

# Tema 4:

# Programación de Restricciones

Sistemas Inteligentes  
3 Grado de Ingeniería Informática  
Esp. Computación

# Programación de restricciones

- Definición
- Restricciones
- Resolución de problemas
  - Modelar
  - Procesar
    - Reducción de problemas: Consistencia
    - Algoritmos de búsqueda
- Heurísticas

# Definición

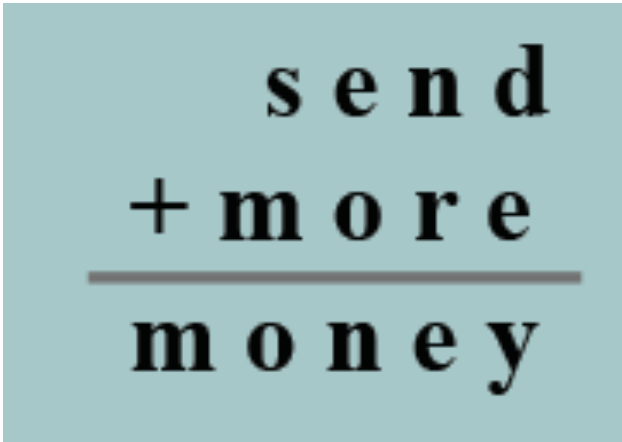
- Llamamos problema de ***satisfacción de restricciones*** (CSP: Constraint Solving Problem) a un triple  $(V,D,R)$  formado por:
  - Un **conjunto de variables**  $V = \{X_1, \dots, X_n\}$ .
  - Para cada variable de  $V$  un conjunto de posibles valores  $D_i$ , que llamaremos ***dominio de  $X_i$*** .
  - Un **conjunto de restricciones**, normalmente binarias,  $C_{ij}(X_i, X_j)$  que determinan los valores que las variables pueden tomar simultáneamente.

# Ejemplo

send  
+ more  
-----  
money

# Ejemplo

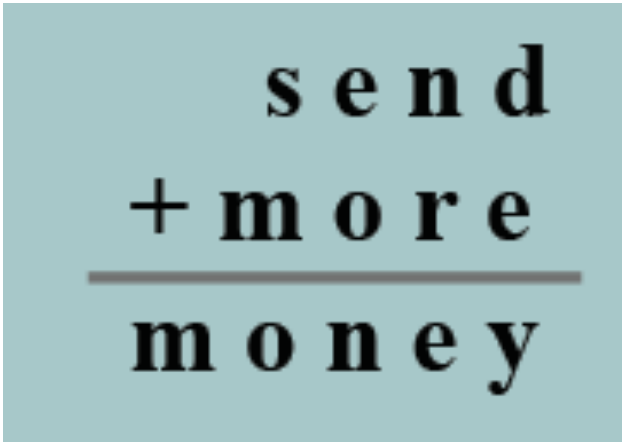
- Resolver:



s e n d  
+ m o r e  
-----  
m o n e y

# Ejemplo

- Resolver:



send  
+ more  
—  
money

- Variables: s,e,n,d,m,o,r,y

# Ejemplo

- Resolver:

$$\begin{array}{r} \text{ s e n d} \\ + \text{ m o r e} \\ \hline \text{ m o n e y} \end{array}$$

- Variables: s,e,n,d,m,o,r,y
- Dominio: s,e,n,d,m,o,r,y  $\in \{0,1,2,3,\dots,9\}$

# Ejemplo

- Resolver:

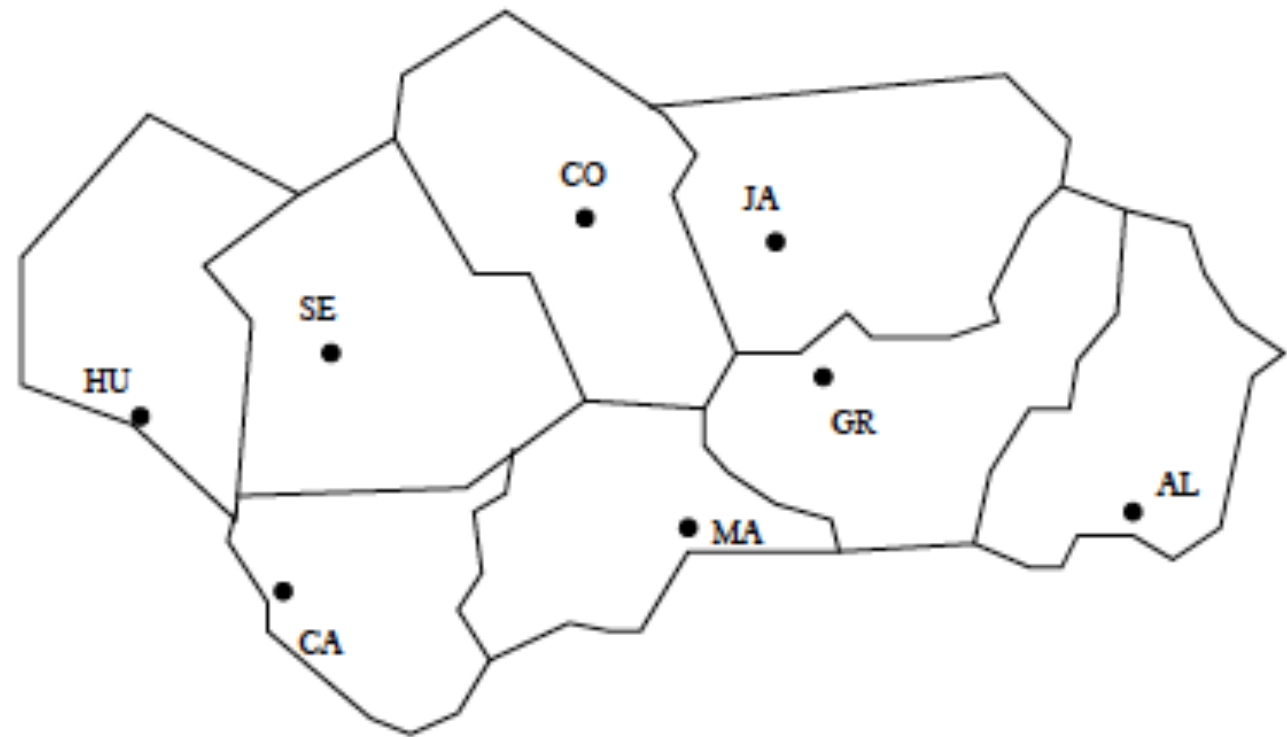
$$\begin{array}{r} \text{s e n d} \\ + \text{m o r e} \\ \hline \text{m o n e y} \end{array}$$

- Variables: s,e,n,d,m,o,r,y
- Dominio: s,e,n,d,m,o,r,y  $\in \{0,1,2,3,\dots,9\}$
- Restricciones:
  - $10^3(s+m) + 10^2(e+o) + \dots = 10^4m + 10^3o + 10^2n \dots$



# Ejemplo

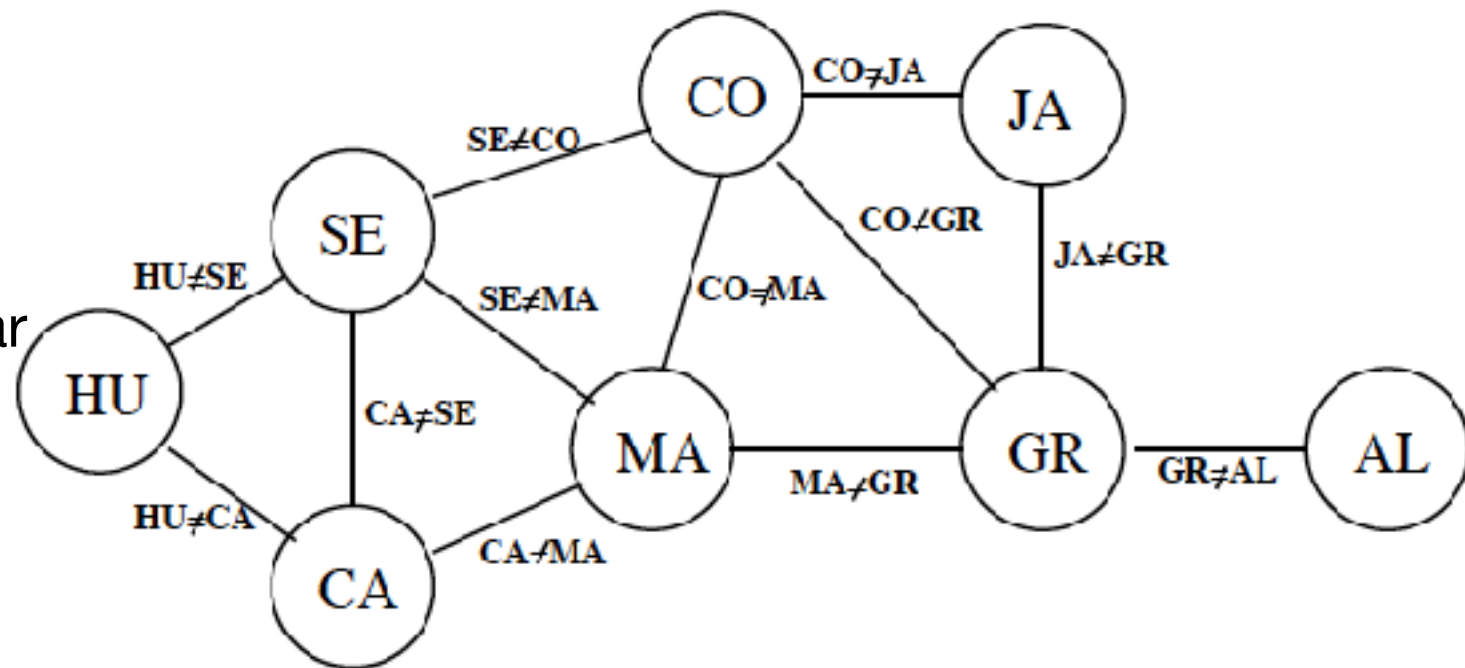
- Usando tres colores (rojo, azul, verde) colorear el mapa de Andalucía.
- Variables:  
 $\{HU, SE, JA, CO, CA, GR, AL, MA\}$
- Dominios: {rojo, azul, verde}
- Restricciones:  $P \neq Q$ , para cada par de provincias vecinas P y Q
- Problema con dominios finitos y restricciones binarias (de obligación).



# Ejemplo

- Usando tres colores (rojo, azul, verde) colorear el mapa de Andalucía.

- Variables:  
 $\{HU, SE, JA, CO, CA, GR, AL, MA\}$
- Dominios: {rojo, azul, verde}
- Restricciones:  $P \neq Q$ , para cada par de provincias vecinas P y Q
- Problema con dominios finitos y restricciones binarias (de obligación).



# Tipos de Problemas

- Clasificación según el tipo de restricciones:
  - Restricciones de obligación (hard constraints).
  - Restricciones de preferencia (soft constraints).
- Clasificación según los dominios:
  - Dominios discretos (finitos o infinitos).
  - Dominios continuos.
- Clasificación según el número de variables implicadas en las restricciones:
  - Restricciones binarias.
  - Restricciones múltiples.

# Definición

- El **objetivo** es **encontrar un valor** para cada **variable** de manera que se satisfagan todas las restricciones del problema.
- Las estrategias de búsqueda de soluciones tratan de encontrar las tuplas de valores  $(v_1, \dots, v_n)$  de las variables  $X_1, \dots, X_n$  que satisfacen las restricciones.

# Variables

- Un CSP **discreto** es aquel en el que todas las variables son discretas, es decir, toman valores en dominios finitos.
- Un CSP **continuo** es un CSP en el que todas las variables son continuas, es decir, tienen dominios continuos.
- Un CSP **mixto** consta de variables continuas y discretas.
- Un CSP **binario** es aquel en el que todas las restricciones tienen a los sumo dos variables respectivamente.
- Un CSP no binario o **n-ario** es aquel en el que las restricciones tienen cualquier cualquier número de variables.

# Restricciones

- Discretas: las variables participantes están acotadas en dominios discretos.
- Continuas: las variables participantes están acotadas en dominios continuos.
- Binarias: son restricciones en las que sólo participan dos variables.
- N-arias: son restricciones en las que sólo participan cualquier número de variables ( $n > 2$ ).
- Fuertes (hard): son restricciones cuya satisfabilidad es imprescindible.
- Débiles (soft): son restricciones cuya satisfabilidad no es imprescindible.
- Difusas (fuzzy): son restricciones definidas sobre niveles de preferencia.
- **Disyuntivas**: son restricciones compuestas por un conjunto disjunto de restricciones.

# Restricciones

- Cada restricción limita el conjunto de asignaciones para las variables implicadas
- Pueden darse de dos formas:
  - Explícita (mediante tablas)
  - Implícita
    - $X1 > x2$

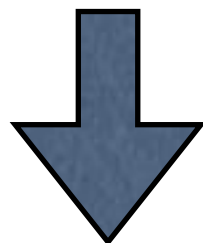
# Aridad de las restricciones

- Restricción unitaria: Tiene una sola variable afectada.
- La restricción puede usarse para excluir un valores del dominio de definición, por ejemplo,  $X > 5$ .
- Restricción binaria: Tiene dos variables afectadas.
- Una restricción binaria entre variables de dominios de tamaño  $m$  y  $n$ , por ejemplo,  $e \neq n$



# Resolución

MODELADO  
CSP



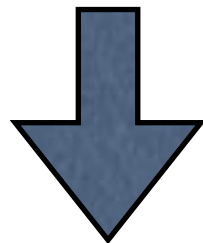
RESOLUCIÓN  
CSP

Variables  
Dominios  
Restricciones

Técnicas  
Resolución

# Resolución

MODELADO  
CSP



RESOLUCIÓN  
CSP

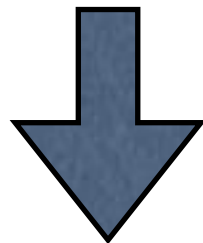
Variables  
Dominios  
Restricciones

Expresividad

Técnicas  
Resolución

# Resolución

MODELADO  
CSP



RESOLUCIÓN  
CSP

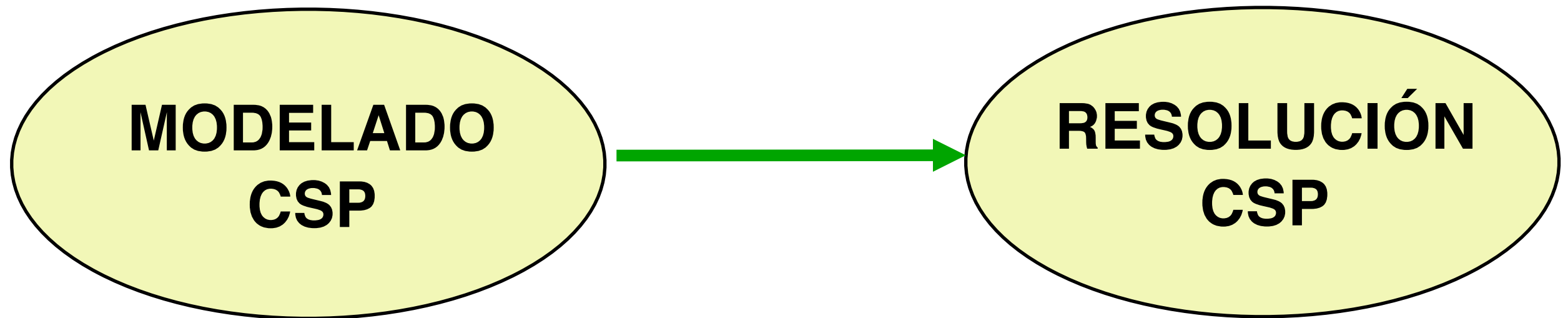
Variables  
Dominios  
Restricciones

Expresividad

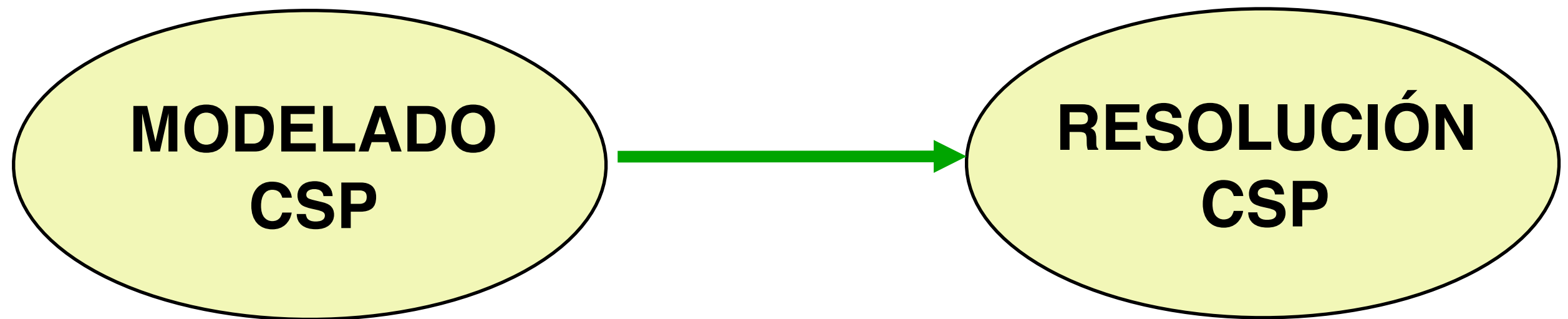
Técnicas  
Resolución

Eficiencia

# Ejemplo



# Ejemplo

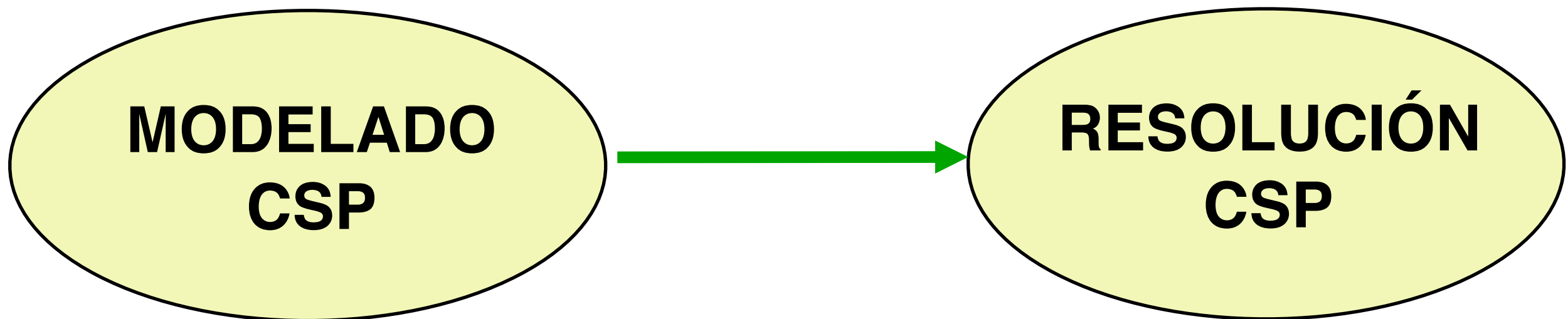


**s e n d**  
**+ m o r e**  

---

**m o n e y**

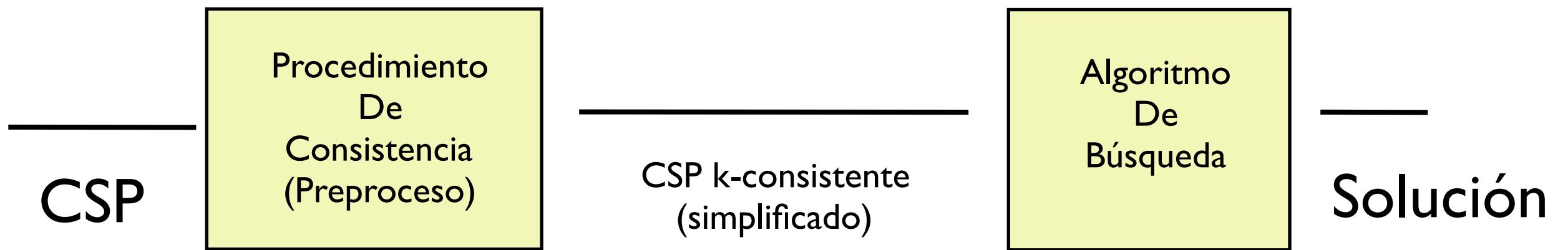
# Ejemplo



$$\begin{array}{r} \text{send} \\ + \text{more} \\ \hline \text{money} \end{array} \Rightarrow \begin{array}{r} 9567 \\ + 1085 \\ \hline 10652 \end{array}$$

# Resolución del CSP

- **Procesar** el problema de satisfacción de restricciones resultante.
  - Técnicas de consistencia
  - Algoritmos de búsqueda
- **CSPs Distribuidos.** La introducción de los agentes y los sistemas multiagentes en el campo de la Inteligencia Artificial han generado una nueva manera de resolver los CSPs de forma distribuida.





# Algoritmos de Búsqueda

# Algoritmos de Búsqueda

- Algoritmos para resolver problemas de satisfacción de restricciones
  - Espacio de estados
  - Algoritmo de genera y comprueba
  - Algoritmo de backtracking simple
  - Algoritmo de backtracking con chequeo previo

# Espacio de estados

- Estados:
  - Asignaciones parciales (y consistentes con las restricciones) de valores a variables:  $\{X_{i1} = v_{i1}, \dots, X_{ik} = v_{ik}\}$
  - Estado inicial:  $\{\}$
  - Estados finales: asignaciones completas
- Operadores:
  - Asignar un valor (de su dominio) a una variable no asignada en el estado
  - Aplicabilidad: la asignación resultante tiene que ser consistente
- Soluciones mediante búsqueda en profundidad
  - Puesto que las soluciones están a una profundidad fija (igual al número de variables), profundidad es preferible a anchura

# Simplificaciones

- El orden en el que se aplican los operadores es irrelevante para la solución final (conmutatividad)
- Por tanto, los operadores pueden reducirse a considerar las posibles asignaciones a una única variable no asignada
- El camino hacia la solución no es importante
- No necesitamos una representación explícita para los operadores
- No es posible repetir un estado durante la búsqueda:
- No es necesario realizar comprobaciones para evitar repeticiones
- Es decir, no es necesario tener una lista CERRADOS con las asignaciones parciales ya analizadas

# Búsqueda: implementación

- Estados: representados como listas de asociación
- Usaremos una función que elige la nueva variable a asignar (de entre las no asignadas) y otra función que va a ordenar los valores del dominio de la variable seleccionada:
  - SELECCIONA-VARIABLE(ESTADO)
  - ORDENA-VALORES(DOMINIO)

# Búsqueda: implementación

**FUNCION PSR-BUSQUEDA-EN-PROFUNDIDAD()**

**ABIERTOS = {AsigVacía}**

**Mientras que ABIERTOS no esté vacía,**

**ACTUAL = primero(ABIERTOS)**

**Si ACTUAL es una asignación completa**

**devolver ACTUAL y terminar.**

**en caso contrario,**

**NUEVOS-SUCESORES = SUCESORES(ACTUAL)**

**ABIERTOS = NUEVOS-SUCESORES + ABIERTOS**

**3. Devolver FALLO.**

# Búsqueda: implementación

- Función que calcula los sucesores:

```
FUNCION SUCESTORES( ) :
```

```
    VARIABLE = SELECCIONA-VARIABLE(ESTADO)
```

```
    DOMINIO-ORD = ORDENA-VALORES(PSR-VAR-DOMINIO(VARIABLE))
```

```
    SUCESTORES = [ ]
```

```
    Para cada VALOR en DOMINIO-ORD {
```

```
        NUEVO-ESTADO = ESTADO + {VARIABLE=VALOR}
```

```
        Si NUEVO-ESTADO es consistente con *RESTRICCIONES*,
```

```
            añadir NUEVO-ESTADO a SUCESTORES
```

```
    Devolver SUCESTORES
```

# Genera y Comprueba

- Se asigna un valor del dominio permitido a cada una de las variables de la tupla que se va a evaluar, y después se comprueba que forman una solución, es decir, si verifican todas las restricciones. Si no las verifica entonces se desecha esa tupla y genera la siguiente.
- El proceso se repite hasta encontrar una solución, o hasta que se hayan generado y comprobado todos los casos posibles.
- Problema: Este algoritmo evalúa cada vez las restricciones entre todas las variables.



# Algoritmo Backtracking

- El algoritmo de búsqueda en profundidad anterior se suele llamar algoritmo de backtracking, aunque usualmente, se presenta de manera recursiva:

```
FUNCION PSR-BACKTRACKING()
```

```
    Devolver PSR-BACKTRACKING-REC({})
```

```
FUNCION PSR-BACKTRACKING-REC(ESTADO)
```

```
    Si ESTADO es una asignación completa, devolver ESTADO y terminar
```

```
    VARIABLE = SELECCIONA-VARIABLE(ESTADO)
```

```
    DOMINIO-ORD = ORDENA-VALORES(CSP-VAR-DOMINIO(VARIABLE))
```

```
    Para cada VALOR en DOMINIO-ORD,
```

```
        NUEVO-ESTADO = ESTADO + {VARIABLE=VALOR}
```

```
        Si NUEVO-ESTADO es consistente con *RESTRICCIONES*:
```

```
            RESULTADO = PSR-BACKTRACKING-REC(NUEVO-ESTADO)
```

```
            Si RESULTADO no es FALLO, devolver RESULTADO y terminar
```

```
    Devolver FALLO
```

# Algoritmo de backtracking simple

- Se asigna un valor del dominio permitido a la siguiente variable a evaluar, y se comprueba si forma parte de la solución parcial
- Si no verifica las restricciones entonces se desecha ese valor y se toma el siguiente valor del dominio permitido para la variable.

# Algoritmo de backtracking simple

- Si se intentan todos los valores de la variable y ninguno forma parte de la solución, entonces se pasa a eliminar el valor de la variable anterior evaluada y se toma el siguiente valor para esa variable.
- El proceso se repite hasta encontrar una solución, o hasta que se hayan probado todos los casos posibles.
- Este algoritmo solo computa las restricciones entre la nueva variable a evaluar y las variables computadas previamente.

# Backtracking con chequeo previo

- Se asigna un valor de su dominio a la nueva variable a evaluar, y se comprueba no solo si forma parte de la solución parcial sino que además se mira si algún valor de una futura asignación tiene conflicto con este valor y si es así se elimina temporalmente del dominio de esa futura variable.
- La ventaja de este método es que si el dominio de una futura variable llega a ser vacío, esto significa que la solución parcial es inconsistente, con lo cual se prueba con otro valor de la variable activa o se vuelve atrás y el dominio de las futuras variables restaurado.

# Backtracking con chequeo previo

- Con el backtracking simple no se habrá detectado la inconsistencia hasta que no se hubieran evaluado todas las futuras variables.
- En el algoritmo de backtracking con chequeo previo las ramas de los árboles que van a dar inconsistencia son podadas con anterioridad.

# Heurísticas

- En principio, Backtracking realiza una búsqueda ciega, ineficiente en la práctica
- Sin embargo, es posible dotar al algoritmo de cierta heurística que mejora considerablemente su rendimiento
- Estas heurísticas son de propósito general
  - Independientes del problema
  - Sacan partido de la estructura especial de los CSP
- Posibles mejoras heurísticas:
  - Selección de nueva variable a asignar
  - Orden de asignación de valores a la variable elegida
  - Propagación de información a través de las restricciones
  - Vuelta atrás “inteligente” en caso de fallo

# Mejoras

- Mejoras iterativas para un CSP
- Planteamiento como un problema de búsqueda local:
  - Estados: asignaciones completas (consistentes o inconsistentes)
  - Estado inicial escogido aleatoriamente
  - Estados finales: soluciones al CSP
  - Generación de sucesor: elegir una variable y cambiar el valor que tiene asignado (usando cierta heurística y aleatoriedad)

# Preproceso

## Técnicas híbridas y de preproceso

- Técnicas heurísticas de preproceso:
  - Ordenación de variables
  - Ordenación de valores
  - Ordenación de restricciones
- Algoritmos híbridos:
  - “forward checking”
  - Técnicas de consistencia



# Ordenación de Variables

# Ordenación de Variables

- La idea de la **ordenación** de variables es clasificar las variables de la más **restringida** a la menos restringida.
- En general, las variables deberían ser instanciadas en orden creciente a la talla de los dominios.
- Esta ordenación se puede realizar **estáticamente** al inicio de la búsqueda o **dinámicamente** reordenando las variables restantes cada vez que una variable es asignada.

# Ordenación de Variables

- La ordenación dinámica no es siempre posible en todos los algoritmos.
- Por ejemplo, con el backtracking cronológico no hay información extra disponible durante la búsqueda que se pueda utilizar para tomar una elección diferente a de la ordenación inicial.

# Ordenación de Variables

- Se han analizado varias heurísticas de ordenación de variables.
- La más común se basa en el principio “**first-fail**” (FF) que se puede explicar como:  
*“Para tener éxito, probar primero donde es más probable que falle.”*
- El principio FF puede parecer desencaminado, porque nuestro objetivo no es fallar. La razón es que si la solución parcial no nos lleva a una solución completa, entonces cuanto antes lo descubramos mejor.

# Ordenación de Variables

- La variables con **menos alternativas** posibles es seleccionada para la instanciación.
- El **orden** de la instanciación de las variables es **distinto** para las distintas ramas del árbol de búsqueda y se determina de forma **dinámica**
- Sin embargo, si la solución parcial nos lleva a una solución completa, entonces las variables restantes deben ser instanciadas y aquella con el dominio más pequeño es más difícil encontrarle un valor, ya que instanciar las otras variables hace que se reduzca aun más su dominio y nos lleve a la inconsistencia. Así el principio anterior se puede modificar como:
- **“Tratar con los casos mas difíciles primero: ellos se harán cada vez más difíciles si los aplazamos”**

# Ordenación de Variables

- Otra heurística: cuando todas las variables tienen el mismo número de valores es elegir la variable que participa en más restricciones
  - en la ausencia de información más específica sobre qué restricciones son más difíciles de satisfacer
  - Esta heurística sigue también el principio de tratar con los casos más duros primero.
- Existe otra heurística para la ordenación estática de variables que es apropiada para el backtracking cronológico:
  - seleccionar la variable que tenga el mayor número de restricciones con las variables pasadas.

# Ordenación de Valores

# Ordenación de Valores

- Una vez que se ha tomado la decisión de **instanciar** una variable, hay que asignarle uno de los posibles **valores**.
- el orden en el cual se consideran estos valores tiene un **impacto** substancial en tiempo para encontrar la **primera** solución.
- Sin embargo si se requiere encontrar **todas** las soluciones del problema, entonces la ordenación de valores es **indiferente**.
- Si no hay conflictos lo más sencillo es considerar el orden “natural” de los dominios y asignar desde el menor valor hasta el mayor.



# Ordenación de Valores

- Ordenación diferente: árbol de búsqueda diferente
- **Ventaja** si nos asegura que la rama que nos lleva a una **solución** se explora **antes** que otras ramas que nos llevan a situaciones **sin salida**.
- Por ejemplo si el CSP tiene una solución y se selecciona un primer valor correcto para cada variable, entonces se puede encontrar una solución sin necesidad de backtracking.

# Ordenación de Valores

- ¿Cómo deberíamos seleccionar el valor de una variable?
- Puede que **ningún valor** sea **posible**, en ese caso, cada valor de la variable actual tendrá que ser considerado y el orden **no importa**.
- Por el contrario, si podemos encontrar una **solución** completa **basada** en las instanciaciones **pasadas**, seleccionaremos el valor **más probable** de tener **éxito** y menos probable de generar un conflicto
- Por lo tanto se aplica el principio de “**primero el exitoso**”

# Ordenación de Valores

- Otra heurística es preferir el valor (de los disponibles) que nos permita **resolver** de una forma **más fácil** el CSP.
- Esto requiere estimar la dificultad de resolver el CSP. (coste adicional)
- Una posible heurística es **preferir** aquellos valores que **maximizan** el número de **opciones** disponibles.
- también coste adicional.

# Ordenación de Restricciones

- Poco trabajo se ha realizado sobre ordenación de restricciones.
- La idea se basa en ordenar las restricciones desde la más restrictiva a la menos restrictiva de forma que podemos lo antes posible las ramas del árbol de búsqueda.
- La más común se basa en el principio “**first-fail**”

# Algoritmos híbridos

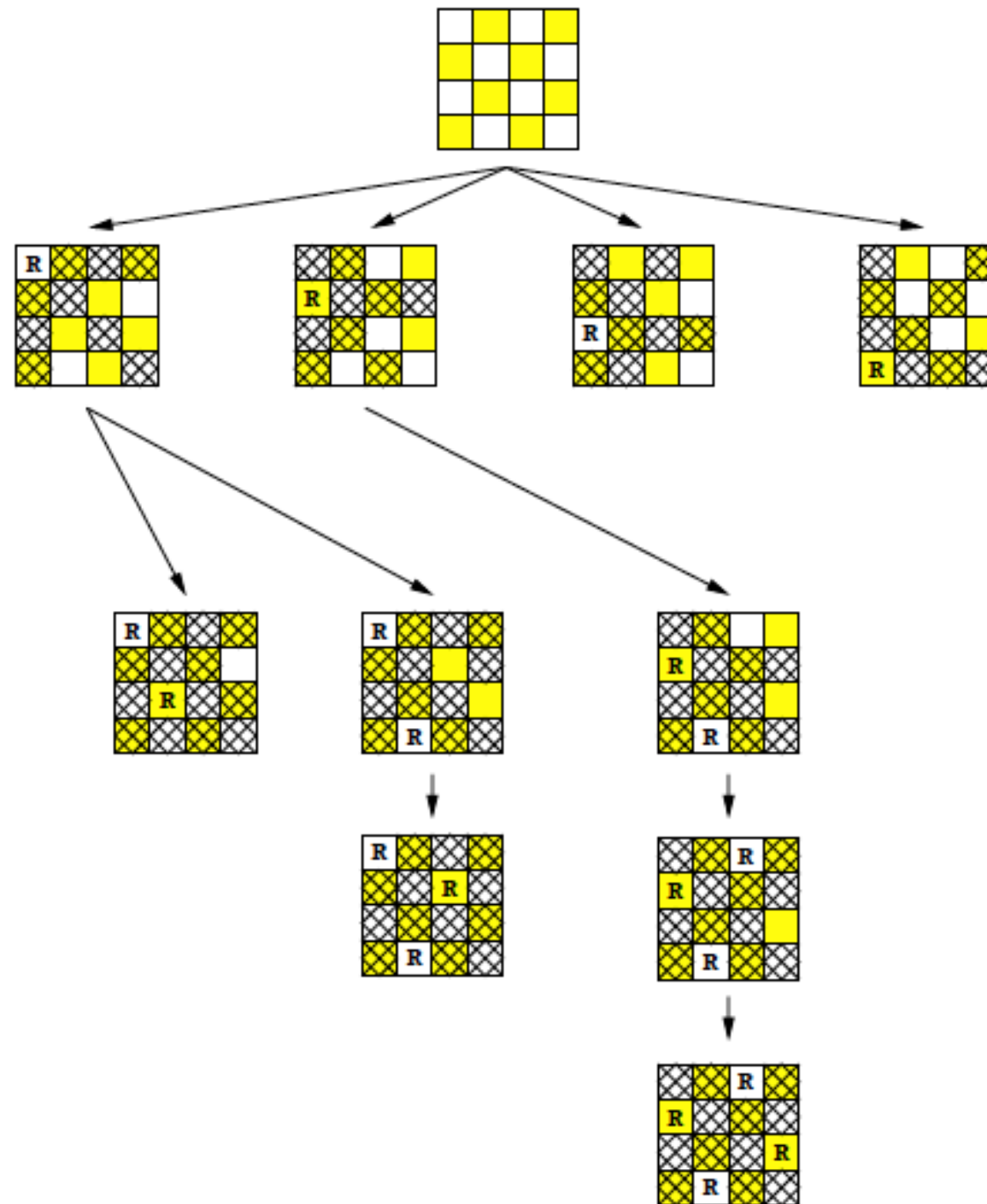
# Forward checking

- Para cada estado, mantener información sobre los posibles valores para las variables que quedan por asignar:
- Cada vez que se asigna un nuevo valor a una variable, quitar del dominio de las variables por asignar, aquellos valores que no sean consistentes con el nuevo valor asignado
- Cuestiones técnicas:
  - Cada nodo del árbol debe contener el estado junto la lista de valores posibles en las variables por asignar
  - Muy fácil de usar en conjunción con la heurística MRV (minimum remaining values)

# Ejemplo

- Ejemplo (seleccionando variables en orden alfabético):
  - $\{AL=R\}$  --> eliminar R del dominio de GR
  - $\{AL=R, CA=R\}$  --> eliminar R de los dominios de HU, SE y MA
  - $\{AL=R, CA=R, CO=A\}$  --> eliminar A de los dominios de MA, SE y JA
  - $\{AL=R, CA=R, CO=A, GR=V\}$  --> eliminar V de los dominios de JA y MA
- En este momento, Malaga no tiene ningún valor posible. Por tanto, no es necesario seguir explorando en el árbol
- Es decir, se ha detectado inconsistencia sin necesidad de asignar valores a Huelva y Jaen, y para cualquier extensión de la asignación parcial construida hasta el momento

# Ejemplo: 4-Reinas





# Técnicas de consistencia

# Técnicas de consistencia

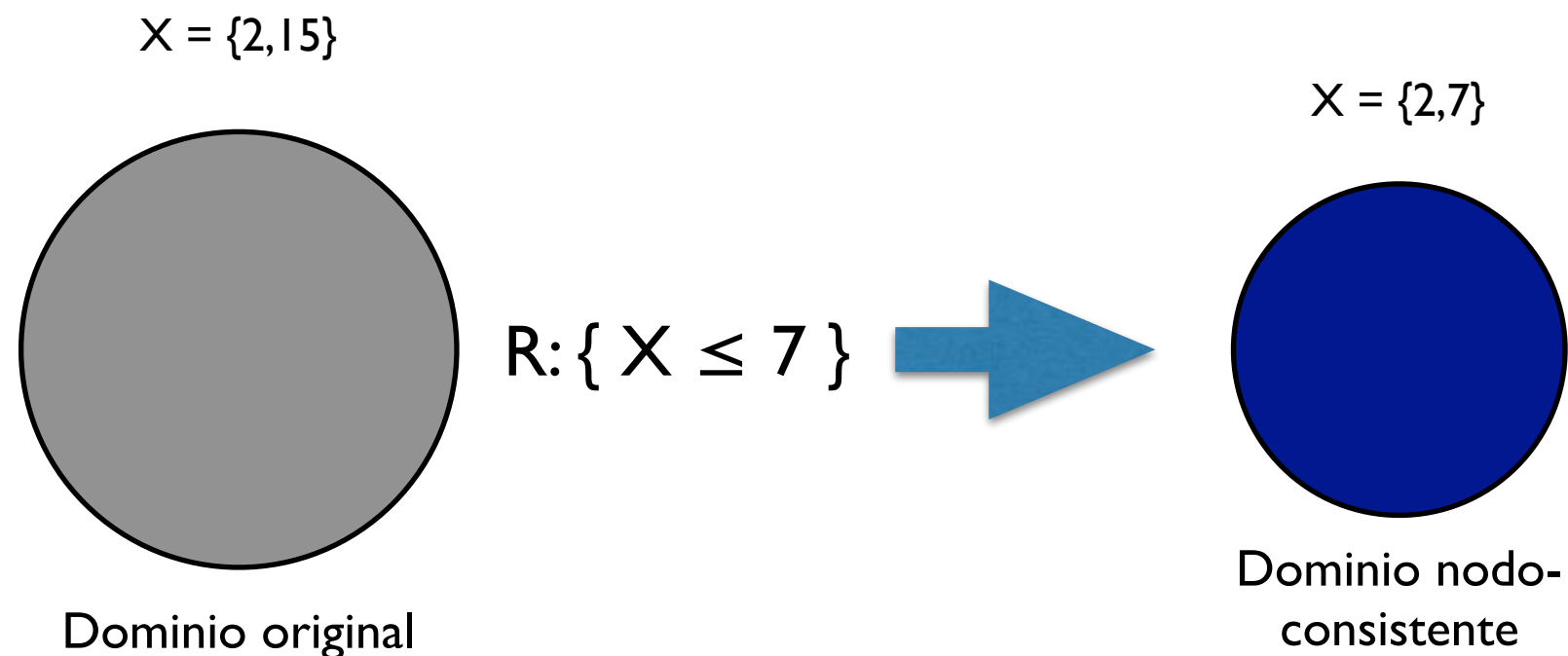
- Formas de **mejorar la eficiencia** de los algoritmos de búsqueda.
- Borran valores inconsistentes de las variables y ayudan a **podar el espacio de búsqueda**.
- Estas técnicas de consistencia local se usan como etapas de **preproceso** donde se detectan y se eliminan las inconsistencias locales antes de empezar o durante la búsqueda con el fin de reducir el árbol de búsqueda.

# Técnicas de consistencia

- Profundidad del árbol
  - Cuando hay muchas variables es alta.
- Orden de ramificación
  - Cuando las variables pueden tomar muchos valores distintos.

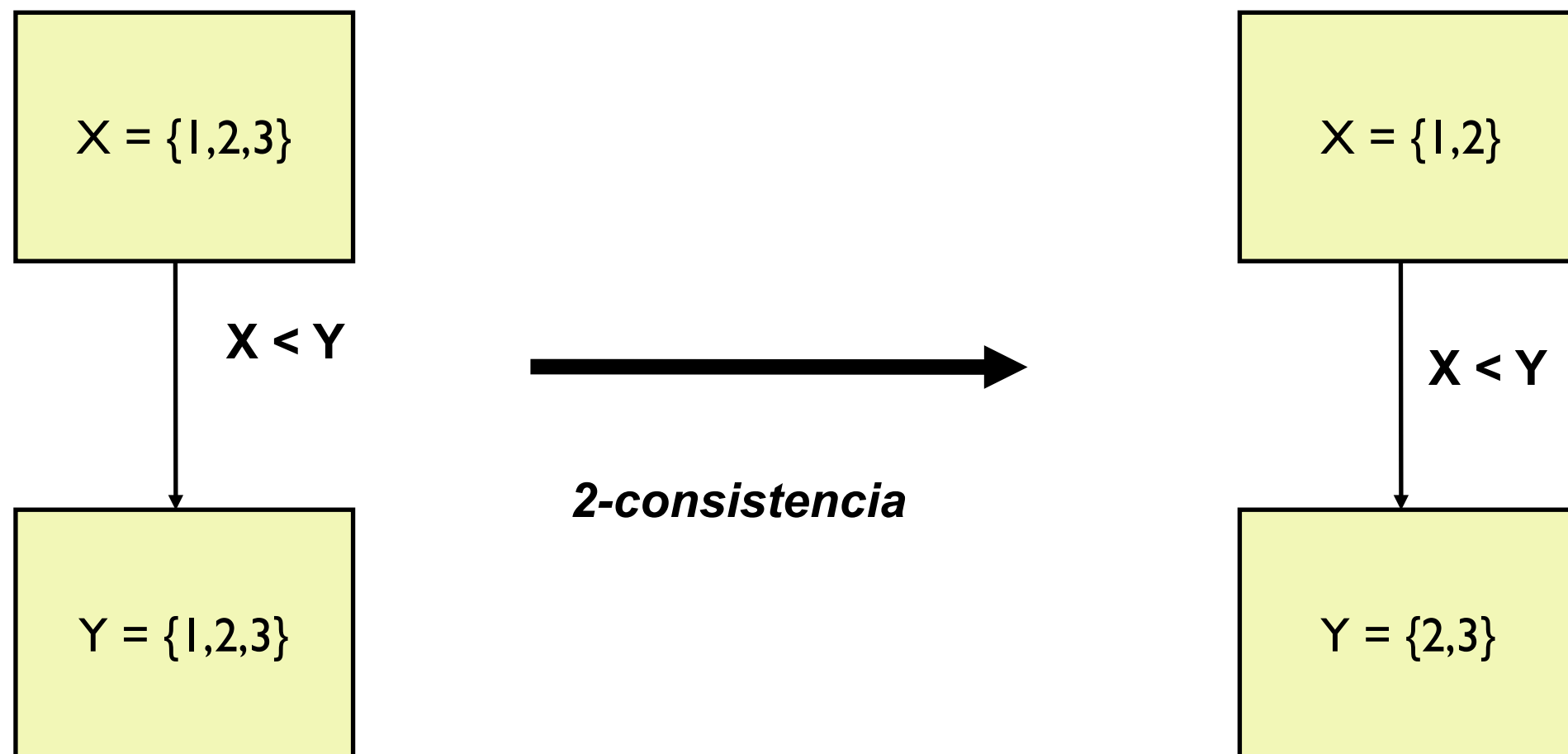
# Niveles de Consistencia Local

- Consistencia de Nodo (1-consistencia)
- Sobre restricciones de aridad 1



# Niveles de Consistencia Local

- Consistencia de Arco (2-consistencia)
- Sobre restricciones de aridad 2



# Consistencia en arcos

- Un **arco es una restricción** en la que hay una variable distinguida
- Notación:  $B > E$  ,  $CO \neq SE$  ,  $|V_i - V_j| \neq |i - j|$
- Arco consistente respecto a un conjunto de dominios asociados a un conjunto de variables:
  - Para cualquier valor del dominio asociado a la variable distinguida del arco, existen valores en los dominios de las restantes variables que satisfacen la restricción del arco

# Consistencia en arcos

- Ejemplo: si SE y CO tienen ambas asignadas como dominio {rojo, verde} , el arco  $CO \neq SE$  es consistente
- si el dominio de SE es {rojo} y el de CO es {rojo, verde} , el arco  $CO \neq SE$  no es consistente.
- Nota : el arco puede hacerse consistente eliminando rojo del dominio de CO

# Algoritmo AC3

## Objetivo:

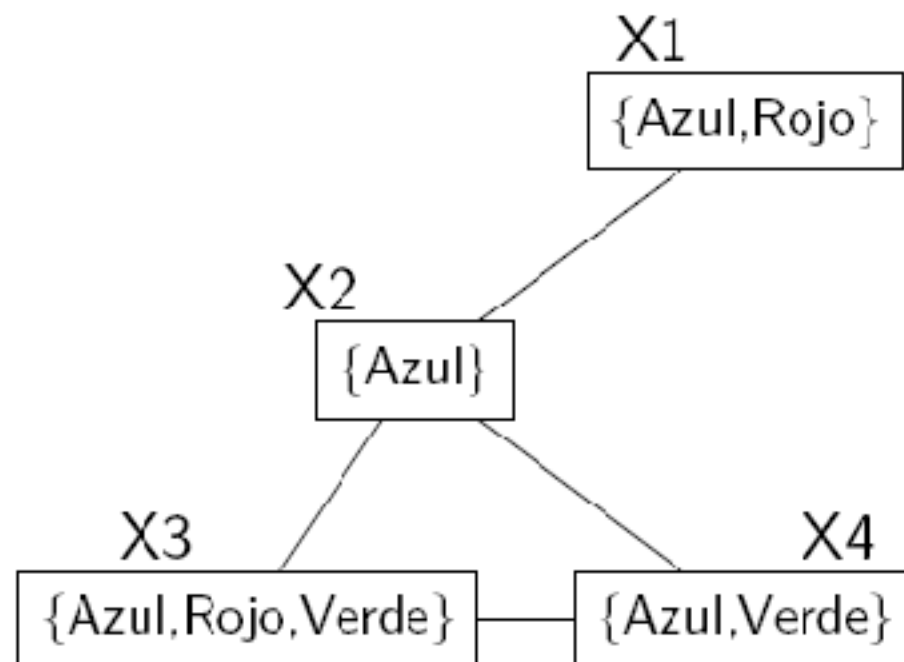
- a partir de los dominios iniciales de las variables devolver un conjunto de dominios actualizado tal que todos los arcos del problema sean consistentes
- Actualización de dominios:
  - Si un arco es inconsistente, podemos hacerlo consistente eliminando del dominio de la variable distinguida aquellos valores para los que no existen valores en los dominios de las restantes variables que satisfagan la restricción



# Algoritmo AC3

- Revisión de arcos:
- si el dominio de una variable se actualiza, es necesario revisar la consistencia de los arcos en los que aparece la variable como variable no distinguida
- Criterio de parada:
  - Todos los arcos son consistentes respecto a los dominios de las variables
  - O algún dominio queda vacío (inconsistencia)

# Ejemplo 1

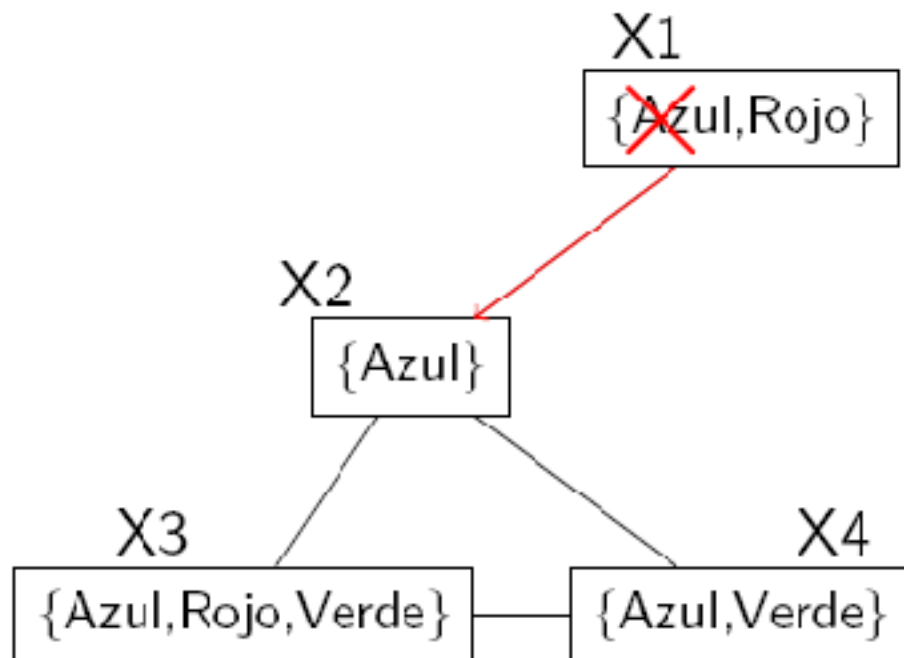


Lista de arcos inicial:

(X1,X2), (X2,X1), (X2,X3), (X3,X2),  
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

# Ejemplo 1

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

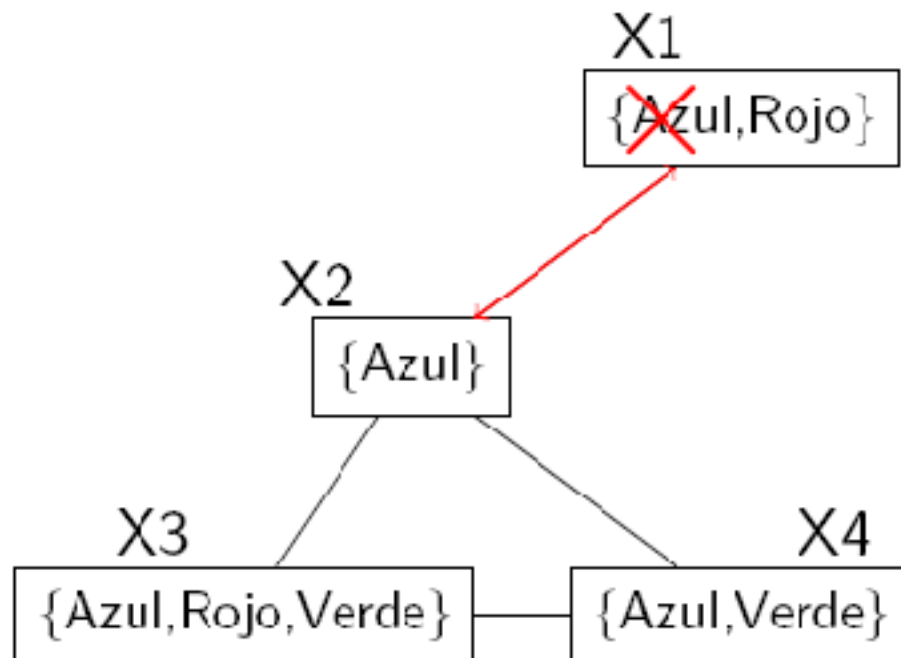


Lista de arcos inicial:

~~(X1, X2)~~ (X2, X1), (X2, X3), (X3, X2),  
(X2, X4), (X4, X2), (X3, X4), (X4, X3)

# Ejemplo 1

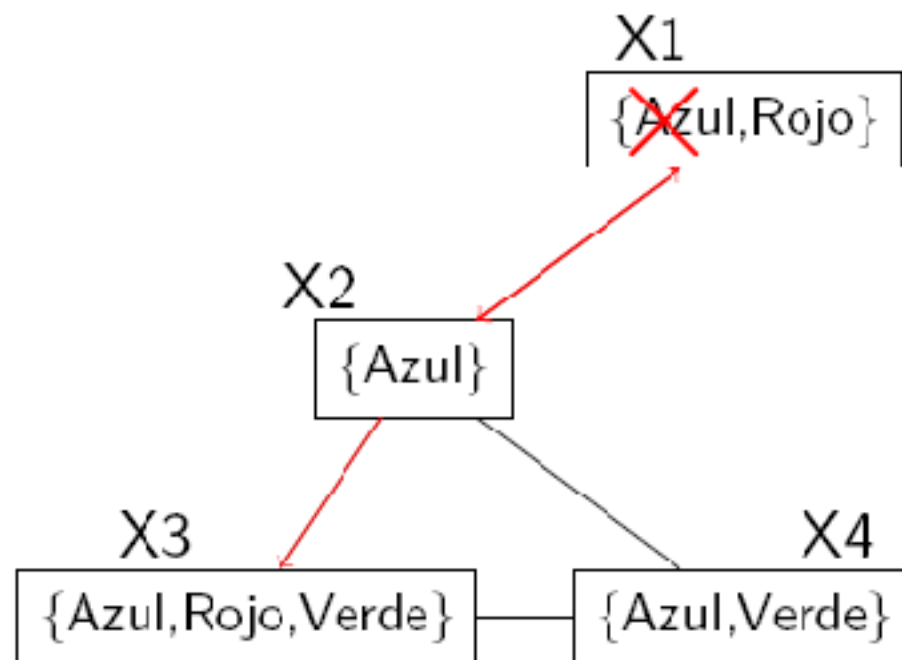
1.  $X_1 - X_2 \rightarrow$  Quitar Azul de  $X_1$
2.  $X_2 - X_1 \rightarrow$  Todo consistente



Lista de arcos inicial:

~~(X1,X2)~~ ~~(X2,X1)~~ (X2,X3), (X3,X2),  
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

# Ejemplo 1

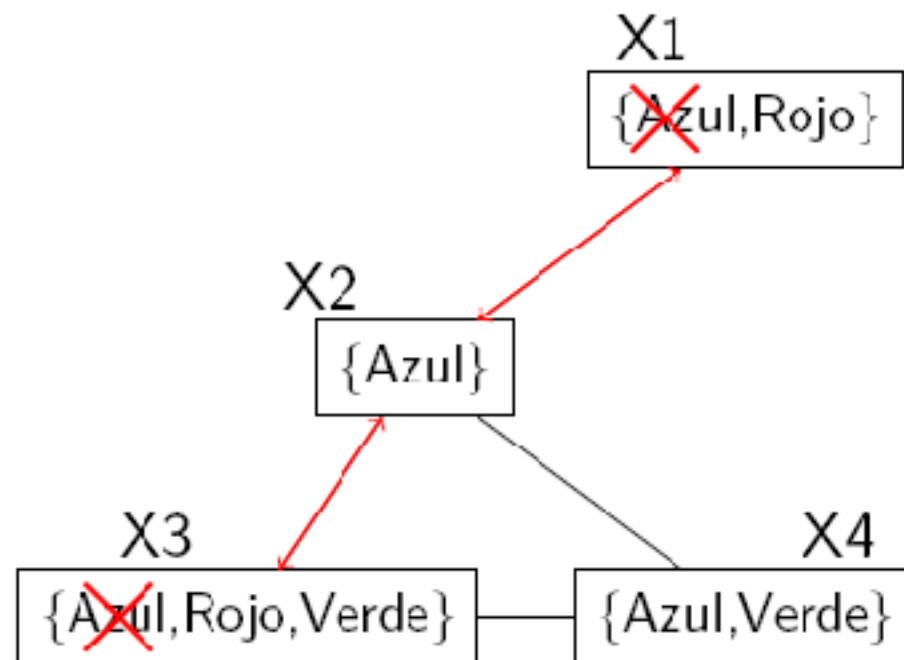


1.  $X_1 - X_2 \rightarrow$  Quitar Azul de  $X_1$
2.  $X_2 - X_1 \rightarrow$  Todo consistente
3.  $X_2 - X_3 \rightarrow$  Todo consistente

Lista de arcos inicial:

~~(X1,X2)~~ ~~(X2,X1)~~ ~~(X2,X3)~~, (X3,X2),  
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

# Ejemplo 1

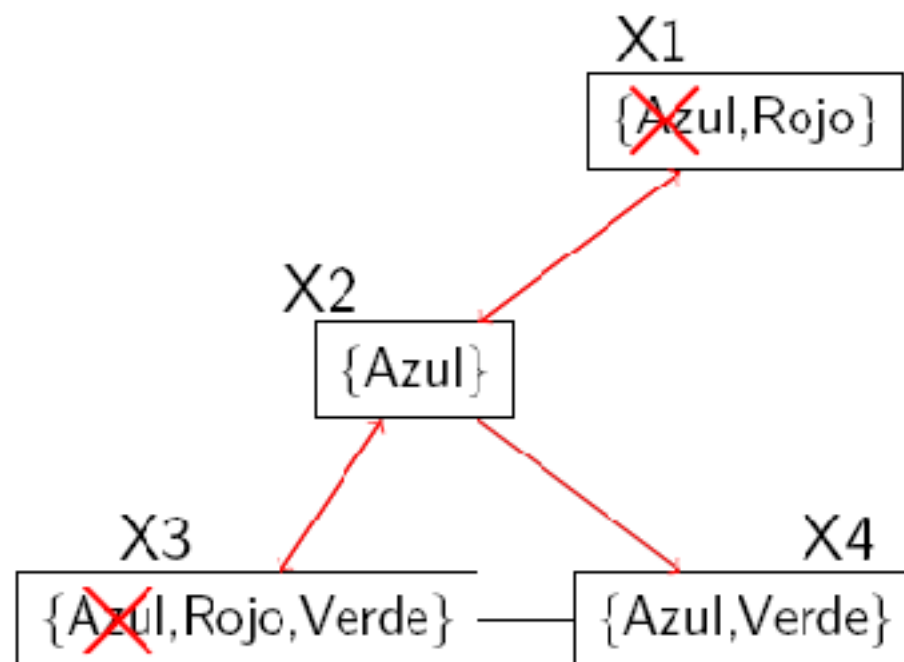


1.  $X_1 - X_2 \rightarrow$  Quitar Azul de  $X_1$
2.  $X_2 - X_1 \rightarrow$  Todo consistente
3.  $X_2 - X_3 \rightarrow$  Todo consistente
4.  $X_3 - X_2 \rightarrow$  Quitar Azul de  $X_3$ , Tendríamos que añadir  $X_4 - X_3$  pero ya está

Lista de arcos inicial:

~~(X1,X2), (X2,X1), (X1,X3), (X3,X1),~~  
(X2,X4), (X4,X2), (X3,X4), (X4,X3)

# Ejemplo 1

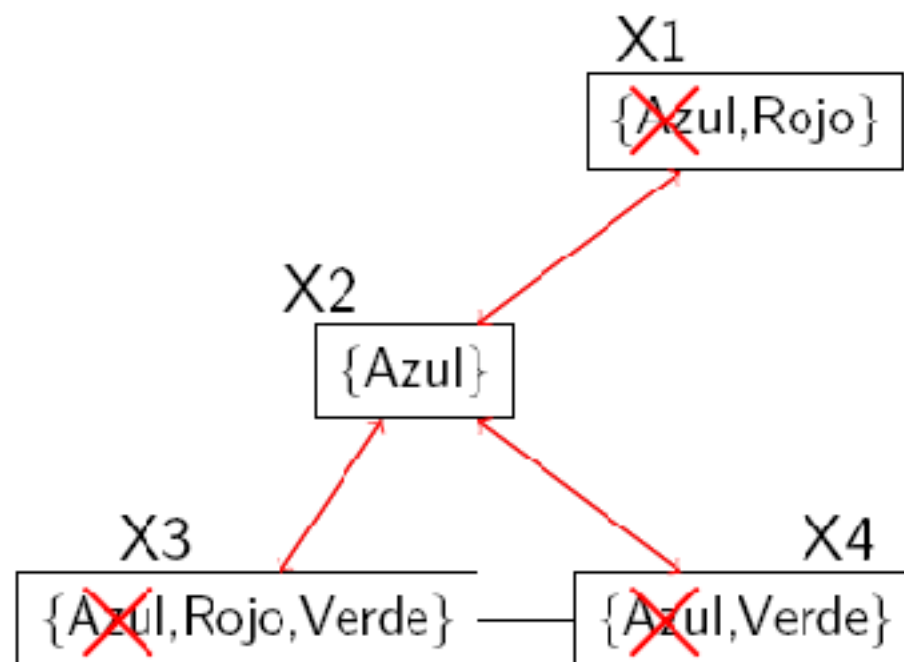


1.  $X_1 - X_2 \rightarrow$  Quitar Azul de  $X_1$
2.  $X_2 - X_1 \rightarrow$  Todo consistente
3.  $X_2 - X_3 \rightarrow$  Todo consistente
4.  $X_3 - X_2 \rightarrow$  Quitar Azul de  $X_3$ , Tendríamos que añadir  $X_4 - X_3$  pero ya está
5.  $X_2 - X_4 \rightarrow$  Todo consistente

Lista de arcos inicial:

~~(X1,X2)~~, ~~(X2,X1)~~, ~~(X2,X3)~~, ~~(X3,X2)~~,  
~~(X3,X4)~~, (X4,X2), (X3,X4), (X4,X3)

# Ejemplo 1



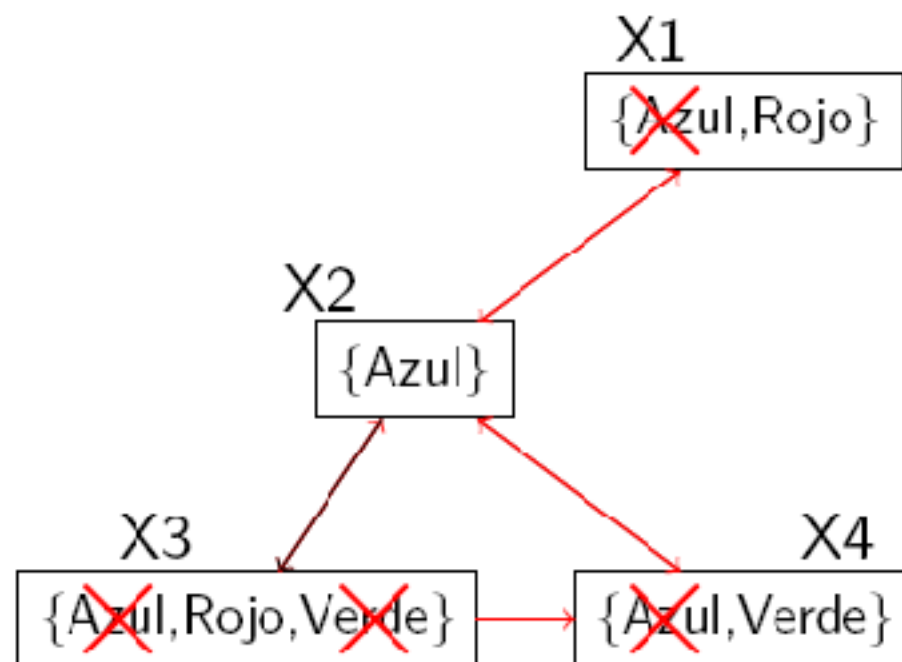
1.  $X_1 - X_2 \rightarrow$  Quitar Azul de  $X_1$
2.  $X_2 - X_1 \rightarrow$  Todo consistente
3.  $X_2 - X_3 \rightarrow$  Todo consistente
4.  $X_3 - X_2 \rightarrow$  Quitar Azul de  $X_3$ , Tendríamos que añadir  $X_4 - X_3$  pero ya está
5.  $X_2 - X_4 \rightarrow$  Todo consistente
6.  $X_4 -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

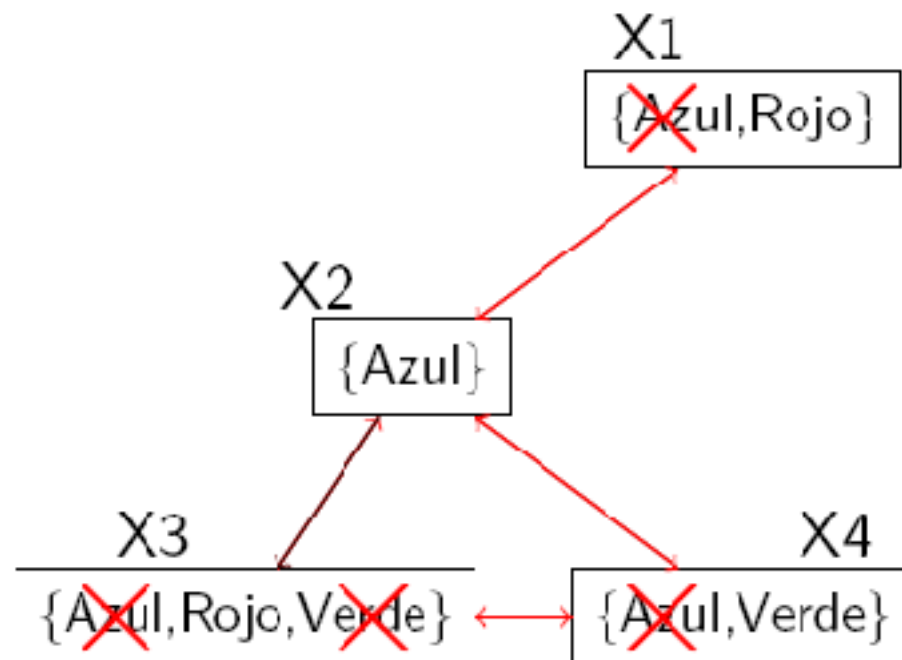


Lista de arcos inicial:

(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 1

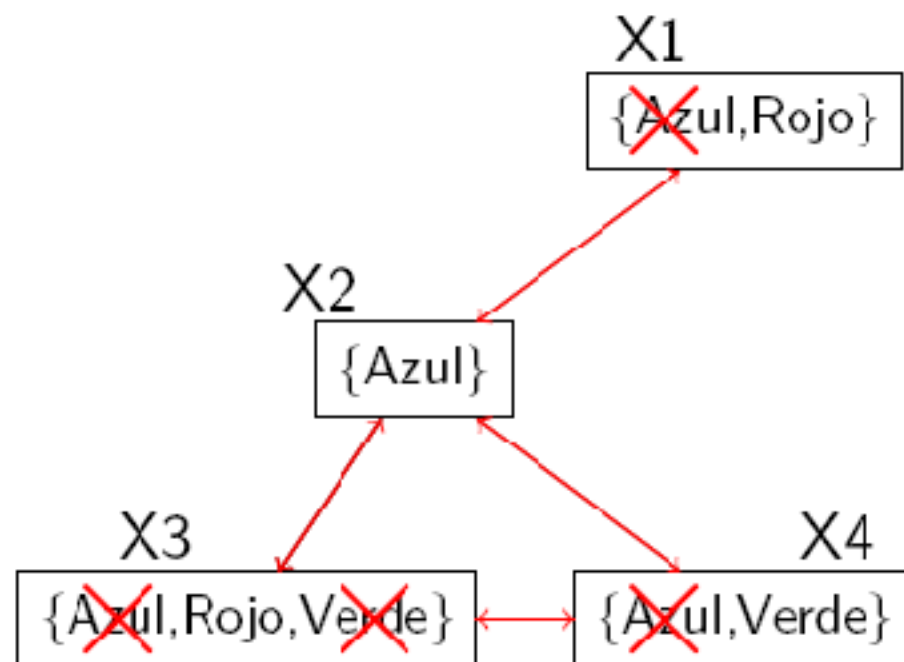


Lista de arcos inicial:

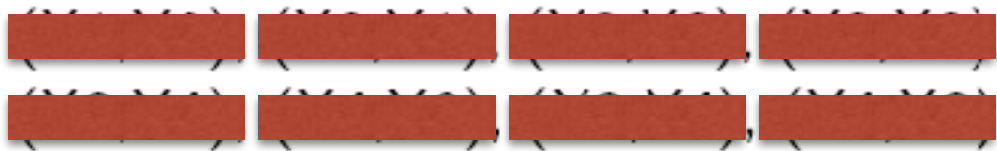
(X2,X3),

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 1



Lista de arcos inicial:



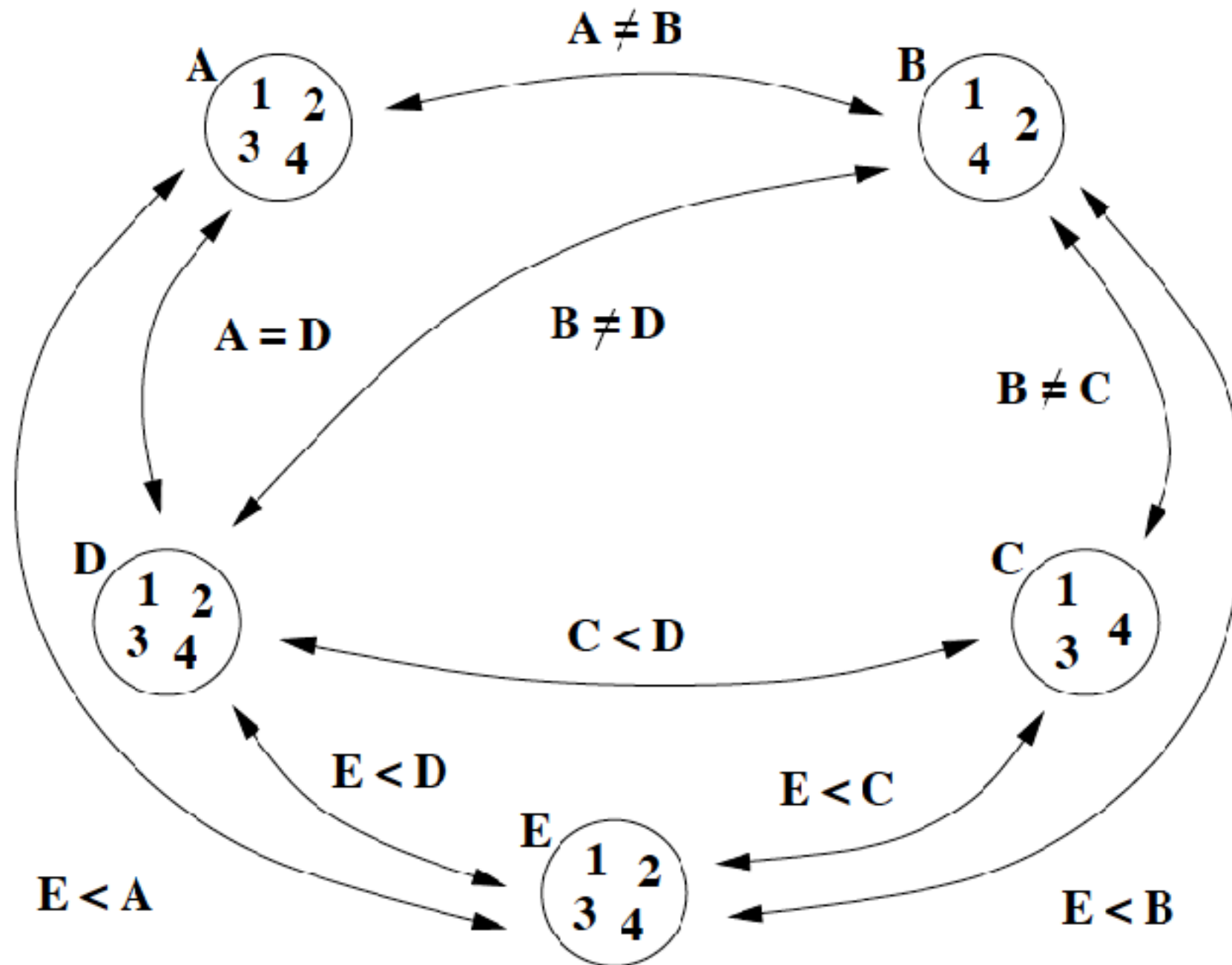
1.  $X_1 - X_2 \rightarrow$  Quitar Azul de  $X_1$
2.  $X_2 - X_1 \rightarrow$  Todo consistente
3.  $X_2 - X_3 \rightarrow$  Todo consistente
4.  $X_3 - X_2 \rightarrow$  Quitar Azul de  $X_3$ , Tendríamos que añadir  $X_4 - X_3$  pero ya está
5.  $X_2 - X_4 \rightarrow$  Todo consistente
6.  $X_4 -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

# Ejemplo 2

- Planificación de las acciones de un robot (Poole, pag.149): Un robot necesita planificar cinco actividades (A, B, C, D y E), donde cada actividad ha de comenzar en un momento en el tiempo (1, 2, 3, o 4) y dura exactamente una unidad de tiempo.
- Restricciones:
  - La actividad B no puede realizarse en el momento número 3.
  - La actividad C no puede realizarse en el momento número 2.
  - Las actividades A y B no pueden realizarse simultáneamente.
  - Las actividades B y C no pueden realizarse simultáneamente.
  - La actividad C ha de realizarse antes de la D.
  - Las actividades B y D no pueden realizarse simultáneamente.
  - Las actividades A y D han de realizarse simultáneamente.
  - La actividad E ha de ser la primera.

# Ejemplo 2

- Representación como grafo:



# Ejemplo 2

Variables y dominios	Arcos pendientes de revisar
(A 1 2 3 4) (B 1 2 4) (C 1 3 4) (D 1 2 3 4) (E 1 2 3 4)	$\underline{A} \neq B, A \neq \underline{B}, \underline{B} \neq C, B \neq \underline{C}, \underline{C} < D, C < \underline{D}, \underline{B} \neq D, B \neq \underline{D},$ $\underline{A} = D, A = \underline{D}, \underline{A} > E, A > \underline{E}, \underline{B} > E, B > \underline{E}, \underline{C} > E, C > \underline{E},$ $\underline{D} > E, D > \underline{E}$
(A 1 2 3 4) (B 1 2 4) (C 1 3) (D 1 2 3 4) (E 1 2 3 4)	$C < \underline{D}, \underline{B} \neq D, B \neq \underline{D}, \underline{A} = D, A = \underline{D}, \underline{A} > E, A > \underline{E}, \underline{B} > E,$ $B > \underline{E}, \underline{C} > E, C > \underline{E}, \underline{D} > E, D > \underline{E}, \underline{B} \neq C$
(A 1 2 3 4) (B 1 2 4) (C 1 3) (D 2 3 4) (E 1 2 3 4)	$\underline{B} \neq D, B \neq \underline{D}, \underline{A} = D, A = \underline{D}, \underline{A} > E, A > \underline{E}, \underline{B} > E, B > \underline{E},$ $\underline{C} > E, C > \underline{E}, \underline{D} > E, D > \underline{E}, \underline{B} \neq C, \underline{C} < D$
(A 2 3 4) (B 1 2 4) (C 1 3) (D 2 3 4) (E 1 2 3 4)	$A = \underline{D}, \underline{A} > E, A > \underline{E}, \underline{B} > E, B > \underline{E}, \underline{C} > E, C > \underline{E}, \underline{D} > E,$ $D > \underline{E}, \underline{B} \neq C, \underline{C} < D, A \neq \underline{B}$
(A 2 3 4) (B 1 2 4) (C 1 3) (D 2 3 4) (E 1 2 3)	$\underline{B} > E, B > \underline{E}, \underline{C} > E, C > \underline{E}, \underline{D} > E, D > \underline{E}, \underline{B} \neq C, \underline{C} < D,$ $A \neq \underline{B}, \underline{A} > E$
(A 2 3 4) (B 2 4) (C 1 3) (D 2 3 4) (E 1 2 3)	$B > \underline{E}, \underline{C} > E, C > \underline{E}, \underline{D} > E, D > \underline{E}, \underline{B} \neq C, \underline{C} < D, A \neq \underline{B},$ $\underline{A} > E, A \neq B, B \neq \underline{C}, B \neq \underline{D}$



# Ejemplo 2

Variables y dominios	Arcos pendientes de revisar
(A 2 3 4) (B 2 4) (C 3) (D 2 3 4) (E 1 2 3)	$C > \underline{E}$ , $\underline{D} > E$ , $D > \underline{E}$ , $\underline{B} \neq C$ , $\underline{C} < D$ , $A \neq \underline{B}$ , $\underline{A} > E$ , $\underline{A} \neq B$ , $B \neq \underline{C}$ , $B \neq \underline{D}$ , $C < \underline{D}$
(A 2 3 4) (B 2 4) (C 3) (D 2 3 4) (E 1 2)	$\underline{D} > E$ , $D > \underline{E}$ , $\underline{B} \neq C$ , $\underline{C} < D$ , $A \neq \underline{B}$ , $\underline{A} > E$ , $\underline{A} \neq B$ , $B \neq \underline{C}$ , $B \neq \underline{D}$ , $C < \underline{D}$ , $\underline{B} > E$ , $\underline{C} > E$
(A 2 3 4) (B 2 4) (C 3) (D 4) (E 1 2)	$\underline{B} > E$ , $\underline{C} > E$ , $\underline{C} < D$ , $\underline{B} \neq D$ , $\underline{A} = D$ , $D > \underline{E}$
(A 2 3 4) (B 2) (C 3) (D 4) (E 1 2)	$\underline{A} = D$ , $D > \underline{E}$ , $\underline{A} \neq B$ , $B \neq \underline{C}$ , $B \neq \underline{D}$ , $B > \underline{E}$
(A 4) (B 2) (C 3) (D 4) (E 1 2)	$D > \underline{E}$ , $\underline{A} \neq B$ , $B \neq \underline{C}$ , $B \neq \underline{D}$ , $B > \underline{E}$ , $A \neq \underline{B}$ , $A = \underline{D}$ , $A > \underline{E}$
(A 4) (B 2) (C 3) (D 4) (E 1)	$A \neq \underline{B}$ , $A = \underline{D}$ , $A > \underline{E}$ , $\underline{A} > E$ , $\underline{B} > E$ , $\underline{C} > E$ , $\underline{D} > E$
(A 4) (B 2) (C 3) (D 4) (E 1)	

# Sistemas

- **Choco: A Constraint Programming System**
- Cassowary constraint solver, an open source project for constraint satisfaction (accessible from C, Java, Python and other languages).
- Comet, a commercial programming language and toolkit
- Gecode, an open source portable toolkit written in C++ developed as a production-quality and highly efficient implementation of a complete theoretical background.
- JaCoP, an open source Java constraint solver.
- ...



# Choco

- Open-source (BSD license)
- Hosted on Sourceforge.net
- Created in 1996 by François Laburthe and Narenda Jussien
- Associated with Ecole des Mines de Nantes (Nantes, France), Bouygues (Paris, France) and Amadeus (Nice, France)

# Choco

- Developed to support CP research in combinatorial optimization
  - ‘Easy’ to use
  - Extensible
  - Modular
  - ‘Optimized’
- Library vs. autonomous system