Eines Lògiques i Problemes Complexos

Resolució de Restriccions i Programació Lògica amb Constraints

Mateu Villaret

IMA, UdG

13 de febrer del 2010

Resum

Motivació

- **Intenció:** atacar problemes complexos que es poden veure com a problemes de combinatòria (discrets).

Motivació

- **Intenció:** atacar problemes complexos que es poden veure com a problemes de combinatòria (discrets).
- **Problema:** els algorísmes dedicats són molt sensibles a petits canvis d'enunciat (d'altra banda tan comuns...)

Motivació

- **Intenció:** atacar problemes complexos que es poden veure com a problemes de combinatòria (discrets).
- **Problema:** els algorísmes dedicats són molt sensibles a petits canvis d'enunciat (d'altra banda tan comuns...)
- **Solució:** *"The combination of logic programming (adequate for stating the problems) and consistency techniques (efficient for solving them) gives rise to an appealing and unique language for solving discrete combinatorial problems."*

Constraint Logic Programming

Altres eines

La resolució de restriccions o el "**Constraints Solving**" defineixen un paradigma en si mateix. De fet hi ha molts altres entorns a part del lògic per poder usar les tècniques de constraints:

- CLP: Sicstus, gprolog, ECLiPSe, SWI Prolog, ...
- CCP (Concurrent Constraint Programming): Oz, Mozart.
- ILoG (llibreries per a C++): CPLEX, programació lineal, ...
- Tècniques Multi Agent ...
- SMT
- ...

Fonament i Orígens

La **Programació lògica amb Constraints** està basada en:

- Aspecte **declaratiu** de la programació lògica
- Eficiència de les tècniques de **consistència** per a l'assignació de valors a variables restringides: per mantenir consistència, *node*, *arc* o *path*-consistency i fusió amb algorismes de cerca (*backtracking*) resultant en tècniques com el *forward checking*.

Fonament i Orígens

La **Programació lògica amb Constraints** està basada en:

- Aspecte **declaratiu** de la programació lògica
- Eficiència de les tècniques de **consistència** per a l'assignació de valors a variables restringides: per mantenir consistència, *node*, *arc* o *path*-consistency i fusió amb algorismes de cerca (*backtracking*) resultant en tècniques com el *forward checking*.

Orígens:

- Pares: John Jaffar and Jean-Louis Lassez, and Colmerauer (finals 80's) **Prolog III**, European Computer-Industry Research Centre (ECRC) (finals 80's) **CHIP** (Constraint Handling in Prolog), ...

Resum

Definició (i)

Un problema de resolució de restriccions (CSP) binari consta de:

- Un **conjunt de variables**: $x_1, x_2, \ldots x_n$,
- un **conjunt de dominis** per a les variables: $D_1, D_2, \ldots, D_n$,
- un **conjunt de restriccions** $\mathcal{C}$ entre parells de variables x_i, x_j per a $i < j \leq n$ i que denotem: $C_{\langle i,j \rangle}(x_i, x_j)$

Diem que els valors v_i, v_j (per a les variables x_i, x_j) **són consistents** sii $C_{\langle i,j \rangle}(v_i, v_j)$ és cert.

Definició (i)

Un problema de resolució de restriccions (CSP) binari consta de:

- Un **conjunt de variables**: $x_1, x_2, \ldots x_n$,
- un **conjunt de dominis** per a les variables: $D_1, D_2, \ldots, D_n$,
- un **conjunt de restriccions** $\mathcal{C}$ entre parells de variables x_i, x_j per a $i < j \leq n$ i que denotem: $C_{\langle i,j \rangle}(x_i, x_j)$

Diem que els valors v_i, v_j (per a les variables x_i, x_j) **són consistents** sii $C_{\langle i,j \rangle}(v_i, v_j)$ és cert.

Considerem $x_1 \in \{1,2,3,4,5\}, x_2, x_3 \in \{1,2,3\}$

Sigui $\mathcal{C}$:

$$\begin{aligned} C_{\langle 1,2 \rangle}(x_1, x_2) &= \quad x_1 \geq x_2 - 1, \\ C_{\langle 1,3 \rangle}(x_1, x_3) &= \quad x_1 \leq x_3 + 1, \\ C_{\langle 2,3 \rangle}(x_2, x_3) &= \quad x_2 = x_3 - 1 \end{aligned}$$

Els valors $1, 2, 3$ per a x_1, x_2 i x_3 són consistents.

Definició (i)

Un problema de resolució de restriccions (CSP) binari consta de:

- Un **conjunt de variables**: $x_1, x_2, \ldots x_n$,
- un **conjunt de dominis** per a les variables: $D_1, D_2, \ldots, D_n$,
- un **conjunt de restriccions** $\mathcal{C}$ entre parells de variables x_i, x_j per a $i < j \leq n$ i que denotem: $C_{\langle i,j \rangle}(x_i, x_j)$

Diem que els valors v_i, v_j (per a les variables x_i, x_j) **són consistents** sii $C_{\langle i,j \rangle}(v_i, v_j)$ és cert.

Solucions

Donat un CSP l'objectiu és trobar totes les assignacions de valors a variables $\langle (x_1, v_1), (x_2, v_2), \ldots, (x_n, v_n) \rangle$ de manera que totes les restriccions $C_{\langle i,j \rangle}(x_i, x_j)$ per a $i < j \leq n$ es satisfacin.

Definició (ii)

- Un **CSP general** pot tenir restriccions que relacionin més de dues variables.
  - Tot CSP general es pot traduir a un CSP binari.
  - La idea és intercanviar el rol de les variables i les restriccions
- Un **CSP Booleà** restringeix les variables als dominis $\{0, 1\}$
  - Tot CSP es pot traduir a un CSP Booleà.
  - Codificació SAT: les variables booleanes indiquen si cert valor s'ha assignat a certa variable.

Com atacar el problema

S'ha de trobar una assignació que solucioni els constraints...

linear programming generar solucions senceres i explorar a partir d'aquestes.

- Bon comportament: programació lineal en reals $\in P$.
  En enters $\in NP$.
- Més difícil de modelitzar: calen equacions lineals...

Com atacar el problema

S'ha de trobar una assignació que solucioni els constraints...

Com a CSPs permet fer cerques en assignacions parcials

Algorismes de Cerca Sistemàtica	Generate and test Backtracking BackJumping Backmarcking ...
Tècniques de Consistència	Node Consistency Arc-consistency Path-consistency
Propagació de Restriccions:	Forward-checking

Generate and Test

L'algorisme de generate and test

```
GT (X:variables, V:assignacio, C:constraints)
   si X=={} llavors
        retorna satisfa(V,C)?
   sino
        x <- tria una variable de X
        pertot  v ∈ domini_de x fer
                R := GT(X -{x}, V++(x,v),C)
                si R!=FALS llavors retorna R
        fper
        retorna FALS
fi

crida: GT(X,{},C)
```

Es generen (completament) totes les possibles assignacions fins que s'en troba una que satisfà el problema!!!

Chronologic Backtracking

L'algorisme de Chronological backtracking

algorisme BT(X:vars, A:assignacio, C:constraints)
  si $X == \{\}$ **llavors retorna** A
  $x :=$ tria una variable d'X
  percada $v \in$ domini_de(x) **fes**
    si consistent($A++(x,v), C$) **llavors**
      $R :=$ BT($X - \{x\}, A++(x,v), C$)
      si $R! =$ fail **llavors** retorna R
  fper
  retorna fail
falgorisme

Crida BT($X, \{\}, C$)

El procediment consistent(A, C), comprova que tots els parells de variables **ja assignats** a A siguin consistents a C.

Debilitats del Chronological Backtracking

- **Trashing**: no tenir en compte la rao del conflicte!
  - P.ex: $A, B, C, D, E \in \{1..10\}, A > E$. L'algorisme provarà totes les assignacions per a B, C i D abans de passar a $A = 2$. Solució: **Backjumping**

Debilitats del Chronological Backtracking

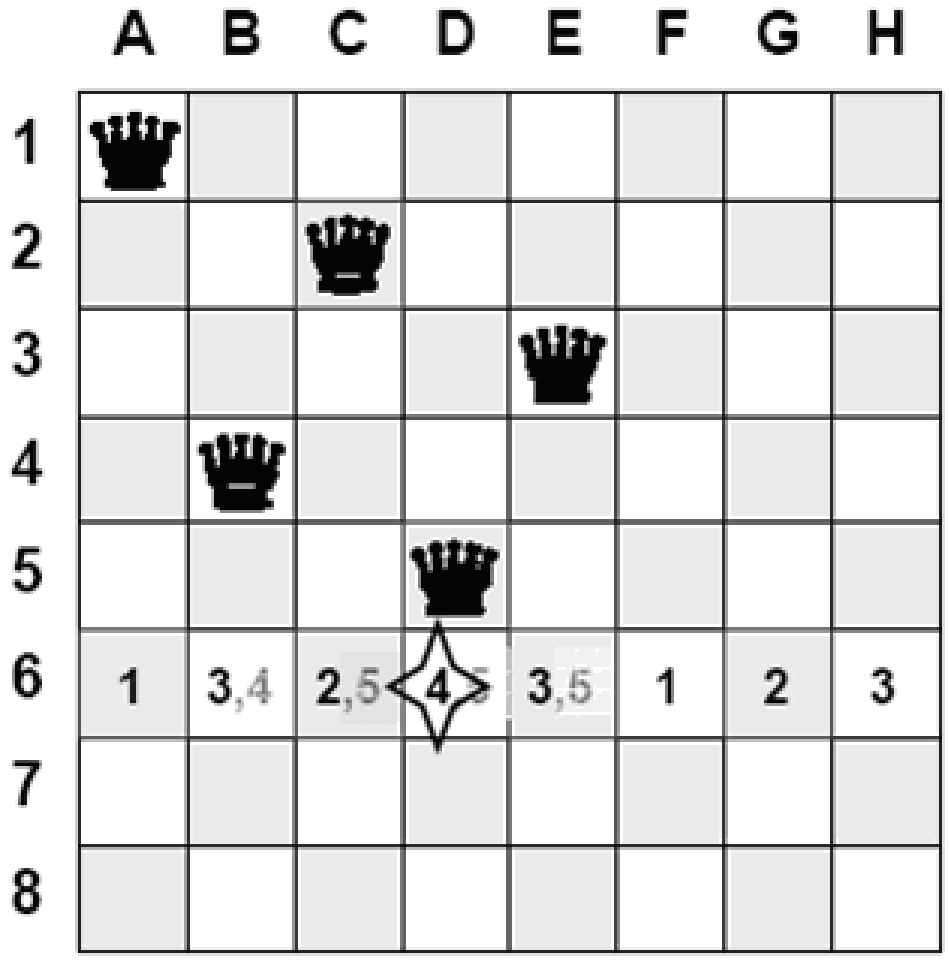
- **Trashing**: no tenir en compte la rao del conflicte!
  - P.ex: $A, B, C, D, E \in \{1..10\}, A > E$. L'algorisme provarà totes les assignacions per a B, C i D abans de passar a $A = 2$.
    Solució: **Backjumping**
- **Treball Redundant**: es repeteixen innecesàriament controls de consistència.
  - P.ex: $A, B, C, D, E \in \{1..10\}, B + 8 < D, C = 5 * E$. Quan es dona valors a C i E, els valors $1, \ldots, 9$ es controlaran repetidament per a D.
    Solució: **Backchecking i Backmarking**

Debilitats del Chronological Backtracking

- **Trashing**: no tenir en compte la rao del conflicte!
  - P.ex: $A, B, C, D, E \in \{1..10\}, A > E$. L'algorisme provarà totes les assignacions per a B, C i D abans de passar a $A = 2$.
    Solució: **Backjumping**
- **Treball Redundant**: es repeteixen innecesàriament controls de consistència.
  - P.ex: $A, B, C, D, E \in \{1..10\}, B + 8 < D, C = 5 * E$. Quan es dona valors a C i E, els valors $1, \ldots, 9$ es controlaran repetidament per a D.
    Solució: **Backchecking i Backmarking**
- **Detecció tardana del conflicte**: la violació de restriccions es detecta només quan es coneixen els valors.
  - P.ex: $A, B, C, D, E \in \{1..10\}, A = 3 * E$. Que A ha de ser > 2 només es detecta quan es dóna valors a E.
    Solució: **Forward-Checking**

BackJumping

L'exemple de les 8 reines:



BackJumping (i)

Gaschnig's backjumping: Crida BJ($Vars, \{\}, C, 0$)

algorisme BJ(X:vars, A:assig, C:constr, PL:int)
  si $X == \{\}$ **llavors retorna** A
  $x :=$ tria la primera var d'X
  $L := PL + 1$; $Salt := 0$
  percada $v \in$ domini_de(x) **fes**
    $S :=$ consistent($A++(x, v, L), C, L$)
    si $S ==$ fail(J) **llavors**
      $Salt := max\{Salt, J\}$
    sino
      $Salt := PL$
      $R :=$ BJ($X - \{x\}, A++(x, v, L), C, L$)
      si $R! =$ fail(L) **llavors retorna** R
  fper
  retorna fail($Salt$)
falgorisme

BackJumping (ii)

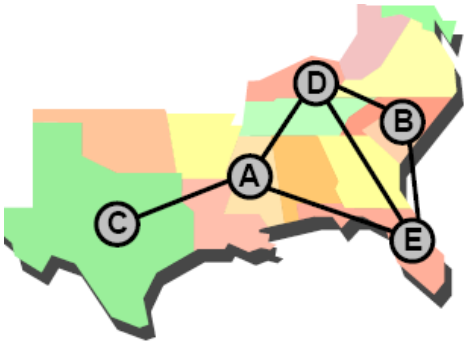
- La funció $\boxed{\texttt{consistent}(A++(x,v,L),C,L)}$ ens tornarà si l'assignació $A++(x,v,L)$ és consistent amb C o el nivell "òptim" d'inconsistència. (per a cada constraint violat, el nivell més baix (més llunyà) amb qui s'ha trobat conflicte)
- $\boxed{\textbf{si } S == \texttt{fail}(J) \textbf{ llavors } Salt := max\{Salt, J\}}$, anem recordant per a tots els valors de la variable que tractem, el nivell més proper de conflicte.
- $\boxed{\textbf{si } R! = \texttt{fail}(L) \textbf{ llavors } \texttt{retorna } R}$, fa que es propagui cap endarrera el backtracking provocant un backjumping o propaga l'assignació correcta en cas d'encert.
- $\boxed{\textbf{retorna } \texttt{fail}(Salt)}$ provoca la primera crida endarrera del backjumping (Backtracking).

BackJumping (iii)

- Altres possibilitats: **graph-based backjumping** (sempre salta a la variable més propera que tingui constraints amb la variable actual. Fatal per a problemes amb grafs complets...)

BackJumping (iii)

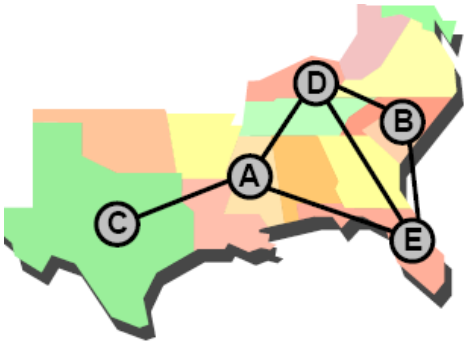
- Defecte del Gaschnig's backjumping:
  - El problema del "treball redundant" no s'estalvia amb el backjumping ja que s'ha de tornar a fer trossos que potser no tenen res a veure amb el conflicte...



node	vertex		
A	1		1
B	2		1
C	1 2		1 2
D	1 2 3	backjump	1 2
E	1 2 3		1 2 3

BackJumping (iii)

- Defecte del Gaschnig's backjumping:
  - El problema del "treball redundant" no s'estalvia amb el backjumping ja que s'ha de tornar a fer trossos que potser no tenen res a veure amb el conflicte...



node	vertex	
A	1	1
B	2	1
C	1 2	1 2
D	1 2 3	1 2
E	1 2 3	1 2 3

backjump

  - El **Backcheking** permet recordar que les assignacions (X, a) i (Y, b) són incompatibles, de manera que mentres sigui present l'assignació (X, a), (Y, b) no es probarà mai.
  - El **Backmarcking** exten el Backcheking recordant incompatibilitats i compatibilitats.
  - Altres opcions passen pel **Dynamic Backtracking** que permet "reordenar" les variables...

Altres idees

Per tal de fer la cerca més tractable hi ha altres propostes:

- Incompletes
  - Límit en backtracks, límit en profunditat o en amplada, límit de crèdit en alternatives ...
- Completes
  - Heurístiques tipus: ordenar variables segons dominis, o segons restricció, ordenar tria de valors ...

Introducció a les Tècniques de Consistència

$A < B, A \in \{3, \ldots, 7\}, B \in \{1, \ldots, 5\}$

Clarament no hi ha cap valor de B que permeti a A valer $5, 6$ o 7, així com tampoc hi ha cap valor de A que permeti a B valer $1, 2$ o 3, per tant el que segueix seria un CSP equivalent amb un espai de cerca menor:

$$A < B, A \in \{3, 4\}, B \in \{4, 5\}$$

Fixeu-vos però que $\langle (A, 4), (B, 4) \rangle$ no seria solució...

Introducció a les Tècniques de Consistència

$A < B, A \in \{3, \ldots, 7\}, B \in \{1, \ldots, 5\}$

Clarament no hi ha cap valor de B que permeti a A valer $5, 6$ o 7, així com tampoc hi ha cap valor de A que permeti a B valer $1, 2$ o 3, per tant el que segueix seria un CSP equivalent amb un espai de cerca menor:

$$A < B, A \in \{3, 4\}, B \in \{4, 5\}$$

Fixeu-vos però que $\langle (A, 4), (B, 4) \rangle$ no seria solució...

Modelitzem el CSP (binari) com un graf

- els nodes són els noms de les variables.
- cada arc entre dos nodes correspon a una restricció entre les dues variables dels nodes.
- els constraints unaris seran arcs cíclics.

Node Consistència

La **Consistència de node (NC)** és la més simple: *Un node d'una variable X és node consistent si tot valor del domini actual de X satisfà totes les restriccions unàries d'X. Un CSP és node consistent si ho són tots els seus nodes.*
Si un CSP és NC podem eliminar les restriccions unàries.

Algoritme NC

algorisme NC(G)
  percada $x \in$ nodes_de(G) **fes**
    percada $v \in$ domini_de(x) **fes**
      si v no satisfa algun constraint unari d'X **llavors**
        treu v de domini_de(x)
    fper
  fper
falgorisme

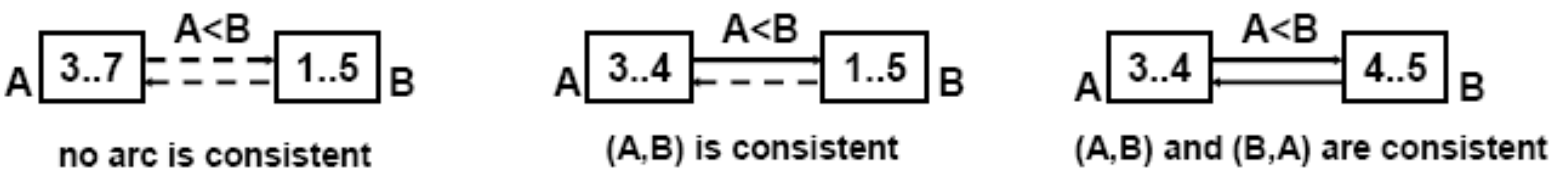
ARC Consistència

- La **Consistència d'arc (AC)**:*Un arc* (X_i, X_j) *és arc consistent sii per a cada valor* $v_i \in D_i$ *existeix un valor* $v_j \in D_j$ *tal que* $\langle (X_i, v_i), (X_j, v_j) \rangle$ *satisfà el constraint binari entre* X_i *i* X_j.
- Un CSP és arc consistent si ho són tots els seus arcs en totes direccions: (X_i, X_j) i (X_j, X_i).

ARC Consistència

- La **Consistència d'arc (AC)**: *Un arc* (X_i, X_j) *és arc consistent sii per a cada valor* $v_i \in D_i$ *existeix un valor* $v_j \in D_j$ *tal que* $\langle (X_i, v_i), (X_j, v_j) \rangle$ *satisfà el constraint binari entre* X_i *i* X_j.
- Un CSP és arc consistent si ho són tots els seus arcs en totes direccions: (X_i, X_j) i (X_j, X_i).

$A < B, A \in \{3, \ldots, 7\}, B \in \{1, \ldots, 5\}$



- l'arc-consistència d'(A, B) fa que el domini d'A sigui $\{3, 4\}$ (no toca el de B).
- l'arc-consistència d'(B, A) fa que el domini de B sigui $\{4, 5\}$ (no toca el d'A).
- al final tenim arc-consitència.

ARC Consistència

Clarament podem esborrar tots els valors del domini per obtenir arc consistència sense perdre solucions.

Algoritme REVISA

algorisme REVISA(X_i, X_j)
  $ESBORRAT := FALS$
  percada $v_i \in D_i$ **fes**
    si no hi ha $v_j \in D_j$ tal que (X_i, X_j) consistent **llavors**
      esborra v_i de D_i
      $ESBORRAT := CERT$
    fsi
  fper
  retorna $ESBORRAT$
falgorisme

Compte, no n'hi ha prou en aplicar REVISA un cop a cada arc...

ARC Consistència

Algoritme AC-1

algorisme `AC-1(`G`)`
  $Q := \{(X_i, X_j) \in arcs(G), i \neq j\}$
  repeteix
    $CANVIAT := FALS$
    percada `arc` $(X_i, X_j) \in Q$ **fes**
      $CANVIAT := REVISA(X_i, X_j)$ *or* $CANVIAT$
    fper
  finsque `no` $CANVIAT$
falgorisme

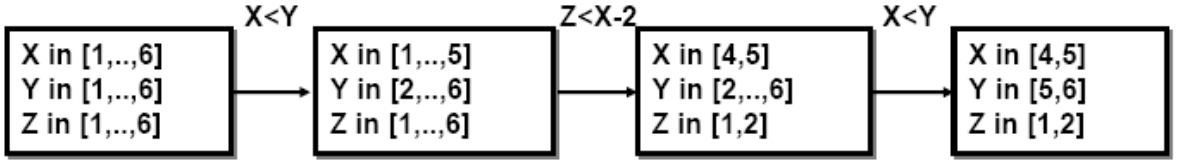
ARC Consistència

Algoritme AC-1

algorisme `AC-1(G)`
  $Q := \{(X_i, X_j) \in arcs(G), i \neq j\}$
  repeteix
    $CANVIAT := FALS$
    percada `arc` $(X_i, X_j) \in Q$ **fes**
      $CANVIAT := REVISA(X_i, X_j)$ *or* $CANVIAT$
    fper
  finsque `no` $CANVIAT$
falgorisme

Un sol canvi implica la revisió de tots els arcs...

Example: X in [1,..,6], Y in [1,..,6], Z in [1,..,6], X<Y, Z<X-2

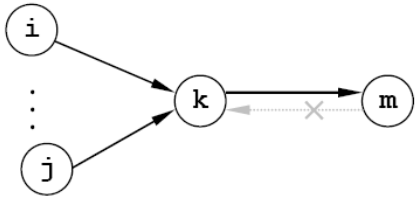


ARC Consistència

Algoritme AC-1

algorisme `AC-1(G)`
  $Q := \{(X_i, X_j) \in arcs(G), i \neq j\}$
  repeteix
    $CANVIAT := FALS$
    percada `arc` $(X_i, X_j) \in Q$ **fes**
      $CANVIAT := REVISA(X_i, X_j)$ *or* $CANVIAT$
    fper
  finsque `no` $CANVIAT$
falgorisme

- Canviar el domini de X_k només implica revisar arcs $(X_i's, X_k)$
- Si revisem (X_k, X_m) i canviem D_k no cal (re-)revisar (X_m, X_k)



ARC Consistència

Algoritme AC-3

algorisme AC-3(G)
  $Q := \{(X_i, X_j) \in arcs(G), i \neq j\}$
  mentre no buit Q **fes**
    Treu algun arc $(X_k, X_m) \in Q$
    si REVISA(X_k, X_m) **llavors**
      $Q := Q \cup \{(X_i, X_k) \in Q \| i \neq k, i \neq m\}$
    fsi
  fmentre
falgorisme

- Encara es poden evitar comprobacions de parells de valors ja que sovint es repeteixen
- AC-4, afegeix tècniques i eedd per a controlar-ho
- Si bé AC-3 és cúbic i AC-4 és quadràtic, el cost mitjà no és tant bo i requereix molta memòria.

ARC Consistència

Hi ha moltes seqüeles AC més encara: direccional, tenint en compte simetries, ordenant constraints,...
Usant AC podem esborrar molts valors incompatibles, però

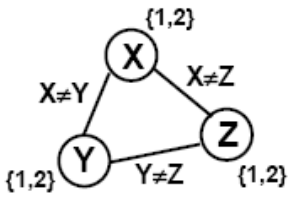
- Obtenim una solució?
- Arribem a saber si existeix una solució?

ARC Consistència

Hi ha moltes seqüeles AC més encara: direccional, tenint en compte simetries, ordenant constraints,...
Usant AC podem esborrar molts valors incompatibles, però

- Obtenim una solució?
- Arribem a saber si existeix una solució?NO!

El constraint $X, Y, Z \in \{1, 2\}, X \neq Y, Y \neq Z, Z \neq X$ és arc consistent però no te solució!!!

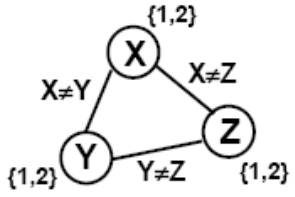


ARC Consistència

Hi ha moltes seqüeles AC més encara: direccional, tenint en compte simetries, ordenant constraints,...
Usant AC podem esborrar molts valors incompatibles, però

- Obtenim una solució?
- Arribem a saber si existeix una solució?NO!

El constraint $X, Y, Z \in \{1, 2\}, X \neq Y, Y \neq Z, Z \neq X$ és arc consistent però no te solució!!!



Ara bé:

- Si algun domini queda buit llavors el CSP no te solució
- Si tots els dominis són singletons llavors tenim una solució
- En general, AC poda l'espai de cerca

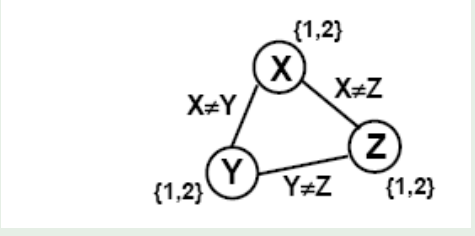
Com enfortir el nivell de consistència?

ARC Consistència

Hi ha moltes seqüeles AC més encara: direccional, tenint en compte simetries, ordenant constraints,...
Usant AC podem esborrar molts valors incompatibles, però

- Obtenim una solució?
- Arribem a saber si existeix una solució?NO!

El constraint $X, Y, Z \in \{1, 2\}, X \neq Y, Y \neq Z, Z \neq X$ és arc consistent però no te solució!!!



Ara bé:

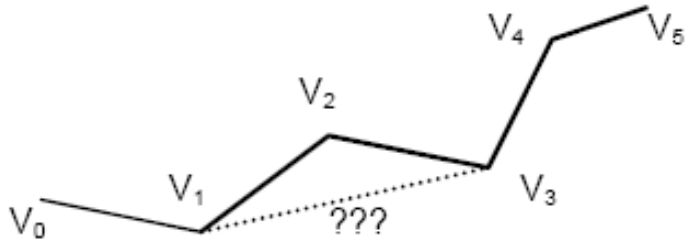
- Si algun domini queda buit llavors el CSP no te solució
- Si tots els dominis són singletons llavors tenim una solució
- En general, AC poda l'espai de cerca

Com enfortir el nivell de consistència?
Considerant més constraints junts.

PATH Consistència (i)

Definició

- El camí $(X_0, X_1, \ldots, X_m)$ és **camí (path) consistent** sii per a cada parell de valors $v_0 \in D_0$ i $v_m \in D_m$ satisfent totes les restriccions binaries entre X_0 i X_m, existeix una assignació de valors a les variables $X_1, \ldots, X_{m-1}$ tal que totes les restriccions binaries entres les variables veïnes X_i, X_{i+1} del camí es satisfan.
- Un CSP és path consistent sii cada camí ho és.



i els constraints entre V_1 i V_3?:

- només s'han de satisfer restriccions entre variables veines.
- n'hi ha prou amb explorar camins de llargada 2.

PATH Consistència (ii)

Relació entre AC i PC

- PC implica AC!
  - l'arc (i, j) és AC-consistent si el camí (i, j, i) és PC-consistent.
- PC és més potent que AC!

PATH Consistència (ii)

Relació entre AC i PC

- PC implica AC!
  - l'arc (i, j) és AC-consistent si el camí (i, j, i) és PC-consistent.
- PC és més potent que AC!
  - El constraint $X, Y, Z \in \{1, 2\}, X \neq Y, Y \neq Z, Z \neq X$ és AC però no és PC.
  - Pel camí (X, Y, Z) l'assignació $\langle (X, 1), (Z, 2) \rangle$ no permet trobar cap valor per Y...

PATH Consistència (ii)

Relació entre AC i PC

- PC implica AC!
  - l'arc (i, j) és AC-consistent si el camí (i, j, i) és PC-consistent.
- PC és més potent que AC!
  - El constraint $X, Y, Z \in \{1, 2\}, X \neq Y, Y \neq Z, Z \neq X$ és AC però no és PC.
  - Pel camí (X, Y, Z) l'assignació $\langle (X, 1), (Z, 2) \rangle$ no permet trobar cap valor per Y...
- AC esborra valors incompatibles dels dominis
  - PC esborra parells de valors
  - PC fa constraints explícits: $A < B, B < C \rightarrow A + 1 < C$
  - PC s'implementa amb operacions sobre matrius
  - hi ha moltes implementacions diferents: PC-1, ... PC-5, ...

PATH Consistència (iii)

Problemes:

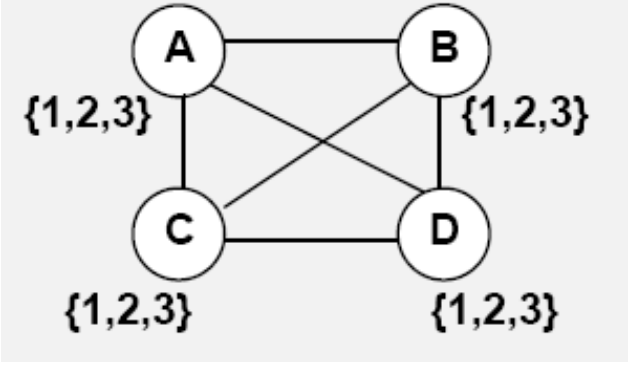
- gasta molta memòria: matrius per a parells de valors!
- ratio potència/eficiència dolent (pitjor que AC)

PATH Consistència (iii)

Problemes:

- gasta molta memòria: matrius per a parells de valors!
- ratio potència/eficiència dolent (pitjor que AC)

$$A, B, C, D \in \{1, 2, 3\}$$
$$A \neq B, A \neq C, A \neq D, B \neq C, B \neq D, C \neq D$$

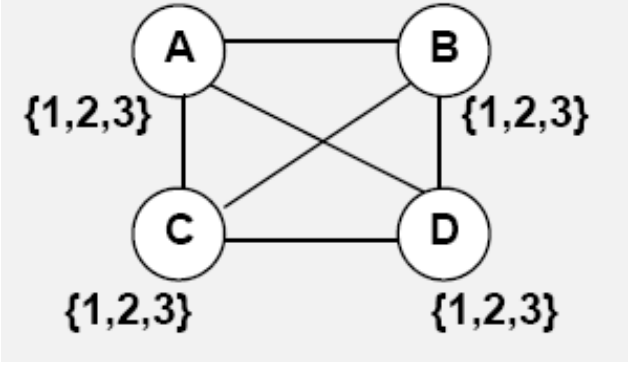


PATH Consistència (iii)

Problemes:

- gasta molta memòria: matrius per a parells de valors!
- ratio potència/eficiència dolent (pitjor que AC)

$$A, B, C, D \in \{1, 2, 3\}$$
$$A \neq B, A \neq C, A \neq D, B \neq C, B \neq D, C \neq D$$



- no és complet ja que aquest exemple és PC però no te solució.

k-Consistència (i)

Formalisme comú:

- AC: un valor d'una variable es propaga a una altra variable
- PC: un parell de valors es propaguen a una altra variable
- ...

Definició

Un CSP **és k-consistent** sii qualsevol assignació consistent de $(k-1)$ variables es pot extendre a una assignació consistent amb una variable més.

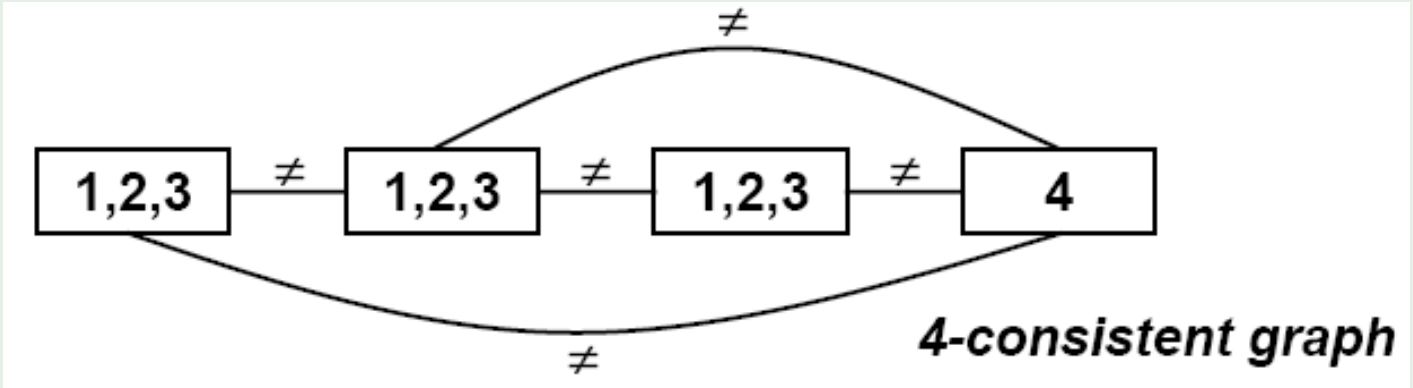
k-Consistència (i)

Formalisme comú:

- AC: un valor d'una variable es propaga a una altra variable
- PC: un parell de valors es propaguen a una altra variable
- ...

Definició

Un CSP **és k-consistent** sii qualsevol assignació consistent de $(k-1)$ variables es pot extendre a una assignació consistent amb una variable més.



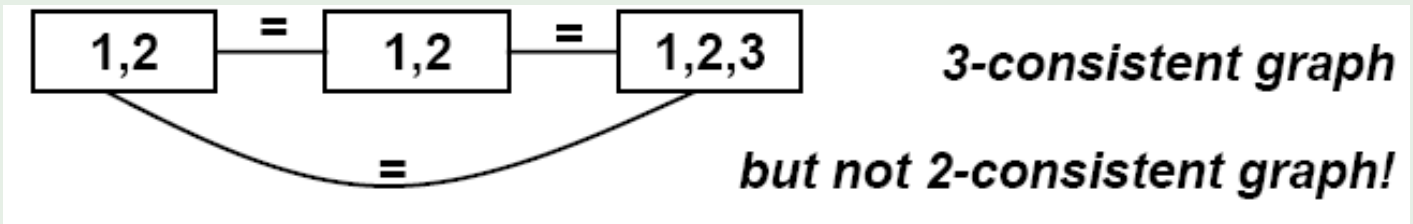
k-Consistència (i)

Formalisme comú:

- AC: un valor d'una variable es propaga a una altra variable
- PC: un parell de valors es propaguen a una altra variable
- ...

Definició

Un CSP **és k-consistent** sii qualsevol assignació consistent de $(k-1)$ variables es pot extendre a una assignació consistent amb una variable més.



k-Consistència (i)

Formalisme comú:

- AC: un valor d'una variable es propaga a una altra variable
- PC: un parell de valors es propaguen a una altra variable
- ...

Definició

Un CSP **és k-consistent** sii qualsevol assignació consistent de $(k-1)$ variables es pot extendre a una assignació consistent amb una variable més.

Definició

Un CSP **és strongly k-consistent** si és j-consistent per a tot $j <= k$.

k-Consistència (ii)

Propietats:

- Hi ha algorismes per a la strongly k-consistència per a $k > 2$ però són molt cars.
- La strongly k-consistència només és "completa" quan el graf te n nodes i $k = n$,
- si $k < n$ en general no evitem la cerca:
  Per exemple un clicke amb desigualtats entre els nodes...

Com resoldre els CSPs? (i)

Hem vist dues propostes diferents:

- **Cerca sistemàtica**
  - són completes: troben la solució o demostren la seva inexistència
  - molt lentes (exponencial): exploren assignacions "visiblement" errònies
- **Tècniques de consistència**
  - incompletes: segueixen quedant valors inconsistents als dominis
  - prou ràpides (polinòmics)

Com resoldre els CSPs? (i)

Hem vist dues propostes diferents:

- **Cerca sistemàtica**
  - són completes: troben la solució o demostren la seva inexistència
  - molt lentes (exponencial): exploren assignacions "visiblement" errònies
- **Tècniques de consistència**
  - incompletes: segueixen quedant valors inconsistents als dominis
  - prou ràpides (polinòmics)

Aprofitem els avantatges de les dues propostes:

- etiquetar les variables pas a pas
- mantenir consistència després de cada pas d'assignació

Com resoldre els CSPs? (ii)

Algorisme d'etiquetatge amb control de consistència

algorisme $\texttt{LBL}(G, va)$
  si $va > |nodes(G)|$ **llavors retorna** $nodes(G)$
  percada $v \in D_{va}$ **fes**
    si $\texttt{consistent}(G, va)$ **llavors**
      $R := \texttt{LBL}(G, va + 1)$
      si $R \neq fail$ **llavors retorna** R
  fper
  retorna *fail*
falgorisme

- La crida per a resoldre el **labelling** d'un CSP G seria: $\texttt{LBL}(G, 1)$
- $\texttt{consistent}(G, va)$ ens permetrà determinar com fer el control de consistència

Backtracking (revisited i)

Mantenim consistència amb els valors ja triats

Backtracking = bàsic "look-back"

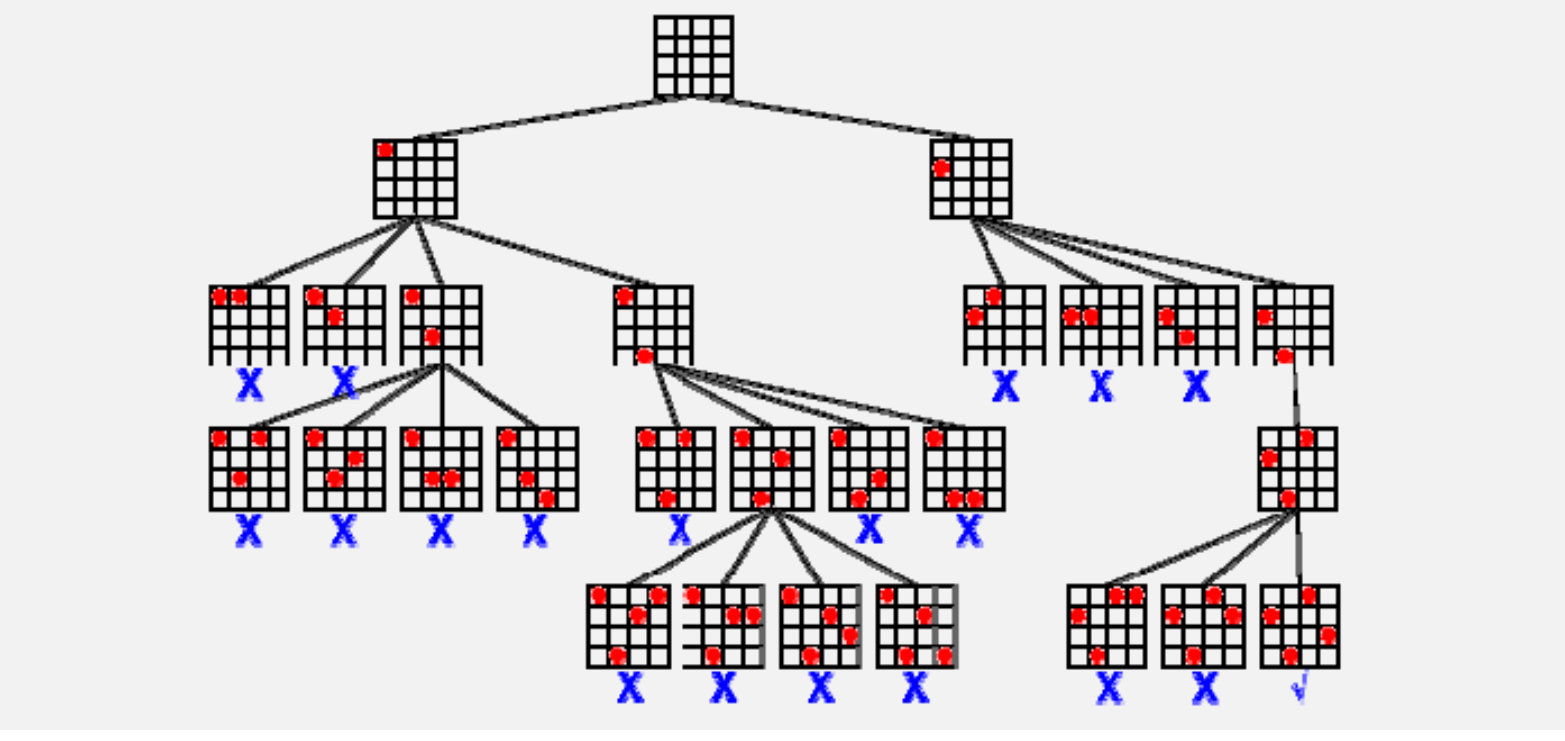
Algoritme AC-Backtracking

algorisme AC-Back(G, va)
  $Q := \{(X_i, X_{va}) \in arcs(G), i < va\}$
  $CONSISTENT := CERT$
  mentre $CONSISTENT \wedge$ no buit Q **fes**
    Treu algun arc $(X_k, X_{va}) \in Q$
    $CONSISTENT :=$ not REVISA(X_k, X_{va})
  fmentre
  retorna $CONSISTENT$
falgorisme

Compte, les variables ja assignades tenen només un valor al domini, per tant si s'ha de revisar algun domini...

Backtracking (revisited ii)

El problema de les 4 reines que no s'amenacen amb backtraking, l'arbre de cerca:



Forward Checking i

Val més prevenir solucions errònies quan abans millor...

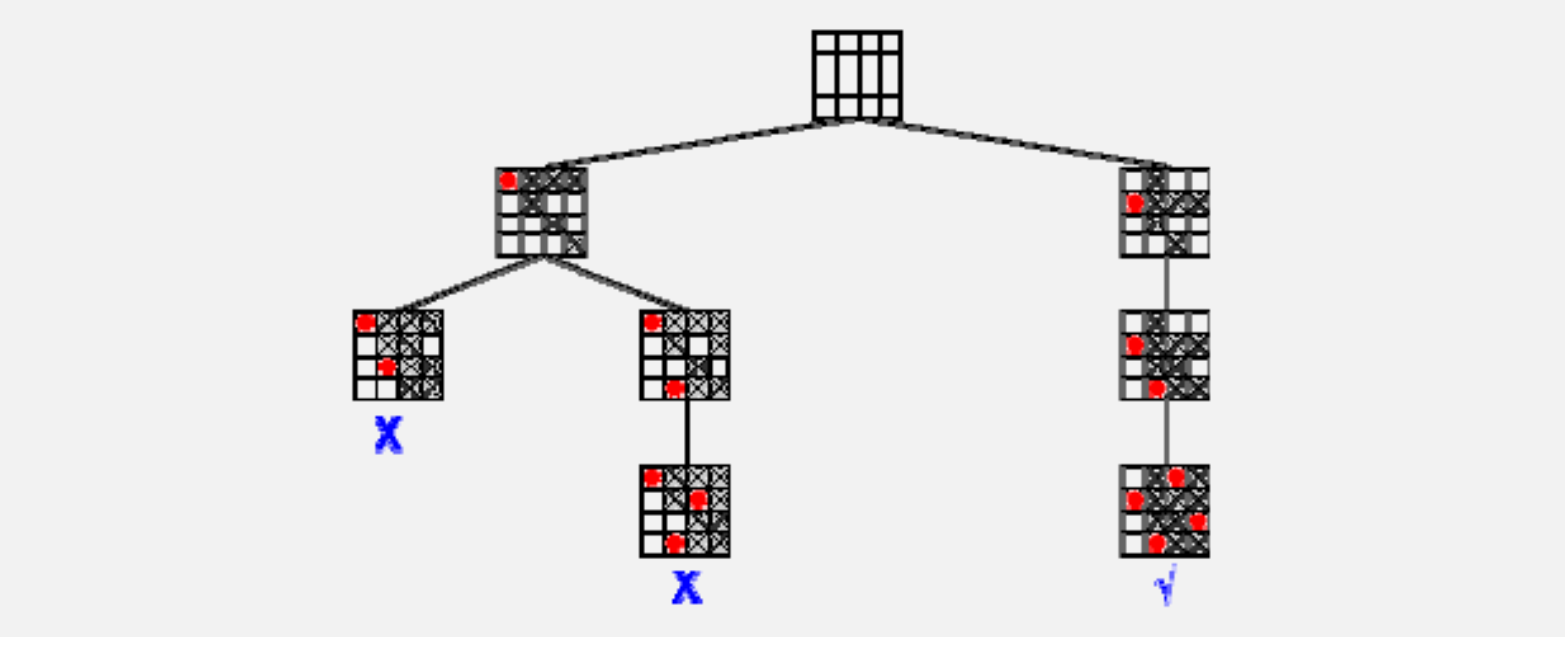
Forward checking permet garantir consistència entre la variable assignada i les variables relacionades mitjançant constraints.

Algoritme AC-Forward-checking

algorisme AC-FC(G, va)
  $Q := \{(X_i, X_{va}) \in arcs(G), i > va\}$
  $CONSISTENT := CERT$
  mentre $CONSISTENT \wedge$ no buit Q **fes**
    Treu algun arc $(X_k, X_m) \in Q$
    si REVISA(X_k, X_m) **llavors**
      $CONSISTENT :=$ no buit D_k
  fmentre
  retorna $CONSISTENT$
falgorisme

Forward Checking ii

El problema de les 4 reines que no s'amenacen amb forward-checking, l'arbre de cerca:



Partial look ahead i

Extenem també el control de consistència entre les variables "futures".

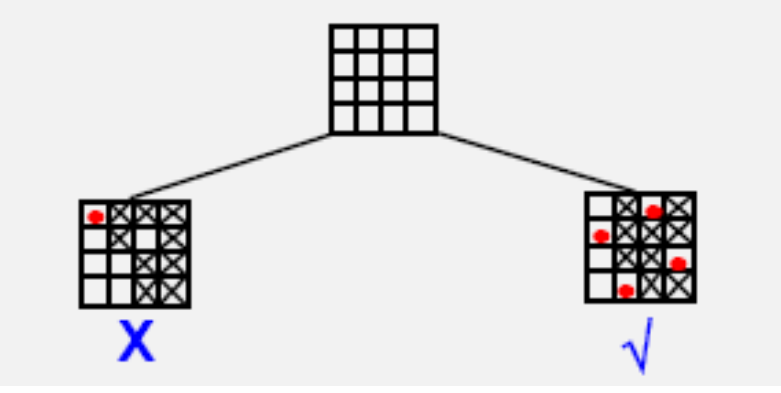
Algoritme Partial look ahead

algorisme dAC-LA(G, va)
  per $i := va + 1$ **fins** n **fes**
    percada arc $(X_i, X_j) \in arcs(G)$ tals que $i > j \wedge j \geq va$ **fes**
      si REVISA(X_i, X_j) **llavors**
        si esbuit(D_i) **llavors retorna** FAIL
    fper
  fper
  retorna CERT
falgorisme

- Per mantenir Directed Arc-consistency (DAC), suposem l'ordre invers al de l'etiquetatge.
- No cal mirar consitència amb variables passades...

Partial look-ahead ii

El problema de les 4 reines que no s'amenacen amb partial look-ahead, l'arbre de cerca:



Full look Ahead

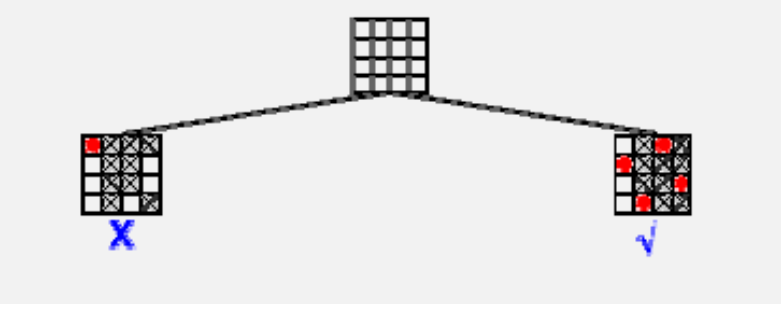
Algoritme AC-3-Look Ahead

algorisme AC-3-LA(G, va)
  $Q := \{(X_i, X_{va}) \in arcs(G), i > va\}$
  $CONSISTENT := TRUE$
  mentre no buit $Q \wedge$ $CONSISTENT$ **fes**
    Treu algun arc $(X_k, X_m) \in Q$
    si REVISA(X_k, X_m) **llavors**
      $Q := Q \cup \{(X_i, X_k) \in$ arcs(G), $i \neq k, i \neq m, i > va\}$
      $CONSISTENT :=$ no buit D_k
    fsi
  fmentre
  retorna $CONSISTENT$
falgorisme

- Els arcs que van a la variable actual es controlen només un cop
- Els arcs cap a variables passades no es checkegen

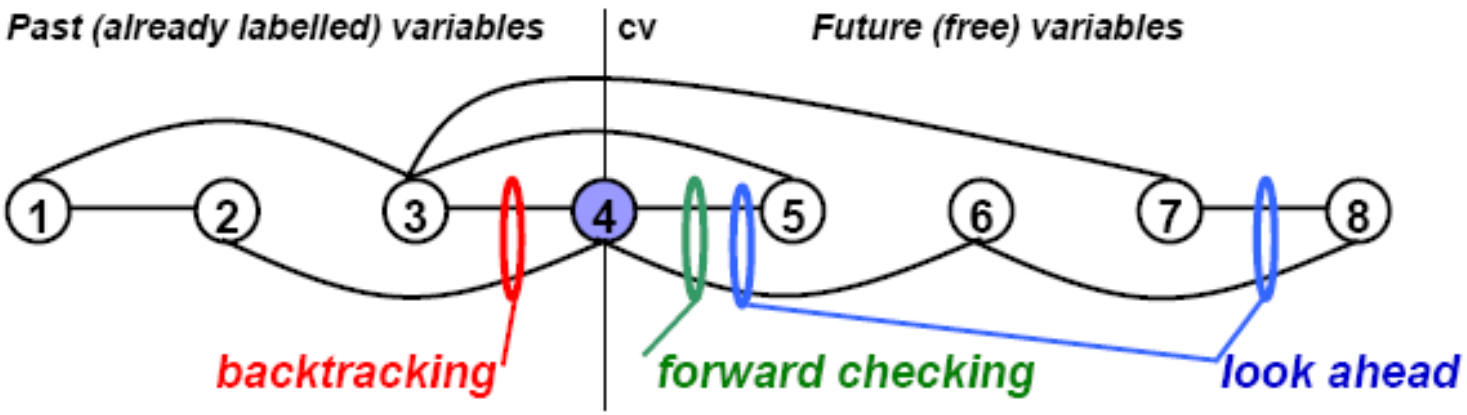
Full look-ahead ii

El problema de les 4 reines que no s'amenacen amb partial look-ahead, l'arbre de cerca:



Comparativa

Les variables que s'estudien segons mètodes:



Optimització de restriccions

Problema d'optimització de restriccions (COP)

Un **Problema d'optimització de restriccions** és un CSP amb una funció objectiu f tal que f associa un valor a cada solució. L'objectiu és trobar una solució que sigui òptima en quant a la funció f (maximitzi o minimitzi).

La tècnica del **Branch & Bound** permet podar part de l'arbre de cerca segons una funció heurística que estima la funció objectiu per assignacions parcials, de manera que **si l'estimació ens permet veure que el camí actual no optimitza, llavors el podem tallar**

- bona heurística per poder podar quan més millor
- compte, sense passar-se!
- no sempre és fàcil trobar bones estimacions...

Branch and Bound

BOUND que te la cota superior actual (suposem que minimitzem) i *BEST* que és la millor solució fins ara. Crida $\texttt{BB}(X, \{\}, C)$.

algorisme $\texttt{BB}(X\texttt{:vars},\ A\texttt{:assignacio},\ C\texttt{:constraints})$
  si $X == \{\}$ **llavors**
    si $f(A) < BOUND$ **llavors**
      $BOUND := f(A); \quad BEST := A$
    fsi
  sino
  $x :=$ `tria una variable d'`X
  percada $v \in D_x$ **fes**
    si $\texttt{const}(A++(x,v), C) \ \wedge\ h(A++(x,v)) < BOUND$ **llavors**
      $\texttt{BB}(X - \{x\}, A++(x,v), C)$
    fsi
  fper
falgorisme

Resum

Un exemple

Donada una carta on hi ha primers, segons i postres, juntament amb les seves calories, trobar un menú lleuger que no tingui un valor energètic superior a 10

```
pr(canalons,5).   pr(amanida,1). pr(esparrecs,2).
sg(parrillada,6). sg(paella,5).  sg(dorada,3).
po(fruita,2).     po(flam,3).    po(pastis,4).
```

Algorisme amb PL: generate and test

```
menulight(P,S,D):- pr(P,I), sg(S,J), po(D,K),
                   T is I+J+K,
                   T < 10.
```

Un exemple

Donada una carta on hi ha primers, segons i postres, juntament amb les seves calories, trobar un menú lleuger que no tingui un valor energètic superior a 10

```
pr(canalons,5).    pr(amanida,1). pr(esparrecs,2).
sg(parrillada,6). sg(paella,5).   sg(dorada,3).
po(fruita,2).      po(flam,3).     po(pastis,4).
```

Algorisme amb PL: generate and test

```
menulight(P,S,D):- pr(P,I), sg(S,J), po(D,K),
                   T is I+J+K,
                   T < 10.
```

Algorisme amb CLP: test and generate

```
menulight(P,S,D):- I#>0, J#>0, K#>0,
                   I+J+K #=< 10,
                   pr(P,I), sg(S,J), po(D,K).
```

Un esquema algorísmic senzill

Del *"generate and test"* al *"constraint and generate"*:

L'esquema bàsics dels programes CLP

```
main(Vars):-
            definir_dominis(Vars),
            imposar_restriccions(Vars),
            buscar_valors(Vars).
```

Canvis al motor de càlcul

Actualització del model de còmput

- En Prolog, la única *restricció* que hi ha és la **unificació**.
- S'ha de modificar el *motor de càlcul* per a fer lloc per als **constraints**.
  - Afegir als objectius actuals de la resolució, les restriccions trobades fins ara,
  - garantir-ne la consistència a cada pas.
- Hi ha diferents implementacions, nosaltres considerarem la de **SICSTUS** basat en l'esquema $CLP(\mathcal{X})$ i més concretament amb $CLP(\mathcal{FD})$ [vH'92].

Un model senzill per a la semàntica operacional

- Representarem la noció d'estat del còmput actual mitjançant una tupla on hi haurà:
  - Els objectius pendents fins ara. Les clàsules que s'han de demostrar, tal com fèiem amb Prolog.
  - Les restriccions que s'han enviat fins al moment i queden per satisfer.

$$\langle\ O_1, \ldots, O_n,\ C_1, \ldots C_m\ \rangle$$

- ara les "branques" poden fallar degut a que no es pot avançar en resolució o a que s'ha trobat que els constraints són inconsistents.

Un exemple (i)

Donada una carta on hi ha primers, segons i postres, juntament amb les seves calories, trobar un menú lleuger que no tingui un valor energètic superior a 10

Algorisme amb CLP

```
menulight(P,S,D):- I#>0, J#>0, K#>0,
                   I+J+K #=< 10,
                   pr(P,I), sg(S,J), po(D,K).

pr(canalons,5). pr(amanida,1). pr(esparrecs,2).
sg(parrillada,6). sg(paella,5). sg(dorada,3).
po(fruita,2). po(flam,3). po(pastis,4).
```

Un exemple (ii), la traça:

1. Just abans de: `pr(P,I),...`

$$\langle \mathtt{pr}(P, I), \mathtt{sg}(S, J), \mathtt{po}(D, K),\ I > 0, J > 0, K > 0, I{+}J{+}K <= 10 \rangle$$

Un exemple (ii), la traça:

1. Just abans de: `pr(P,I),...`

   $\langle \texttt{pr}(P, I), \texttt{sg}(S, J), \texttt{po}(D, K),\ I > 0, J > 0, K > 0, I+J+K <= 10 \rangle$

2. resolem $\texttt{pr}(P, I)$ amb `pr(canalons,5)`

   $\langle\ \texttt{sg}(S, J), \texttt{po}(D, K),\ 5 > 0, J > 0, K > 0, 5 + J + K <= 10 \rangle$

Un exemple (ii), la traça:

1. Just abans de: `pr(P,I),...`

   $\langle \texttt{pr}(P, I), \texttt{sg}(S, J), \texttt{po}(D, K),\ I > 0, J > 0, K > 0, I+J+K <= 10 \rangle$

2. resolem $\texttt{pr}(P, I)$ amb `pr(canalons,5)`

   $\langle\ \texttt{sg}(S, J), \texttt{po}(D, K),\ 5 > 0, J > 0, K > 0, 5 + J + K <= 10 \rangle$

3. resolem $\texttt{sg}(S, J)$ amb `sg(parrillada,6)` i fallem

   $\langle\ \texttt{po}(D, K),\ 6 > 0, K > 0, 5 + 6 + K <= 10 \rangle$

Un exemple (ii), la traça:

1. Just abans de: `pr(P,I),...`

   $\langle \texttt{pr}(P,I), \texttt{sg}(S,J), \texttt{po}(D,K),\ I > 0, J > 0, K > 0, I+J+K <= 10 \rangle$
2. resolem $\texttt{pr}(P,I)$ amb `pr(canalons,5)`

   $\langle\ \texttt{sg}(S,J), \texttt{po}(D,K),\ 5 > 0, J > 0, K > 0, 5+J+K <= 10 \rangle$
3. resolem $\texttt{sg}(S,J)$ amb `sg(parrillada,6)` i fallem

   $\langle\ \texttt{po}(D,K),\ 6 > 0, K > 0, 5+6+K <= 10 \rangle$
4. resolem $\texttt{sg}(S,J)$ amb `sg(paella,5)` i fallem

   $\langle\ \texttt{po}(D,K),\ 5 > 0, K > 0, 5+5+K <= 10 \rangle$

Un exemple (ii), la traça:

1. Just abans de: `pr(P,I),...`

   $\langle \texttt{pr}(P,I), \texttt{sg}(S,J), \texttt{po}(D,K),\ I > 0, J > 0, K > 0, I+J+K <= 10 \rangle$
2. resolem $\texttt{pr}(P,I)$ amb `pr(canalons,5)`

   $\langle\ \texttt{sg}(S,J), \texttt{po}(D,K),\ 5 > 0, J > 0, K > 0, 5+J+K <= 10 \rangle$
3. resolem $\texttt{sg}(S,J)$ amb `sg(parrillada,6)` i fallem

   $\langle\ \texttt{po}(D,K),\ 6 > 0, K > 0, 5+6+K <= 10 \rangle$
4. resolem $\texttt{sg}(S,J)$ amb `sg(paella,5)` i fallem

   $\langle\ \texttt{po}(D,K),\ 5 > 0, K > 0, 5+5+K <= 10 \rangle$
5. resolem $\texttt{sg}(S,J)$ amb `sg(dorada,3)`

   $\langle\ \texttt{po}(D,K),\ 3 > 0, K > 0, 5+3+K <= 10 \rangle$

Un exemple (ii), la traça:

1. Just abans de: `pr(P,I),...`

   $\langle \texttt{pr}(P,I), \texttt{sg}(S,J), \texttt{po}(D,K),\ I > 0, J > 0, K > 0, I+J+K <= 10 \rangle$
2. resolem $\texttt{pr}(P,I)$ amb `pr(canalons,5)`

   $\langle\ \texttt{sg}(S,J), \texttt{po}(D,K),\ 5 > 0, J > 0, K > 0, 5+J+K <= 10 \rangle$
3. resolem $\texttt{sg}(S,J)$ amb `sg(parrillada,6)` i fallem

   $\langle\ \texttt{po}(D,K),\ 6 > 0, K > 0, 5+6+K <= 10 \rangle$
4. resolem $\texttt{sg}(S,J)$ amb `sg(paella,5)` i fallem

   $\langle\ \texttt{po}(D,K),\ 5 > 0, K > 0, 5+5+K <= 10 \rangle$
5. resolem $\texttt{sg}(S,J)$ amb `sg(dorada,3)`

   $\langle\ \texttt{po}(D,K),\ 3 > 0, K > 0, 5+3+K <= 10 \rangle$
6. resolem $\texttt{po}(D,K)$ amb `po(fruita,2)`

   $\langle\ \emptyset,\ 2 > 0, 10 <= 10 \rangle \Rightarrow \langle\ \emptyset,\ \emptyset\ \rangle$

La implementació de SICSTUS

Basat en `library(clpfd)` (implementada amb C) que carrega el gestor de restriccions així com unes quantes restriccions globals ja definides.

- El resoledor és bàsicament un planificador per a
  - **indexicals**: regles per a propagar restriccions i resoldre-les o detectar "entailment",
  - **constraints globals**: propagadors generals que tenen algorismes especialitzats.

  els constraints globals només "s'ataquen" quan no queden indexicals.
- combina l'aproximació "**black box**" amb constraints globals ja implementats i "**glass box**" permetent definir-ne a l'usuari i/o controlar opcions.

Built-in Constraints i Cerca

Constraints:

- De domini: `X in D`, `X in_set C`, `domain(Lvars,Inf,Sup)`
- Aritmètics: `X #= Y+2`, `X #\= Y`, `X #=<Y`, ...
- Combinatorics: `all_different`, `element`, `circuit`, `cumulative`, `disjoint1`, `count`, `sum`, `knapsack`, ...
- Lògics:

  `C #/\ D`, `C #=> D`,...
- Reified:

  `C #<=> B` ...

Cerca:

- Predicats: `labeling`, `minimize`, `maximize`, `indomain`
- Opcions: `leftmost`, `min`, `max`, `ff`, `ffc`, ...

Primer tast d'indexicals i consistències d'intèrval

Considerem la següent restricció:

```
A in 2000..5000,  B in 1000..4000, A#=< B.
```

Representació de dominis:

- dominis només decreixen!
- Bit vectors necessiten temps lineal per detectar inconsistència (o bit counter)...
- Representació *intensional*: límit inferior i superior (ok si no hi ha forats).

Primer tast d'indexicals i consistències d'interval

Considerem la següent restricció:

```
A in 2000..5000,  B in 1000..4000, A#=< B.
```

Codificació de la consistència dels dominis:

- per l'exemple concret `A#=< B` tindrem les regles (**indexicals**):
  1. el màxim del domini de `A` no ha de ser més gran que el màxim del domini de `B`,
     Domini d'`A = 2000 .. 5000` $\cap$ `inf .. max(B)` o sigui,
     `max(A) = min(5000, max(B)) = max(B)`
  2. el mínim del domini de `B` no ha de ser més petit que el mínim del domini de `A`
     Domini de `B = 1000 .. 4000` $\cap$ `min(A) .. sup` o sigui,
     `min(B) = max(1000, min(A)) = min(A)`
- indexicals **monòtons** mantenint **consistència d'interval**:
  `A in 2000 .. 4000` $\cap$ `inf .. max(B)`
  `B in 2000 .. 4000` $\cap$ `min(A) .. sup`

Primer tast d'indexicals i consistències d'intèrval

Considerem la següent restricció:

```
A in 2000..5000,  B in 1000..4000, A#=< B.
```

Mantenir la consistència (propagació):

- Quan s'"envia" el constraint ja es modifiquen els dominis:
  `A in 2000 .. 4000` $\cap$ `inf .. max(B)`
  `B in 2000 .. 4000` $\cap$ `min(A) .. sup`
- aquest dominis queden depenents uns dels altres... Quan el limit superior de `B` decreix, s'actualitza el domini d'`A` i es comprova la consistència. (Similarment per `B` i el mínim d'`A`.)
- Els canvis en el límit superior d'`A` o en el límit inferior de `B` no es propaguen...

En general però, els indexicals ens permetran definir altres tipus de consistència quan els dominis tinguin "forats".

Primer tast d'indexicals i consistències d'intèrval (ii)

Considerem la següent restricció:

```
A in 1..9,  B in 3..3, A#\= B.
```

Tindrem la següent traducció a indexicals:

`A in 1 .. 9` $\cap \setminus$ `dom(B)`

`B in 3 .. 5` $\cap \setminus$ `dom(A)`

que no són monòtons: quan el domini d'una variable decreix, creix el de l'altre!!!

Així, les operacions de complementari: $\setminus$ es congelen fins que el que es complementa sigui un singleton: **Node consistència**.

```
?- A in 1..9,  B in 3..3, A#\= B, B#=3, fd_dom(B,S).
        B = 3,
        S = {3},
        A in(1..2)\/(4..9) ?
```

indexicals i consistències

Indexicals

Un **indexical** (*projection constraint*) per a resoldre un constraint $C(X_1, \ldots, X_n)$, te la forma X_i *in* R_i, on R_i és el *rang* i especifica els possibles possibles valors de X_i en funció dels dels altres X_j's. Expressem el constraint C mitjançant n indexicals, un per cada X_i.

- Un **store** S és un conjunt d'especificacions de dominis per a variables: $X \in D$ on D és un conjunt finit.
- Un constraint C és **domini consistent** respecte S sii per cada variable i valor, existeixen altres valors per a les altres variables que satisfan C.
- Un constraint C és **domain entailed** respecte S sii per a qualsevol valor de qualsevol variable es satisfà C.
- **intèrval consistent** i **intèrval entailed** és equivalent però considerant els intèrvals definits pels valors mínims i màxims dels dominis de les variables.

Definir nous constraints mitjançant indexicals

El constraint $X \sharp = Y + C$ (C és una constant)

Versió *Domain consistent* i versió *Interval consistent*

```
eqcd(X,Y,C) +:
   X in dom(Y)+C,
   Y in dom(X)-C.
```

```
eqcd(X,Y,C) +:
   X in min(Y)+C..max(Y)+C,
   Y in min(X)-C..max(X)-C.
```

- cada indexical és una regla encarregada d'eliminar valors incompatibles d'un argument del constraint (X o Y).
- No són objectius Prolog.
- Un indexical pot fer un estat insatisfactible pels dominis (domini que queda buit) i provocar un backtracking.
- Es carrega quan s'invoca el constraint, s'invoca per al filtratge quan es toquen els dominis o els intèrvals dels rangs i desapareix quan ja no es poden variar els rangs i se'n determina la consistència.

Gramàtica dels indexicals

Consulteu el manuals del SICSTUS.

Expressions de Rang:

```
R ::= T..T | R/\R | R\/R | \R | R+T | R-T | R mod T
    | {T,...,T} | dom(X)
```

Termes:

```
T ::= T+T | T-T | T*T | T/>T | T</T | T mod T
   | min(X) | max(X) | card(X) | X | N

N ::= integer | inf | sup
```

Implementació SICSTUS dels indexicals

El compilador de SICSTUS s'encarrega de les definicions amb indexicals amb: `user:term_expansion/2`:

```
user:term_expansion((Head+:Body), Expansion) :-
    functor(Head, N, A),
    Expansion = [:- clpfd:'$fd_install'(N/A, 1, Info)],
    compile(Head, Body, Info).
```

se n'encarrega una Màquina Virtual.

Cost de la consistència d'intèrval i arc-consistència

Consistència d'intèrval:

```
plusi(X,Y,T) +:
   X in min(T)-max(Y)..max(T)-min(Y),
   Y in min(T)-max(X)..max(T)-min(X),
   T in min(X)+min(Y)..max(X)+max(Y).
?- X in 1..5, Y in 2..8, plusi(X,Y,T).
X in 1..5, Y in 2..8,T in 3..13
```

L'arc consistència és més costosa:

```
plusd(X,Y,T) +:
   X in dom(T)-dom(Y),
   Y in dom(T)-dom(X),
   T in dom(X)+dom(Y).
?- X in {1}\/{3}, Y in {10}\/{20}, plusd(X,Y,T).
X in {1}\/{3}, Y in {10}\/{20}, T in {11}\/{13}\/{21}\/{23
```

Exemple/Exercici (i)

Definiu indexicals pel constraint: `ajusta(X,Z,K,N)` tal que per `ajusta(X,Z,5,10)` tenim:

Z	1	2	3	4	5	6	7	8	9	10
X	6	7	8	9	10	1	2	3	4	5

Així `X = (Z+K+N-1) mod N + 1` i `Z = (X-K+N-1) mod N + 1`).
Podem expressar el constraint via indexicals... com?

Exemple/Exercici (i)

Definiu indexicals pel constraint: `ajusta(X,Z,K,N)` tal que per `ajusta(X,Z,5,10)` tenim:

Z	1	2	3	4	5	6	7	8	9	10
X	6	7	8	9	10	1	2	3	4	5

Així `X = (Z+K+N-1) mod N + 1` i `Z = (X-K+N-1) mod N + 1`).
Podem expressar el constraint via indexicals... com?
Definició:

```
ajusta(X,Z,K,N)+:
    X in ((dom(Z)+K+N-1) mod N) + 1,
    Z in ((dom(X)-K+N-1) mod N) + 1.
```

Encara es pot fer millor...

Exemple/Exercici (i)

Definiu indexicals pel constraint: `ajusta(X,Z,K,N)` tal que per `ajusta(X,Z,5,10)` tenim:

Z	1	2	3	4	5	6	7	8	9	10
X	6	7	8	9	10	1	2	3	4	5

Així `X = (Z+K+N-1) mod N + 1` i `Z = (X-K+N-1) mod N + 1`).
Podem expressar el constraint via indexicals... com?
Definició:

```
ajusta(X,Z,K,N)+:
    X in (dom(Z) + K) \/ (dom(Z) + K - N),
    Z in (dom(X) - K) \/ (dom(X) - K + N).
```

Exemple/Exercici (ii)

Definiu indexicals pel constraint: `no_attack(L1,C1,L2,C2)` tal que dues reines posades a les files `L1` i `L2` (constants) si les posem a les columnes `C1` i `C2` respectivament (variables), no s'ataquin...

Exemple/Exercici (ii)

Definiu indexicals pel constraint: `no_attack(L1,C1,L2,C2)` tal que dues reines posades a les files `L1` i `L2` (constants) si les posem a les columnes `C1` i `C2` respectivament (variables), no s'ataquin... Sabem que

```
C1 \= C2,  L1+C1 \= L2+C2  , L1+C1 \= L2+C2
```

Per tant podem definir:

```
no_attack(L1,C1,L2,C2)+:
  C1 in \({C2}\/{C2+L2-L1}\/{C2+L1-L2}),
  C2 in \({C1}\/{C1+L1-L2}\/{C1+L2-L1}).
```

Fixem-nos que les variables apareixen en "negatiu", per tant el constraint es congela fins que sigui singleton... Ara bé, tenim **consistència de node**. Podem fer-ho amb arc-consistència?

Exemple/Exercici (ii)

Definiu indexicals pel constraint: `no_attack(L1,C1,L2,C2)` tal que dues reines posades a les files `L1` i `L2` (constants) si les posem a les columnes `C1` i `C2` respectivament (variables), no s'ataquin... Podem definir:

```
no_attack(L1,C1,L2,C2)+:
 C1 in unionof(Y,dom(C2),\({Y}\/{Y+L2-L1}\/{Y+L1-L2})),
 C2 in unionof(X,dom(C1),\({X}\/{X+L1-L2}\/{X+L2-L1})).
```

Obtenim **arc-consistència** permetent a una reina situar-se en una posició que no és atacada per com a mínim una possible posició de l'altra reina.

Exemple/Exercici (ii)

Definiu indexicals pel constraint: `no_attack(L1,C1,L2,C2)` tal que dues reines posades a les files `L1` i `L2` (constants) si les posem a les columnes `C1` i `C2` respectivament (variables), no s'ataquin...
Podem millorar-ho ... sempre que `C2` pugui posar la reina a 4 llocs les posicions de `C1` són possibles en respecte `C2` ... per tant podem indicar que només s'avaluï l'arc-consistència quan la cardinalitat sigui menor que 4.
Podem definir:

```
no_attack(L1,C1,L2,C2)+:
 C1 in (4..card(C2)) ? (inf..sup) \/
    unionof(Y,dom(C2),\({Y}\/{Y+L2-L1}\/{Y+L1-L2})),
 C2 in (4..card(C1)) ? (inf..sup) \/
    unionof(X,dom(C1),\({X}\/{X+L1-L2}\/{X+L2-L1})).
```

Reificació

Què és reificar?

Significa "cosificar". En el nostre cas serà per associar un valor de veritat a la satisfacció d'un constraint.Compte, no tots els constraints són reificables!.

En SICSTUS:

```
?- domain([X,Y],1,2), X#>Y #<=>B.
```

Comptar occurrències d'X a una llista de FD vars

```
exactly(_, [], 0).
exactly(X, [Y|L],N):-
   X #= Y #<=> B,
   N #= M+B,
   exactly(X,L,M).
```

Reificació amb indexicals

X = Y + C

```
?- eqcd(X,Y,5) <=> B.
-----------------------------
eqcd(X,Y,C) +:              %constraint solving positiu
    X in dom(Y)+C,
    Y in dom(X)-C.

eqcd(X,Y,C) -:              %constraint solving negatiu
    X in \{Y+C},
    Y in \{X-C}.

eqcd(X,Y,C) +?              %deteccio entailment
    X in {Y+C}.

eqcd(X,Y,C) -?              %deteccio disentailment
    X in \dom(Y)+C.
```

Reificació i disjuncions

El problema d'evitar solapament entre dues tasques amb inicis `I1`, `I2` i durada `D1` i `D2` es pot formular:

```
disjoint(I1,I2,D1,D2) :- I1 + D1 #=< S2.
disjoint(I1,I2,D1,D2) :- I2 + D2 #=< S1.
```

Backtracking... cerca exponencial!
Alternativa:

```
disjoint(I1,I2,D1,D2) :-
    (I1 + D1 #=< I2) #<=> B1,
    (I2 + D2 #=< I1) #<=> B2,
    B1 + B2 #>= 1.
```

Exemple/Exercici Reificació

Una llista `S=[S0,S1,...Sn]` és **màgica** sii hi ha exactament `Si` ocurrències de `i` a la seqüència `S` per a cada enter de 0 a n.
Per exemple:

```
n = 3: [1,2,1,0]   i [2,0,2,0]
n = 4: [2,1,2,0,0]
...
```

Defineix el predicat `magica(L,N)` que per una `N` donada ens doni Ls màgiques (si es que en te...)

Consistència d'intèrval i de domini

Els constraints bàsics també estan implementats mantenint diferent tipus de consistència...
Vegem el següent exemple:

```
?- X+Y#=Z, X=1,Z=6, Y in 1..10, Y#\=5.
no

?- X+Y#=Z #<=> B, X=1,Z=6, Y in 1..10, Y#\=5.
X=1,
Z=6,
Y in (1..4)\/(6..10),
B in 0..1.
```

En general, els dominis d'igualtat, comparacions d'ordre així com els constraints aritmètics mantenen consistència d'intèrval. El de desigualtat, de domini. Com que `1+5#=6` és cert, `X+Y#=Z` no queda no és "intèrval-contradit".

Arquitectura del SICSTUS: constraints globals

Alguns Built-in global constraints:

- `all_different(L)`: implementació eficient a base d'algorismes sobre grafs [Regi94].
- `element(I,L,Y)`: l'Ièssim element d'L ha de ser Y. Basat en AC-4.
- `cumulative(S,D,R,L)`: Modela tasques començant en temps S, durada S i requerint D recursos d'un màxim d'L. Basats en mètodes OR.
- ...

De **constraints globals** també s'en poden programar de nous per l'usuari, permetent controlar canvis de domini de variables i requeriment de filtratge de dominis. Vegeu capítol 10.34.8 del manual.

SICSTUS

Consideracions Finals

- És bo conèixer el model de còmput actualitzat amb els constraints per a fer-se una idea de la semàntica operacional del SICSTUS amb constraints.
- Habitualment en tindrem prou utilitzant els predicats "Built-in".
- Per si se'n necessita més: manual de SICSTUS, capítol CLP(FD).

Resum

Què és Modelar un problema de Constraints?

En general podem dir que un **model** $M = \langle X, D, C \rangle$, en CLP consta de 3 parts:

- X, un conjunt de variables,
- D, un domini per a cada variable,
- C, un conjunt de restriccions a satisfer.

Què és Modelar un problema de Constraints?

En general podem dir que un **model** $M = \langle X, D, C \rangle$, en CLP consta de 3 parts:

- X, un conjunt de variables,
- D, un domini per a cada variable,
- C, un conjunt de restriccions a satisfer.

Donat un problema P, diem que M **representa el problema** P, o M **és un model de** P, si

- qualsevol solució de C correspon a una solució de P i
- qualsevol solució de P es pot derivar de com a mínim una solució de C.

Què és Modelar un problema de Constraints?

En general podem dir que un **model** $M = \langle X, D, C \rangle$, en CLP consta de 3 parts:

- X, un conjunt de variables,
- D, un domini per a cada variable,
- C, un conjunt de restriccions a satisfer.

Donat un problema P, diem que M **representa el problema** P, o M **és un model de** P, si

- qualsevol solució de C correspon a una solució de P i
- qualsevol solució de P es pot derivar de com a mínim una solució de C.

Compte, les simetries ho compliquen tot...

Propagació i cerca

Assumim que estem en un mecanisme de **cerca i propagació**.

- la cerca es fa a partir de *choice points* tal que cada "branca" aposta per una solució exclusiva i el conjunt de branques és exhaustiu.
- Un cop triat una branca, s'actualitza el conjunt de restriccions actuals i es propaga...
- Si es troba alguna inconsistència, es fa *backtrack* fins a la següent opció del choice point.

Propagació i cerca

Assumim que estem en un mecanisme de **cerca i propagació**.

- la cerca es fa a partir de *choice points* tal que cada "branca" aposta per una solució exclusiva i el conjunt de branques és exhaustiu.
- Un cop triat una branca, s'actualitza el conjunt de restriccions actuals i es propaga...
- Si es troba alguna inconsistència, es fa *backtrack* fins a la següent opció del choice point.

El tipus de consistència que s'utilitza, així com l'estratègia de cerca poden influir notablement en el rendiment, per tant condicionen la modelització!!!

Perspectiva (viewpoint) (i)

Un **viewpoint** és una tupla de variables i dominis $\langle X, D \rangle$.

- Per a un problema P, s'ha de poder associar un significat a cada assignació de valors del domini a les variables.
- Afegint un conjunt de restriccions C d'acord amb el problema P, ha de filtrar les assignacions vàlides.
- A partir de viewpoints diferents podem fer models diferents.

Perspectiva (viewpoint) (i)

Un **viewpoint** és una tupla de variables i dominis $\langle X, D \rangle$.

- Per a un problema P, s'ha de poder associar un significat a cada assignació de valors del domini a les variables.
- Afegint un conjunt de restriccions C d'acord amb el problema P, ha de filtrar les assignacions vàlides.
- A partir de viewpoints diferents podem fer models diferents.

Típicament el millor viewpoint ha de permetre expressar els constraints fàcilment i concisa!!!

Perspectiva (viewpoint) (i)

Un **viewpoint** és una tupla de variables i dominis $\langle X, D \rangle$.

- Per a un problema P, s'ha de poder associar un significat a cada assignació de valors del domini a les variables.
- Afegint un conjunt de restriccions C d'acord amb el problema P, ha de filtrar les assignacions vàlides.
- A partir de viewpoints diferents podem fer models diferents.

Típicament el millor viewpoint ha de permetre expressar els constraints fàcilment i concisa!!!

Un bon viewpoint per les n reines?

Per les n-reines definim n variables amb domini $\{1..n^2\}$. Una assignació (q_i, a) significa que la reina i està a la casella a.

Perspectiva (viewpoint) (ii)

Example

El quadrat màgic Posar els números de l'1..9 en un quadrat de 3×3 tal que les sumes de cada fila, cada columna i les dos diagonals sumin el mateix.

4	3	8
9	5	1
2	7	6

Perspectiva (viewpoint) (ii)

Example

El quadrat màgic Posar els números de l'1..9 en un quadrat de 3×3 tal que les sumes de cada fila, cada columna i les dos diagonals sumin el mateix.

Dos viewpoints, quin és millor?

1. Una variable per cada casella amb el domini d'1..9 indicant els valors que hi poden anar.
2. Una variable per cada número amb el domini d'1..9 indicant a la casella on va.

La versió 1: $X_1 + X_2 + X_3 = X_4 + X_5 + X_6 = ... = X_1 + X_4 + X_7 = ... = X_1 + X_5 + X_9 = ...$
Amb la versió 2...

Restriccions (i)

Un cop tenim el viewpoint, com escrivim els constraints?:

- $x = y$, $x + y = z$
- $x = y$, $2y = z$ com el de dalt però sap que z és parell

Restriccions (i)

Un cop tenim el viewpoint, com escrivim els constraints?:

- $x = y, \quad x + y = z$
- $x = y, \quad 2y = z$ com el de dalt però sap que z és parell

Típicament caldrà saber coses com:

- els constraints que ens dóna el solver
- quina consistència força cada constraint
- quina complexitat tenen els propagadors
- ...

Compromís entre el temps gastat en propagar i el temps estalviat en cercar...

Restriccions (ii)

- **Combinar constraints** per millorar la propagació. (cas reines i diferents tipus d'amenaça)
- **Reescriure constraints** eliminant variables. (cas eq. lineals)
- Us de **constraints globals** amb diferent tipus de consistència depenent de la naturalesa del problema. Han de **reduir cerca**, **poden reduir (o incrementar) runtime!**. (`alldifferent` si difícil)
- **Reificació**. (per disjuncions)
- **Definicions extensionals**:

Restriccions (ii)

- **Combinar constraints** per millorar la propagació. (cas reines i diferents tipus d'amenaça)
- **Reescriure constraints** eliminant variables. (cas eq. lineals)
- Us de **constraints globals** amb diferent tipus de consistència depenent de la naturalesa del problema. Han de **reduir cerca**, **poden reduir (o incrementar) runtime!**. (`alldifferent` si difícil)
- **Reificació**. (per disjuncions)
- **Definicions extensionals**:

El Black Hole Solitarie

Ordenar una seqüència de cartes de manera que cada carta estigui seguida d'una amb el número següent o amb el número anterior.

Restriccions (ii)

- **Combinar constraints** per millorar la propagació. (cas reines i diferents tipus d'amenaça)
- **Reescriure constraints** eliminant variables. (cas eq. lineals)
- Us de **constraints globals** amb diferent tipus de consistència depenent de la naturalesa del problema. Han de **reduir cerca**, **poden reduir (o incrementar) runtime!**. (`alldifferent` si difícil)
- **Reificació**. (per disjuncions)
- **Definicions extensionals**:

El Black Hole Solitarie

Ordenar una seqüència de cartes de manera que cada carta estigui seguida d'una amb el número següent o amb el número anterior.
Tenim 52 variables amb un domini de l'1 al 52.
De l'1 al 13 són de diamants, del 14 al 26 són de cors, etc.
Representar els constraints extensionalment: si x_i se li assigna el 15 (2 de cors) restringim el domini d'x_i a $\{1, 3, 14, 16, 27, 29, 40, 42\}$.

Variables Auxiliars (i)

Afegir variables a un model sovint permet expressar les restriccions més fàcilment i/o permetre més propagació obtenint extensions de models més eficients!

Variables Auxiliars (i)

Car Sequencing Problem

- S'han de fer una sèrie de cotxes en una línia de producció.
- Cada cotxe te unes opcions que s'han d'instal·lar en estacions especials amb menys capacitat que la línia de producció (p.ex: una estació podrà instal·lar un extra mentre surten dos cotxes de la línia.)
- **S'ha de programar l'ordre de sortida dels cotxes de manera que les capacitats de les opcions no es sobrepassin.**

Classes	1	2	3	4	5	6	Capacitat
Opció 1	1	0	0	0	1	1	1/2
Opció 2	0	0	1	1	0	1	2/3
Opció 3	1	0	0	0	1	0	1/3
Opció 4	1	1	0	1	0	0	2/5
Opció 5	0	0	1	0	0	0	1/5
N. Cotxes	1	1	2	2	2	2	

Variables Auxiliars (ii)

Car Sequencing Problem

Model:

- Variables: $x_1, \ldots x_{10}$
- Dominis: $\{1, \ldots, 6\}$ classe de cada cotxe per posició de la seq.
- Restriccions:
  - Cada classe apareix el número correcte de vegades. (fàcil)
  - Respectem les capacitats de les opcions...(???)

Variables Auxiliars (ii)

Car Sequencing Problem

Model:

- Variables: $x_1, \ldots x_{10}$
- Dominis: $\{1, \ldots, 6\}$ classe de cada cotxe per posició de la seq.
- Restriccions:
  - Cada classe apareix el número correcte de vegades. (fàcil)
  - Respectem les capacitats de les opcions...(???)

Idea: Afegir variables auxiliars $o_{i,j}$!!

- $o_{i,j} = 1$ sii el cotxe a la posició i de la seq. vol l'opció j.
- L'opció 1 té una capacitat 1/2, és a dir, un cotxe cada dos:

$$\forall_{1 \leq i < 10} \quad o_{i,1} + o_{i+1,1} \leq 1$$

- Relacionem les variables auxiliars amb les x_i:
  - $l_{j,k} = 1$ si la classe k requereix l'opció j
  - $o_{i,j} = l_{j,x_i}, 1 \leq i \leq 10, 1 \leq j \leq 5$

Restriccions Implicades (i)

- Les **restriccions implicades** són conseqüències lògiques del conjunt de restriccions existents. (restriccions redundants)
- No canvien el conjunt de solucions
- Han de servir per reduir l'espai de cerca $\Rightarrow$ reduir run-time...

Restriccions Implicades (ii)

Car Sequencing

- Sabem i restringim que no s'excedeixin les capacitats de les opcions.
- Suposem que hi ha 30 cotxes i 12 volen l'opció 1 (capacitat 1/2), anem a afegir restriccions implicades:
  - Com a mínim un cotxe dels de la posició 1 a la 8 ha de requerir l'opció 1, altrament 12 cotxes del 9 al 30 ho requeriran i "no hi caben" !!
  - De l'1 al 10 hi ha d'haver 2 cotxes amb l'opció 1,... i de l'1 al 28 ha de tenir 11 cotxes amb opció 1..

Restriccions Implicades (ii)

Són útils quan redueixen la cerca:

- En algun moment de la cerca, una assignació parcial (que fallaria més endavant) fallarà degut a un constraint implicat.
- Sense el constraint implicat la cerca hagués seguit.

Car Sequencing

Les restriccions afegides controlen la infra-utilització d'estacions per opcions per tal d'evitar manca de recursos més endavant.

Restriccions Implicades (ii)

Són útils quan redueixen la cerca:

- En algun moment de la cerca, una assignació parcial (que fallaria més endavant) fallarà degut a un constraint implicat.
- Sense el constraint implicat la cerca hagués seguit.

Car Sequencing

Les restriccions afegides controlen la infra-utilització d'estacions per opcions per tal d'evitar manca de recursos més endavant.

Quan no ho són?

- Si l'espai de cerca no es redueix... Per exemple depenent de l'ordre d'exploració de les solucions...
- Si afegim diverses implied constraints i realment només algunes redueixen l'espai de cerca i les altes afegeixen sobrecàrrega...

Restriccions Implicades (iii)

- De vegades es poden fer **constraints globals** ad-hoc que inclouen restriccions implicades... (cal mirar-ho)
- De vegades aplicar constraints implicades a **subproblemes** permet fer-los més tractables...

Restriccions Implicades (iii)

- De vegades es poden fer **constraints globals** ad-hoc que inclouen restriccions implicades... (cal mirar-ho)
- De vegades aplicar constraints implicades a **subproblemes** permet fer-los més tractables...

Com trobar restriccions implicades?

- Identifiquem assignacions parcials òbviament errònies que no haurien d'ocórrer durant la cerca: **intuïció**, **observació de la cerca** (afegir variables per a tenir info de l'evolució de la cerca)
- comprovem que efectivament afegir les restriccions implicades funciona.

Trencament de simetries (i)

Les **simetries** transformen solucions en altres solucions

- Pot aparèixer inherentment al problema (reines).
- Es pot introduir amb la model·lització.

Les simetries s'han d'evitar/trencar perquè impliquen una pèrdua de temps:

- després de buscar en opcions que no porten a cap solució, opcions equivalents simètricament poden tornar-se a explorar...
- De vegades les solucions simètriques no ens interessen (p.ex optimització)

Trencament de simetries (ii)

Car Sequencing

- En un model on els cotxes no estan agrupats segons classes, sinó que cada cotxe indica les opcions que vol, afegim simetries entre cotxes que volen la mateixa opció.
- En el model amb classes de cotxes, hem d'agefir restriccions per a assegurar el número exacte de cotxes per cada classe.

Trencament de simetries (ii)

Social Golfers

32 golfistes volen jugar en 8 grups de 4 cada setmana de manera que 2 golfistes juguin junts (en el mateix grup) com a molt un cop. Trobar una planificació per n setmanes.

Trencament de simetries (ii)

Social Golfers

32 golfistes volen jugar en 8 grups de 4 cada setmana de manera que 2 golfistes juguin junts (en el mateix grup) com a molt un cop. Trobar una planificació per n setmanes.

- Variables 0/1: $x_{i,j,k,l} = 1$ si el jugador i és el jugador j-èssim en el grup k de la setmana l, sino $= 0$.
- Quines simetries hi ha? Els jugadors de cada grup es poden permutar en qualsevol solució i donen solucions equivalents, els grups en respecte la setmana també i les setmanes respecte la planificació total també.

Trencament de simetries (ii)

Social Golfers

32 golfistes volen jugar en 8 grups de 4 cada setmana de manera que 2 golfistes juguin junts (en el mateix grup) com a molt un cop. Trobar una planificació per n setmanes.

Reformular per evitar simetries:

- Eliminar simetries entre jugadors en un grup utilitzant **set variables** per representar els grups:
  - $G_{k,l}$ representa el k-èssim grup de la setmana l
  - el valor de $G_{k,l}$ representa el conjunt de jugadors en el grup
- Els constraints a imposar són:
  - la cardinalitat de cada conjunt és 4
  - Els conjunts de cada setmana no es solapen: per a qualsevol l, els conjunts $G_{k,l}$ per $k = 1, \ldots, 8$ tenen intersecció buida
  - qualsevols dos conjunts en setmanes diferents tenen com a molt un element en comú
- Calen resoledors de constraints que suportin set variables

Trencament de simetries (iii)

Molt sovint amb remodelació no n'hi ha prou i cal **afegir restriccions per a trencar simetries**

- a diferència de les restriccions implicades, **es canvia el conjunt de solucions**
- poden permetre més restriccions implicades...

El balancí

Posar 3 nens de 36, 32 i 16 Kg en un balancí de 5 seients per banda separats com a mínim per 2 seients de manera que estiguin en equilibri

Trencament de simetries (iii)

Molt sovint amb remodelació no n'hi ha prou i cal **afegir restriccions per a trencar simetries**

- a diferència de les restriccions implicades, **es canvia el conjunt de solucions**
- poden permetre més restriccions implicades...

El balancí

Posar 3 nens de 36, 32 i 16 Kg en un balancí de 5 seients per banda separats com a mínim per 2 seients de manera que estiguin en equilibri

- $A, B, C \in -5..5$
- $36 * A + 32 * B + 16 * C = 0$
- $|A - B| > 2, |A - C| > 2, |B - C| > 2$

Trencament de simetries (iii)

Molt sovint amb remodelació no n'hi ha prou i cal **afegir restriccions per a trencar simetries**

- a diferència de les restriccions implicades, **es canvia el conjunt de solucions**
- poden permetre més restriccions implicades...

El balancí

Posar 3 nens de 36, 32 i 16 Kg en un balancí de 5 seients per banda separats com a mínim per 2 seients de manera que estiguin en equilibri

- $A, B, C \in -5..5$
- $36 * A + 32 * B + 16 * C = 0$
- $|A - B| > 2, |A - C| > 2, |B - C| > 2$

Solucions:

$$[-4, 2, 5], [-4, 4, 1], [-4, -5, 1], [4, -5, -1], [4, -4, -1], [4, -2, -5]$$

Trencament de simetries (iii)

Molt sovint amb remodelació no n'hi ha prou i cal **afegir restriccions per a trencar simetries**

- a diferència de les restriccions implicades, **es canvia el conjunt de solucions**
- poden permetre més restriccions implicades...

El balancí

Posar 3 nens de 36, 32 i 16 Kg en un balancí de 5 seients per banda separats com a mínim per 2 seients de manera que estiguin en equilibri

- $A, B, C \in -5..5$
- $36 * A + 32 * B + 16 * C = 0$
- $|A - B| > 2, |A - C| > 2, |B - C| > 2$

i si en posem algun a l'esquerra fixat?

$$A < 0$$

Combinació de models (i)

- De vegades una perspectiva és clarament millor que cap altra.
- Pot passar que diferents perspectives permetin diferents maneres d'expressar els constraints i per tant surtin diferents constraints implicats. Quina triem?

Combinació de models (i)

- De vegades una perspectiva és clarament millor que cap altra.
- Pot passar que diferents perspectives permetin diferents maneres d'expressar els constraints i per tant surtin diferents constraints implicats. Quina triem?

No cal que escollim una perspectiva, en podem usar més d'una, només cal que les lliguem mitjançant **channelling constraints**!

Combinació de models (i)

- De vegades una perspectiva és clarament millor que cap altra.
- Pot passar que diferents perspectives permetin diferents maneres d'expressar els constraints i per tant surtin diferents constraints implicats. Quina triem?

No cal que escollim una perspectiva, en podem usar més d'una, només cal que les lliguem mitjançant **channelling constraints**!

Compte, de vegades pot afegir overhead innecessari...
Quines variables guien la cerca?

- Els channelling constraints fa que no sigui massa rellevant, estan interlligades.
- Sol valer la pena començar pel conjunt més petit...

Combinació de models (ii)

Problemes de permutació

- Un **problema de permutacions** per a n variables x_i amb domini $v_1..v_n$ consisteix en assignacions que faci les n variables diferents. (Sol haver-hi altres constraints que filtren les solucions)
- La **perspectiva dual** es pot escriure: cada valor li toca a una variable.
- i es poden lligar:

$$x_i = j \equiv v_j = i$$

- l'`alldifferent` ve directe...

Combinació de models (iii)

Golomb Ruler

Una G-Ruler amb m marques consisteix en: ([0,1,4,10,12,17])

- *un conjunt de m enters* $0 = x_1 < x_2 < \cdots < x_m$
- *les* $\frac{m(m-1)}{2}$ *diferències* $x_j - x_i$ *són diferents*
- *Objectiu: trobar una G-ruler amb la* x_m *més petita*

Combinació de models (iii)

Golomb Ruler

Una G-Ruler amb m marques consisteix en: ([0,1,4,10,12,17])

- *un conjunt de m enters* $0 = x_1 < x_2 < \cdots < x_m$
- *les* $\frac{m(m-1)}{2}$ *diferències* $x_j - x_i$ *són diferents*
- *Objectiu: trobar una G-ruler amb la* x_m *més petita*

Perspectiva 1:

- $x_j - x_i \neq x_l - x_k$ *per a tots els parells diferents*
- $x_1 < x_2 < \cdots < x_m$

Perspectiva 2:

- `alldifferent(`$[d_{1,2}, d_{1,3}, \ldots, d_{m-1,m}]$`)`
- $d_{i,k} = d_{i,j} + d_{j,k}$ *per a* $1 \leq i < j < k \leq m$

Combinació de models (iii)

Golomb Ruler

Una G-Ruler amb m marques consisteix en: ([0,1,4,10,12,17])

- *un conjunt de m enters* $0 = x_1 < x_2 < \cdots < x_m$
- *les* $\frac{m(m-1)}{2}$ *diferències* $x_j - x_i$ *són diferents*
- *Objectiu: trobar una G-ruler amb la* x_m *més petita*

Perspectiva 1:

- $x_j - x_i \neq x_l - x_k$ *per a tots els parells diferents*
- $x_1 < x_2 < \cdots < x_m$

Perspectiva 2:

- `alldifferent(`$[d_{1,2}, d_{1,3}, \ldots, d_{m-1,m}]$`)`
- $d_{i,k} = d_{i,j} + d_{j,k}$ *per a* $1 \leq i < j < k \leq m$

Channelling constraint:

$$d_{i,j} = x_j - x_i$$

Consideracions finals

- Reduir el número de variables. (però que propaguin bé)
- Reduir el número de constraints, per exemple us de constraints globals. (però que propaguin bé)

Consideracions finals

- Reduir el número de variables. (però que propaguin bé)
- Reduir el número de constraints, per exemple us de constraints globals. (però que propaguin bé)
- Afegir més variables: Per a expressar constraints i/o per a permetre perspectives diferents
- Afegir més restriccions: implicades, trencament de simetries i channelling

Consideracions finals

- Reduir el número de variables. (però que propaguin bé)
- Reduir el número de constraints, per exemple us de constraints globals. (però que propaguin bé)
- Afegir més variables: Per a expressar constraints i/o per a permetre perspectives diferents
- Afegir més restriccions: implicades, trencament de simetries i channelling
- **Testejar empíricament!!!**