8-reinas, sudoku, ...
  - Asignación/distribución/ubicación de recursos (distribución de tareas de fabricación, ubicación de gasolineras, antenas de telefonía, ...)

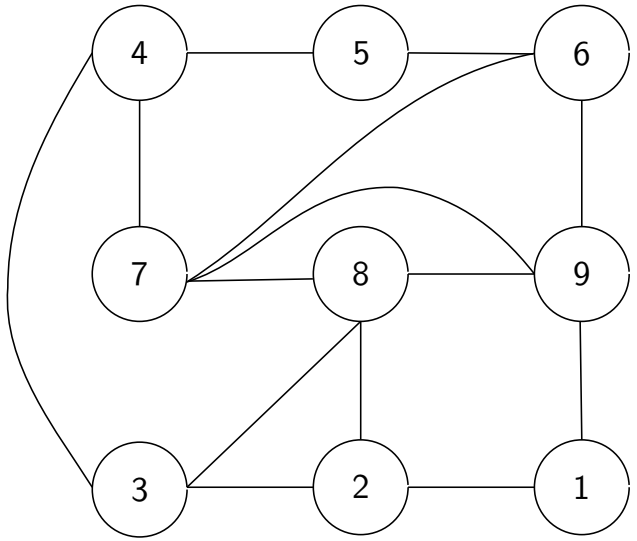
Representación

- Estado = Grafo de restricciones
  - Variables = etiquetas de nodos
  - Dominios = contenido de nodos
  - Restricciones = arcos dirigidos y etiquetados entre nodos
- Ejemplo: colorear un mapa



Dominios={Rojo,Verde,Azul,Amarillo}

Restricción := Desigualdad



Algoritmos

- **Generación y prueba:** enormemente ineficiente
- **Búsqueda ciega**
  - Búsqueda en profundidad con backtracking cronológico
- **Propagación de restricciones**
  - Antes de la búsqueda
  - Durante la búsqueda

Búsqueda en profundidad con backtracking cronológico

- Búsqueda en profundidad sobre las variables
- Asignar valor por estrategia exhaustiva
  - Comprobar restricciones tras cada posible asignación
  - Si no se satisfacen para ningún valor, backtracking sobre la última asignación válida
- La búsqueda se realiza en el espacio de soluciones parciales
- Backtracking cronológico: tipos de variables (pasadas, actual, futuras)

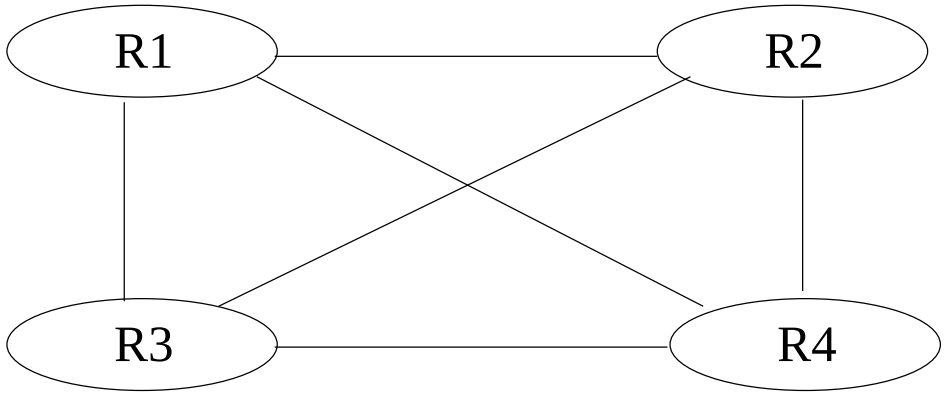
Algoritmo Backtracking Cronológico

Función: backtracking_cronologico(vfuturas, solucion)

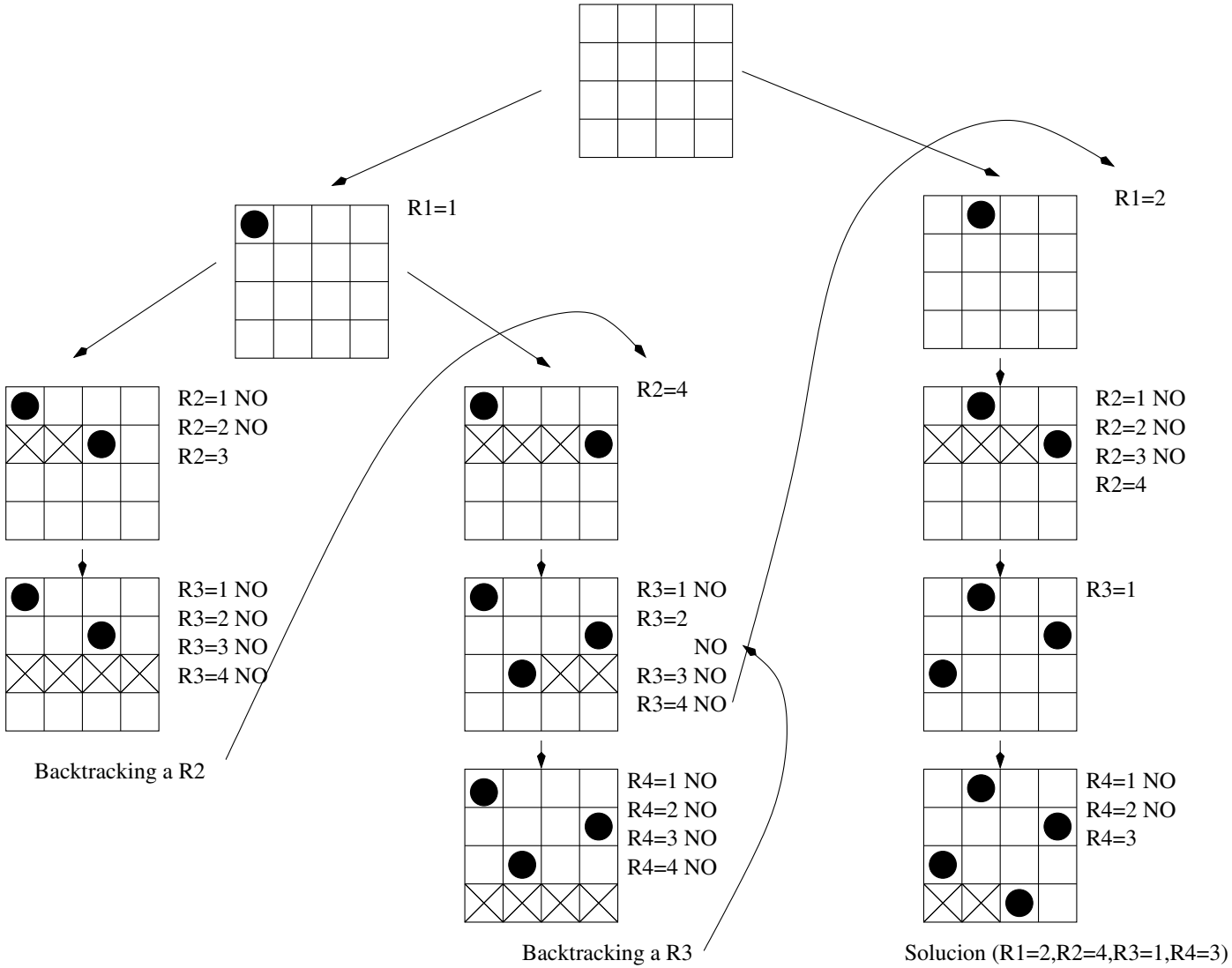
```
si vfuturas.es_vacio?() entonces
    retorna solucion
sino
    vactual ← vfuturas.primero()
    vfuturas.borrar_primero()
    para cada v ∈ vactual.valores() hacer
        vactual.asignar(v)
        solucion.anadir(vactual)
        si solucion.valida() entonces
            solucion ← backtracking_cronologico(vfuturas,solucion)
            si no solucion.es_fallo?() entonces
                retorna solucion
            sino
                solucion.borrar(vactual)
        sino
            solucion.borrar(vactual)
    retorna solucion.fallo()
```

Ejemplo: 4-reinas

- Colocar 4 reinas, 1 en cada fila de un tablero 4x4, sin que se maten
  - Variables: R_1, ... , R_4 (reinas)
  - Dominios: [1 .. 4] para cada R_i (columna)
  - Restricciones: R_i no-mata R_j
- Grafo de restricciones:

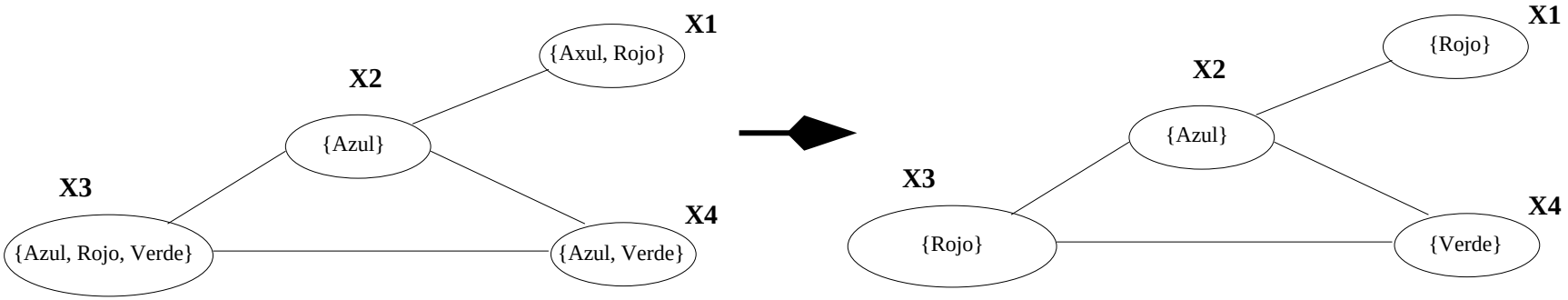


4-reinas mediante backtracking cronológico



Propagación de restricciones

Un conjunto de restricciones puede inducir otras que estaban implícitas. La propagación de restricciones es el proceso de hacerlas explícitas



El papel de la PR es disminuir el espacio de búsqueda. Debemos realizar la propagación:

1. preproceso (eliminar zonas del espacio donde no hay soluciones)
2. durante el proceso: podar el espacio a medida que la búsqueda progresa (Forward Checking)

Propagación de restricciones

Cada ciclo tiene dos partes:

1. Se propagan las restricciones
   Se podrían utilizar de reglas de inferencia.
   Tener en cuenta que las restricciones no tienen por qué ser independientes (Muchas restricciones implican a varias variables, una variable participa en muchas restricciones)
2. Se analiza el resultado:
   1. Solución encontrada
   2. Solución imposible
   3. Seguir buscando: proceso heurístico de búsqueda

Propiedades sobre grafos de restricciones

- Se pueden definir propiedades sobre los grafos de restricciones que permiten reducir el espacio de búsqueda
  - k-consistencia: Poda de valores que no sean posibles para un grupo de k variables
  - Arco consistencia (2-consistencia): Eliminamos valores imposibles para parejas de variables
  - Camino consistencia (3-consistencia): Eliminamos valores imposibles para ternas de variables
  - ...
- Comenzar con un grafo k-consistente (2, 3, ...) reduce el número de backtrackings

Preproceso de arco-consistencia

- Un PSR es arco-consistente si para cada par de variables (X_i, X_j) y para cualquier valor v_k de D_i existe un valor v_l de D_j tal que se satisfacen las restricciones. Es decir, se busca que los valores posibles de X_i sean consistentes con la restricción asociada al arco.
- Lo que realmente pretendemos es que todas las variables sean arco consistentes para todos los arcos que inciden en ellas. Es decir, que los dominios actuales de cada variable sean consistentes con todas las restricciones.

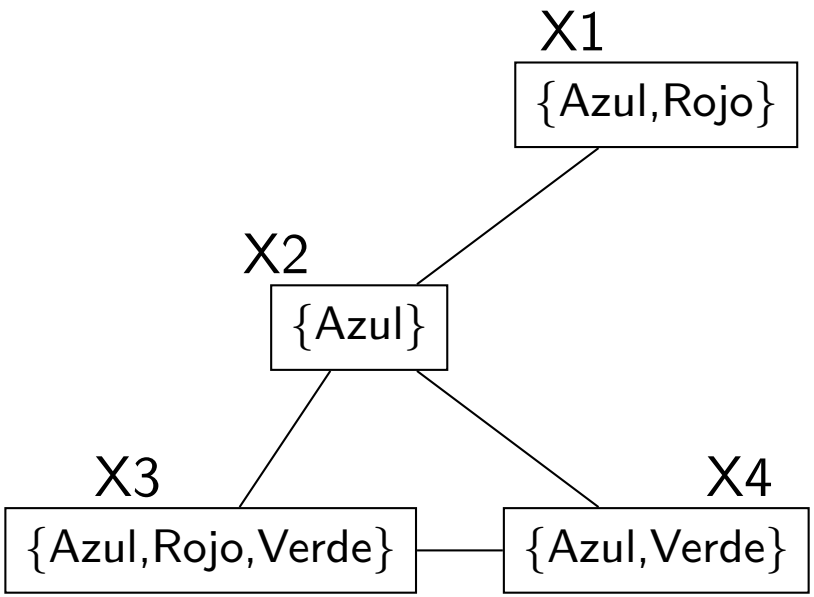
Algoritmo de arco-consistencia

Si un PSR no es arco-consistente se le puede convertir mediante el siguiente algoritmo:

Algoritmo: Arco consistencia

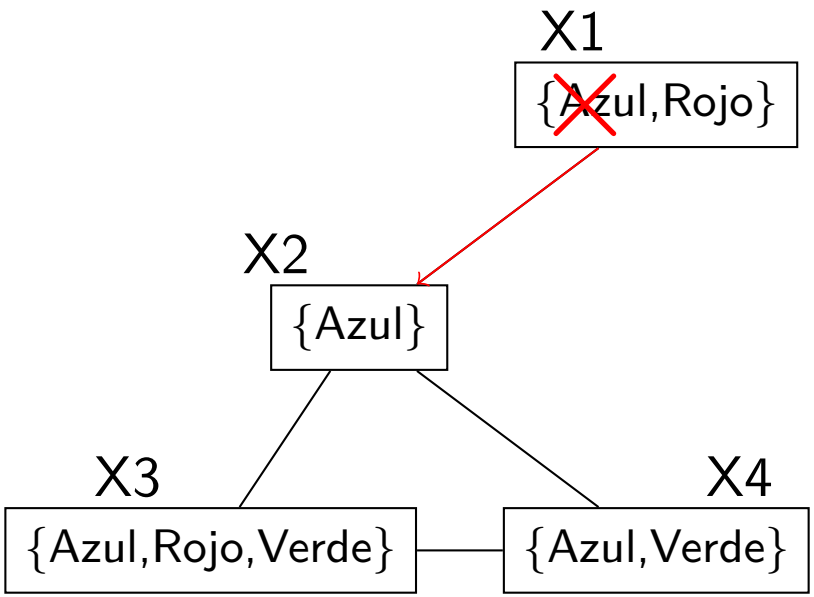
R $\leftarrow$ conjunto de arcos del problema `/* ambos sentidos */`
mientras *se modifiquen los dominios de las variables* **hacer**
  r $\leftarrow$ extraer_arco(R)
  `/*` r_i `es la variable del origen del arco */`
  `/*` r_j `es la variable del destino del arco */`
  para cada $v \in$ *en el dominio de* r_i **hacer**
    si *v no tiene ningún valor en el dominio de* r_j *que cumpla r* **entonces**
      borrar v del dominio de r_i
      añadir todos los arcos que tengan como destino r_i menos el $(r_j \rightarrow r_i)$
    fin
  fin
fin

Ejemplo arco-consistencia



Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

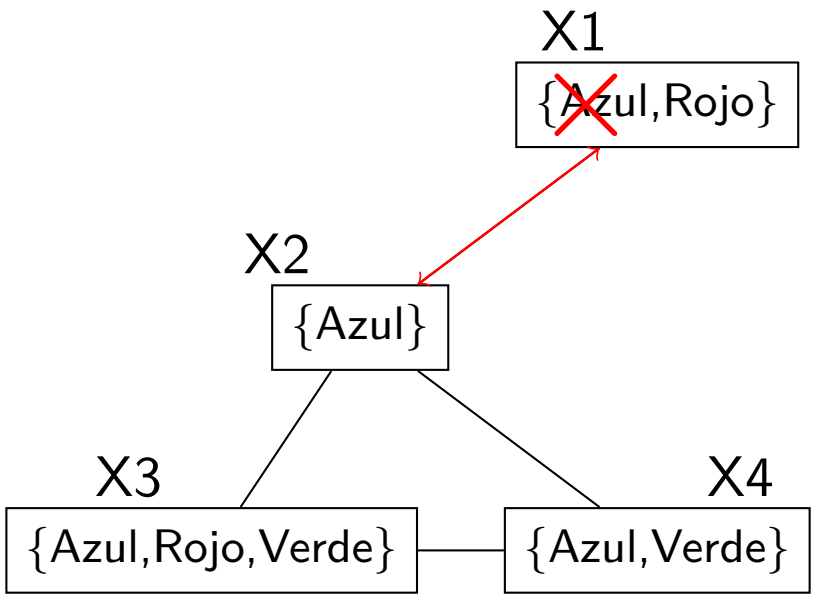
Ejemplo arco-consistencia



1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1

Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

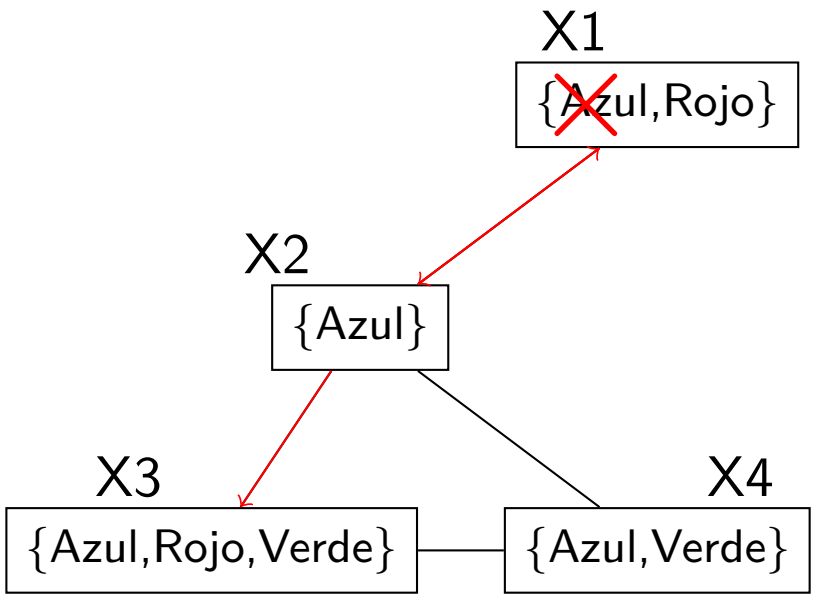
Ejemplo arco-consistencia



1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente

Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

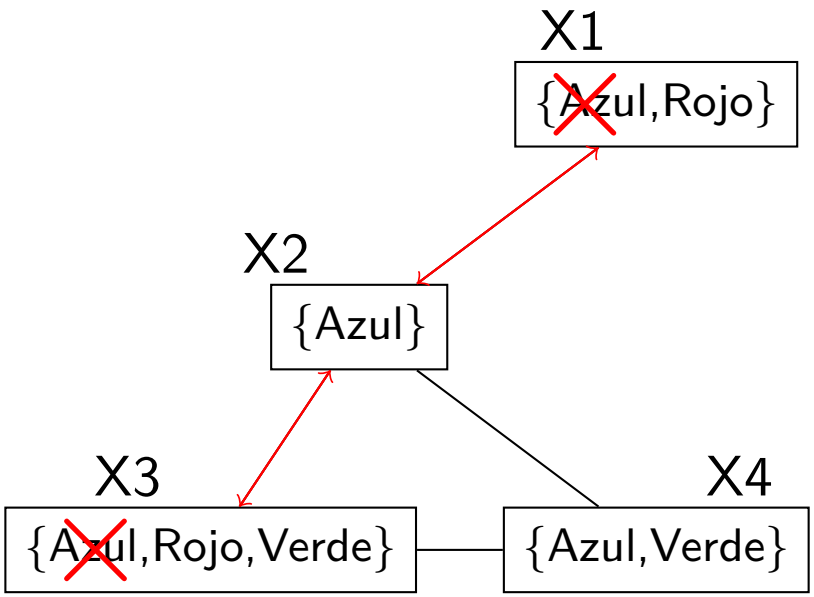
Ejemplo arco-consistencia



1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente
3. $X_2 - X_3 \rightarrow$ Todo consistente

Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2),
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

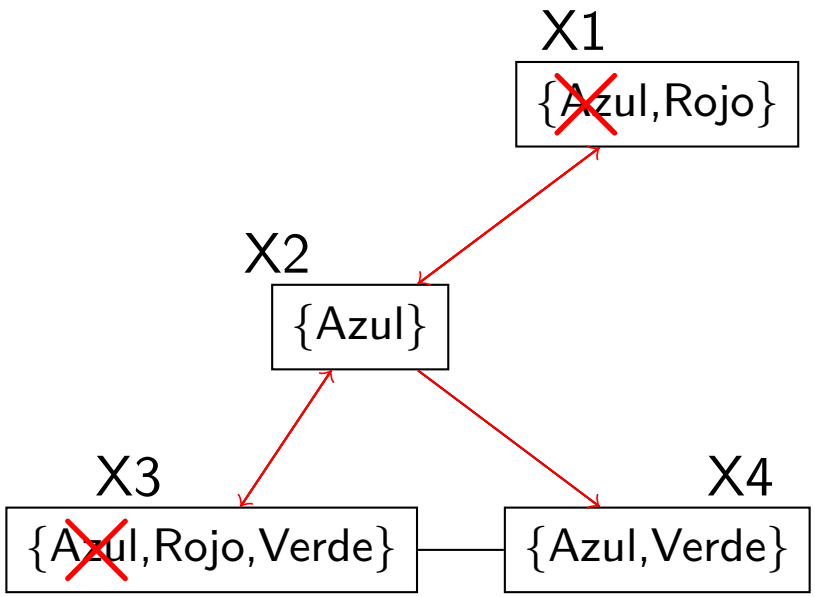
Ejemplo arco-consistencia



1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente
3. $X_2 - X_3 \rightarrow$ Todo consistente
4. $X_3 - X_2 \rightarrow$ Quitar Azul de X_3, Tendríamos que añadir $X_4 - X_3$ pero ya está

Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

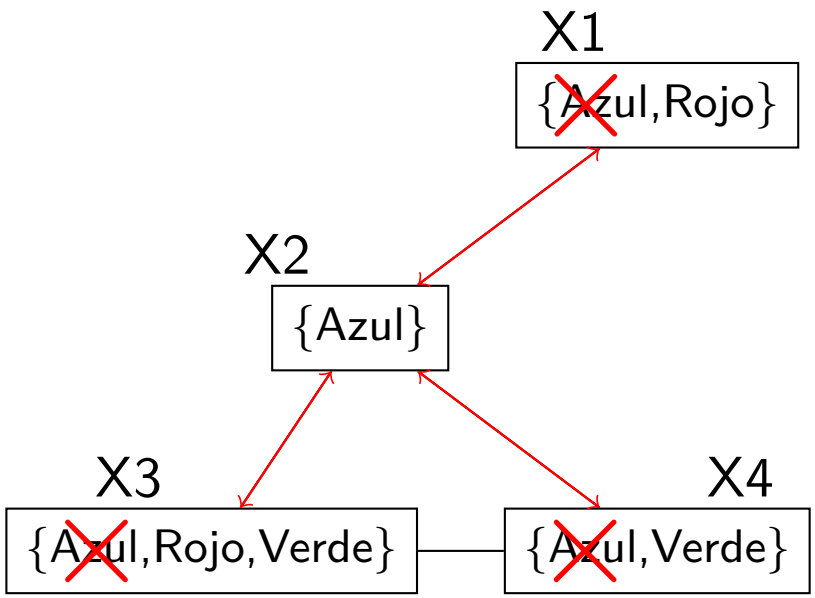
Ejemplo arco-consistencia



1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente
3. $X_2 - X_3 \rightarrow$ Todo consistente
4. $X_3 - X_2 \rightarrow$ Quitar Azul de X_3, Tendríamos que añadir $X_4 - X_3$ pero ya está
5. $X_2 - X_4 \rightarrow$ Todo consistente

Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

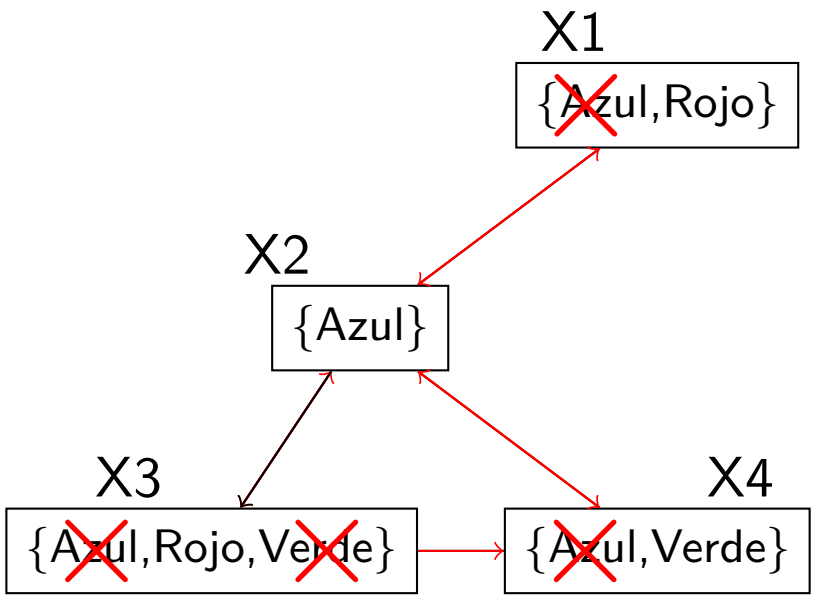
Ejemplo arco-consistencia



1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente
3. $X_2 - X_3 \rightarrow$ Todo consistente
4. $X_3 - X_2 \rightarrow$ Quitar Azul de X_3, Tendríamos que añadir $X_4 - X_3$ pero ya está
5. $X_2 - X_4 \rightarrow$ Todo consistente
6. $X_4 - X_2 \rightarrow$ Quitar Azul de X_4, Tendríamos que añadir $X_3 - X_4$ pero ya está

Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

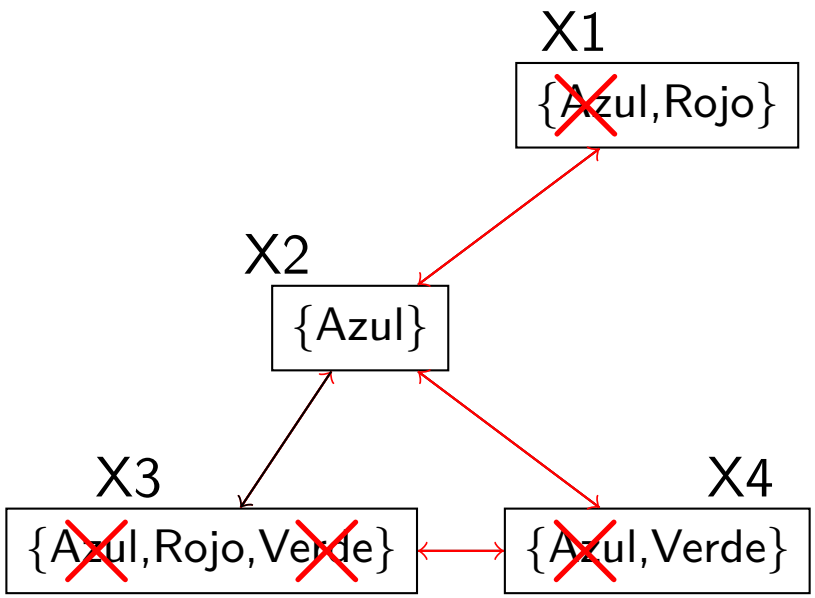
Ejemplo arco-consistencia



Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente
3. $X_2 - X_3 \rightarrow$ Todo consistente
4. $X_3 - X_2 \rightarrow$ Quitar Azul de X_3, Tendríamos que añadir $X_4 - X_3$ pero ya está
5. $X_2 - X_4 \rightarrow$ Todo consistente
6. $X_4 - X_2 \rightarrow$ Quitar Azul de X_4, Tendríamos que añadir $X_3 - X_4$ pero ya está
7. $X_3 - X_4 \rightarrow$ Quitar Verde de X_3, Añadimos $X_2 - X_3$

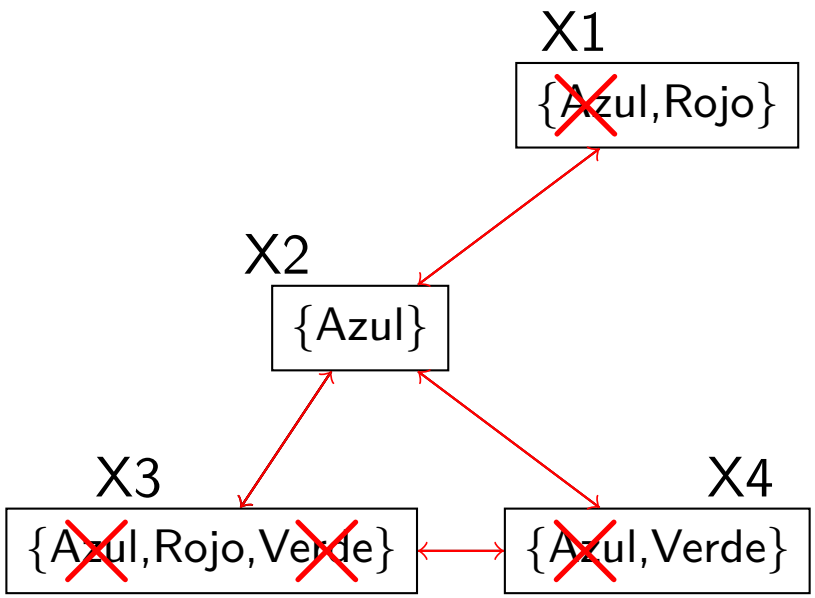
Ejemplo arco-consistencia



Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente
3. $X_2 - X_3 \rightarrow$ Todo consistente
4. $X_3 - X_2 \rightarrow$ Quitar Azul de X_3, Tendríamos que añadir $X_4 - X_3$ pero ya está
5. $X_2 - X_4 \rightarrow$ Todo consistente
6. $X_4 - X_2 \rightarrow$ Quitar Azul de X_4, Tendríamos que añadir $X_3 - X_4$ pero ya está
7. $X_3 - X_4 \rightarrow$ Quitar Verde de X_3, Añadimos $X_2 - X_3$
8. $X_4 - X_3 \rightarrow$ Todo consistente

Ejemplo arco-consistencia



Lista de arcos inicial:
(X1,X2), (X2,X1), (X2,X3), (X3,X2), (X2,X4), (X4,X2), (X3,X4), (X4,X3)

1. $X_1 - X_2 \rightarrow$ Quitar Azul de X_1
2. $X_2 - X_1 \rightarrow$ Todo consistente
3. $X_2 - X_3 \rightarrow$ Todo consistente
4. $X_3 - X_2 \rightarrow$ Quitar Azul de X_3, Tendríamos que añadir $X_4 - X_3$ pero ya está
5. $X_2 - X_4 \rightarrow$ Todo consistente
6. $X_4 - X_2 \rightarrow$ Quitar Azul de X_4, Tendríamos que añadir $X_3 - X_4$ pero ya está
7. $X_3 - X_4 \rightarrow$ Quitar Verde de X_3, Añadimos $X_2 - X_3$
8. $X_4 - X_3 \rightarrow$ Todo consistente
9. $X_2 - X_3 \rightarrow$ Todo consistente

Propagación durante la búsqueda (forward checking)

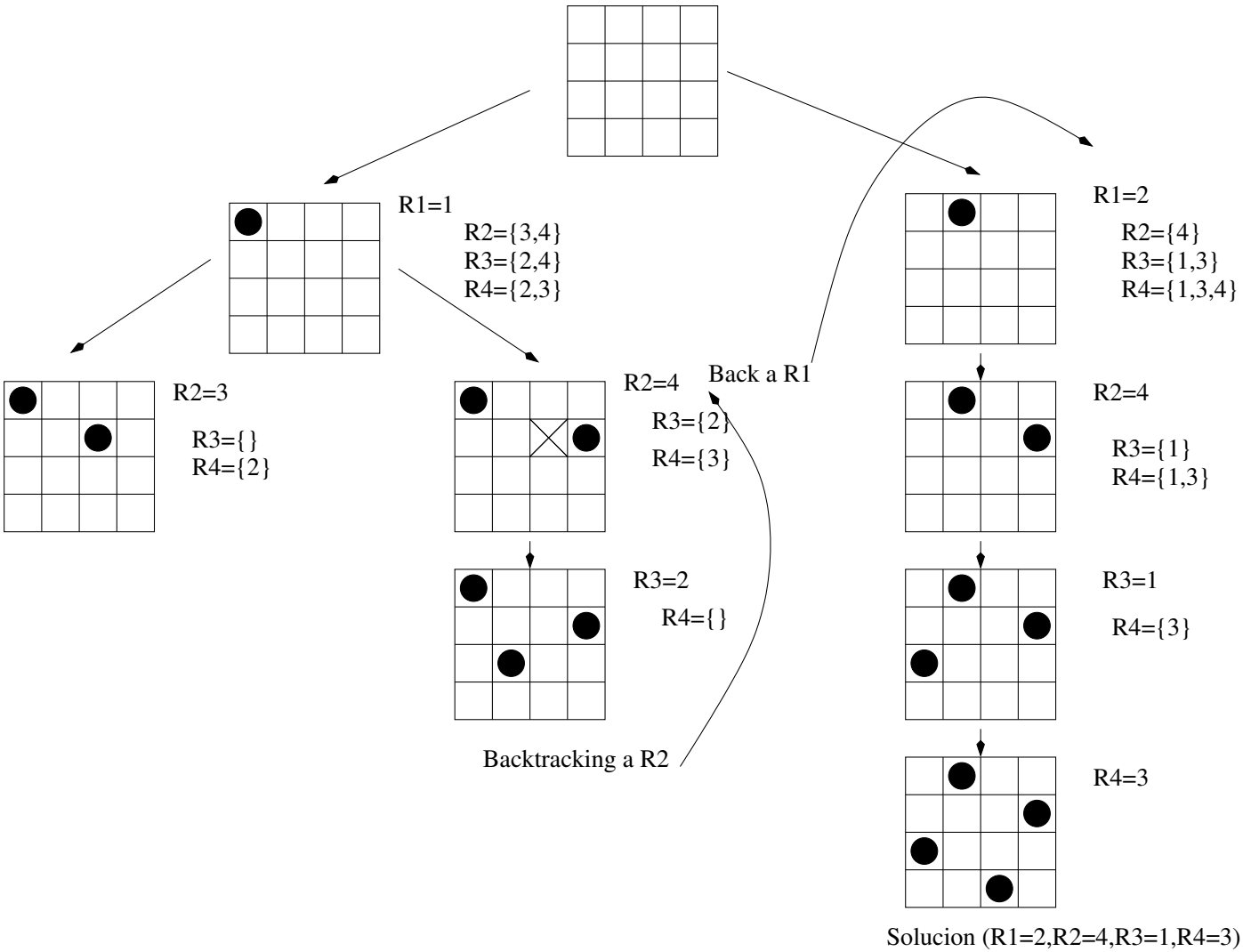
- Modificación del algoritmo de búsqueda en profundidad con backtracking cronológico (introducimos la propagación de restricciones después de cada asignación)
- **Anticipación:** detectar cuanto antes caminos sin solución y podarlos.
  - Asignar un valor y consultar las restricciones sobre las variables futuras con arco desde la actual
  - Se eliminan valores no compatibles de los dominios correspondientes a dichas variables futuras
- Equivale a hacer arco-consistente la variable actual con las futuras en cada paso
- La eficiencia dependerá del problema (incrementamos el coste de cada iteración)

Algoritmo Forward Checking

Función: forward checking (vfuturas, solucion)

```
si vfuturas.es_vacio?() entonces
    retorna solucion
sino
    vactual ← vfuturas.primero()
    vfuturas.borrar_primero()
    para cada v ∈ vactual.valores() hacer
        vactual.asignar(v)
        solucion.anadir(vactual)
        vfuturas.propagar_restricciones(vactual) /* forward checking */
        si no vfuturas.algun_dominio_vacio?() entonces
            solucion ← forward_checking(vfuturas,solucion)
            si no solucion.es_fallo?() entonces
                retorna solucion
            sino
                solucion.borrar(vactual)
        sino
            solucion.borrar(vactual)
    retorna solucion.fallo()
```

Ejemplo: 4-reinas mediante forward checking



Heurísticas adicionales

La búsqueda con backtracking puede mejorarse

- Comprobando consistencias mas restrictivas (con mayor coste)
  - Haciendo arco consistente todo el problema a cada paso (Algoritmo MAC)
- Escogiendo el orden de prueba de las variables
  - ¿Cuando?
    - Antes de la búsqueda (orden fijo)
    - Durante la búsqueda (orden variable)
  - ¿Que orden?
    - Primero variables con mas restricciones
    - Primero variables con menos valores
  - La reordenación de variables puede reducir el tiempo de búsqueda varios ordenes de magnitud en ciertos problemas