Programmation Logique par Contraintes
(introduction)

Michel RUEHER

De la PL vers la PLC

- Introduction de la PLC : motivations, exemples
- Principes de la ***programmation par contraintes**** (définition, résolution)
- La programmation logique avec contraintes (méta-interprète et machine abstraites)
- Limites

***Pour une présentation « grand public », voir :
http://binaire.blog.lemonde.fr/2015/11/20/la-programmation-par-contraintes-expliquee-a-ma-garagiste-ou-a-mon-fleuriste/

PROGRAMMATION LOGIQUE ≈ SPÉCIFICATION EXÉCUTABLE

Instruction ≡ Relation

- ***Spécification d'une solution en terme de relations*** (produits cartésiens) entre entités (ou classes d'entités)

- ***Réversibilité***

  Exemple: **append/3:**

  ```
  append([],L,L).
  append([H|L1],L2,[H|L3]):- append(L1,L2,L3).
  ```

  Concaténation mais aussi décomposition de listes

LIMITES DE LA PROGRAMMATION EN LOGIQUE :

Le prédicat **=/2** n'évalue pas ses arguments et le prédicat **is/2** n'est pas une relation.

?- **1 + X = 3.** → **fail** % **1+X** correspond à l'arbre **+(1,x)**

?- **X is 3 - 1.** → **X = 2** % is : évaluation du terme droit

?- **X + 1 is 3.** → *erreur d'exécution*

Le premier objectif de la PLC était de donner un statut relationnel à l'arithmétique.

LIMITES DE LA PROGRAMMATION EN LOGIQUE :

?- 1 + X = 3→ FAIL

Deux solutions en Prolog :

- **Utilisation des axiomes de Peano**

```
plus(0,Y,Y).
plus (s(X),Y,s(Z)) :- plus (X,Y,Z).
```

1 + X = 3 devient ?- **plus(s(0),Y,s(s(s(0))))**.

Y = s(s(0))

- **Retardement de l'évaluation**

```
plus (X,Y,Z) :-freeze(X, freeze(Y,Z is X+Y)),
               freeze (Y, freeze(Z,X is Z-Y)),
               freeze (X, freeze(X,Y is X-Z)).
```

Ne permet pas de définir une relation d'ordre sur des variables

```
sort(X1,X2,X3) :-  X1 ≥ X2, X2 ≥ X3, ...
?- sort (X1,X2,X3).
```

→ *FAIL*

PROGRAMMATION PAR CONTRAINTES: INTUITIONS

N-REINES:

COMMENT ANTICIPER LES IMPASSES ?

	1	2	3	4
1	R	x	x	x
2	x	x		
3	x		x	
4	x			x

SEND + MORE = MONEY

COMMENT TROUVER L'INFORMATION PERTINENTE

```
  S E N D
+ M O R E
---------
= M O N E Y
```

Knight Tour (Hamiltonian path), 4-Color problem

COMMENT TROUVER LES BONNES HEURISTIQUES ?

SEND + MORE = MONEY : PROLOG

```
sm([S,E,N,D,M,O,R,Y]) :-
        generate([S,E,N,D,M,O,R,Y],[0,1,2,3,4,5,6,7,8,9]),      % generate
        S>0, M >0,                                               % test
        test_sum(S,E,N,D,M,O,R,Y).                               % test
generate([X|Xs],Vs):-
        select(X,Vs,Vr),
        generate(Xs,Vr).
generate([],_).
test_sum(S, E, N, D, M, O, R, Y) :-
        V1 is           1000*S + 100*E + 10*N + D
             +          1000*M + 100*O + 10*R + E,
        V2 is 10000*M + 1000*O + 100*N + 10*E + Y,
        V1 = V2.
```

SEND + MORE = MONEY : AVEC CONTRAINTES

```
sm([S,E,N,D,M,O,R,Y]) :-
      fd_domain([S,E,N,D,M,O,R,Y], 0, 9),     % Définition domaines
      S#>0, M#>0,                             % Pose et résolution
      fd_all_different([S,E,N,D,M,O,R,Y]),    % des
      sum(S,E,N,D,M,O,R,Y),                   % contraintes
      fd_labelingff([S,E,N,D,M,O,R,Y]).       % Recherche
                                              %   & Enumération
sum(S, E, N, D, M, O, R, Y) :-
                    1000*S + 100*E + 10*N + D
        +           1000*M + 100*O + 10*R + E,
    #= 10000*M + 1000*O + 100*N + 10*E + Y.
```

SEND + MORE = MONEY : AVEC CONTRAINTES (2)

Après la **déclaration des domaines** on a:

D(S) = D(E) = D(N) = D(D) = D(M)

= D(O) = D(R) = D(Y) = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Après la **pose des contraintes S#>0** et **M#>0** on a

D(S) = D(M) = {1, 2, 3, 4, 5, 6, 7, 8, 9}

De la contrainte **sum(S,E,N,D,M,O,R,Y)** :

1000*S + 100*E + 10*N + D + 1000*M + 100*O + 10*R + E

= 10000*M + 1000*O + 100*N + 10*E + Y

on déduit :

9000*M = 1000*S + 91*E + D – 900*O – 90*N –Y

SEND + MORE = MONEY : AVEC CONTRAINTES (3)

D'ou:

$$M \leq \frac{1}{9}\max(D(S)) + \frac{91}{9000}\max(D(E)) + \frac{1}{9000}\max(D(D)) + \frac{1}{900}\max(D(R)) - \frac{1}{10}\min(D(O)) - \frac{1}{100}\min(D(N)) - \frac{1}{9000}\min(D(Y))$$

Et

$$M \leq \frac{9}{9} + \frac{91\times9}{9000} + \frac{9}{9000} + \frac{9}{900} - \frac{0}{10} - \frac{0}{100} - \frac{0}{9000} = 1.102$$

$\Rightarrow$ D(M) ={1}

Par propagation on obtient :

D(S) = {9} D(M) = {1} D(O) = {0}

D(E) = {4, 5, 6, 7} D(N) = {5, 6, 7, 8} D(D) = {2, 3, 4, 5, 6, 7, 8}

D(R) = { 2, 3, 4, 5, 6, 7, 8} D(Y) = {2, 3, 4, 5, 6, 7, 8, 9}

SEND + MORE = MONEY AVEC CONTRAINTES (4)

- **3 variables sont instanciées**
- **Enumération (plus petit domaine)**

  Essaye successivement **:**

  E = 4 (échec),

  E = 5 ⇒Solution
- **Conclusion :**
  - **2 énumérations pour trouver la solution**
  - **4 énumérations pour montrer qu'il n'existe qu'une solution … au lieu 10^8**

PROGRAMMATION AVEC CONTRAINTES - DÉFINITIONS

- **Définition d'une contrainte :**

  Une contrainte est une relation logique (une propriété qui doit être vérifiée) entre différentes inconnues

  → ***réduction des valeurs*** que peuvent prendre simultanément les variables

  Exemple : "*x + 3*y = 12*" → restreint valeurs possibles pour *x* et *y*.

  Les ***inconnues*** (variables) : valeurs dans un ensemble donné fini(***domaine***)

- **Forme d'une contrainte:**

  - ***relationnelle, déclarative*****,**
  - définie en ***extension*****:** *"(x=0 et y=1) ou (x=0 et y=2) "*
  - définie en ***intension*** : *"x < y"* ou encore *"A et B => non(C)"*
  - numérique, ensembliste, booléenne, univers de Herbrandt,…

Définition d'un CSP

Un CSP (Constraint Satisfaction Problem) est **un triplet *(X,D,C)*** tel que :

- ***X*** *= { X1, X2, ..., Xn}* =ensemble des variables (les **inconnues** du problème)
- ***D*** : fonction qui associe à chaque variable *Xi* son ***domaine*** *D(Xi)*, l'ensemble des valeurs que peut prendre *Xi*
- ***C*** *= {C1, C2, ..., Ck}* est l'ensemble des contraintes.

Exemple:

$X = \{a,b,c,d\}$ $D(a) = D(b) = D(c) = D(d) = \{0,1\}$ $C = \{ a \neq b, c \neq d, a+c < b \}$

Solution d'un CSP : ***affectation*** des variables, de telle sorte que toutes les contraintes soient satisfaites

CSP sur-contraint : Un CSP qui n'a pas de solution ($\rightarrow$ *max-CSP)*

CSP sous-contraint : Un CSP qui admet beaucoup de solutions différentes

PRINCIPES DE LA PPC

Modélisation sous la forme d'un CSP

- ***identifier*** l'ensemble des ***variables*** *X* (les inconnues du problème) et leurs domaines
- ***identifier*** les ***contraintes*** *C* entre les variables

On ne se soucie pas de savoir comment résoudre le problème : on cherche simplement à le spécifier formellement

PRINCIPES DE LA PPC : modélisation des n-reines

- **Les *inconnues*** : les positions des reines sur l'échiquier
  - association à chaque reine *i* de deux variables *Li* et *Ci* (la ligne et la colonne sur laquelle placer la reine)
  - ou une variable Xi par colonne : valeur sera la position de la reine (numéro de ligne)
- **Les *contraintes* :**
  - *les reines doivent être sur des lignes différentes :*

    {Xi ≠Xj pour i dans {1,2,3,4}
  - les reines doivent être sur des colonnes différentes :

    aucune contrainte avec la deuxième modélisation !
  - les reines doivent être sur des diagonales différentes :

    {Xi ≠Xj +/- (i-j) pour i,j dans {1,2,3,4} et i≠j

TECHNIQUES DE RÉSOLUTION (1)

« Generate & Test »

(recherche naïve, prolog standard)

Énumère toutes les affectations totales pour trouver celles qui satisfont toutes les contraintes :

Tant que toutes les contraintes ne sont pas satisfaites

- Nouvelle instanciation de **toutes** les variables
- Évaluation des contraintes

« Backtrack » chronologique

Tant que toutes les variables ne sont pas instanciées

- Instanciation **d'une** nouvelle variable
- Évaluation des contraintes totalement instanciées

*Si le système courant est **inconsistant → retour arrière** :* choix d'une nouvelle valeur pour la *dernière* variable instanciée pour laquelle il reste des valeurs disponibles

TECHNIQUES DE RÉSOLUTION (2)

« Generate & Test » versus « Backtrack » chronologique

Contraintes: $X = Y$, $X \neq Z$, $Y > Z$

Domaines : $D_X = D_y = D_z = \{1, 2\}$,

Generate & Test:

X	Y	Z	Test
1	1	1	fail
1	1	2	fail
1	2	1	fail
1	2	2	fail
2	1	1	fail
1	1	1	fail
2	1	2	fail
2	2	1	succes

Backtracking:

X	Y	Z	Test
1	1	1	fail
1	1	2	fail
1	2		fail
2	1		fail
2	2	1	succes

TECHNIQUES DE RÉSOLUTION (3)

Programmation par Contraintes: alternance de filtrage et de recherche

- ***Filtrage par consistance locale*: *élimination a priori*** des valeurs qui ne peuvent être dans aucune solution

  → *travail local à chaque contrainte*

- ***Recherche:*** utilisation *d'heuristiques* pour le c*hoix* des variables et valeurs qui devraient permettre de découvrir rapidement les *impasses*

FILTRAGE PAR CONSISTANCE D'ARC

Contraintes :

x1 < x2

x5 + 3 = x2

x3 > x1

x4 > x2 + 3

Domaines :

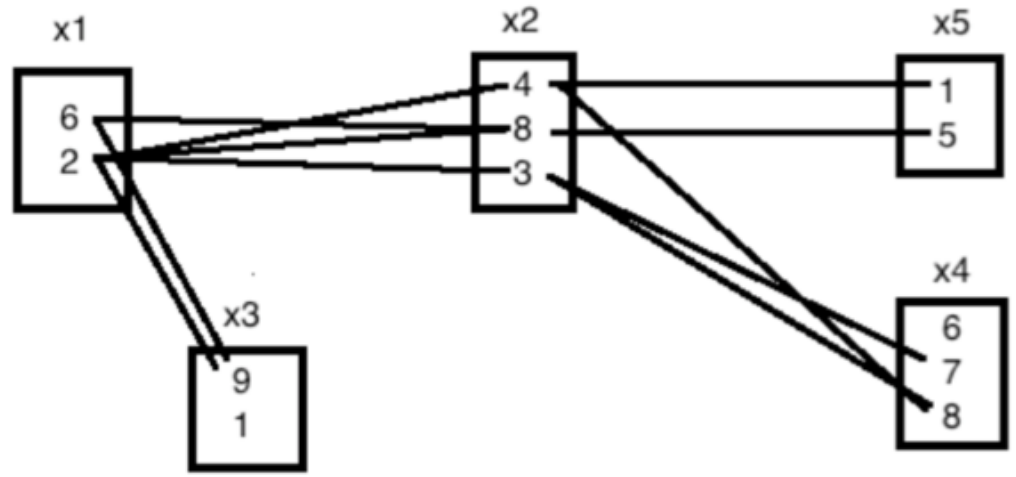
Dx1 = {2, 6},

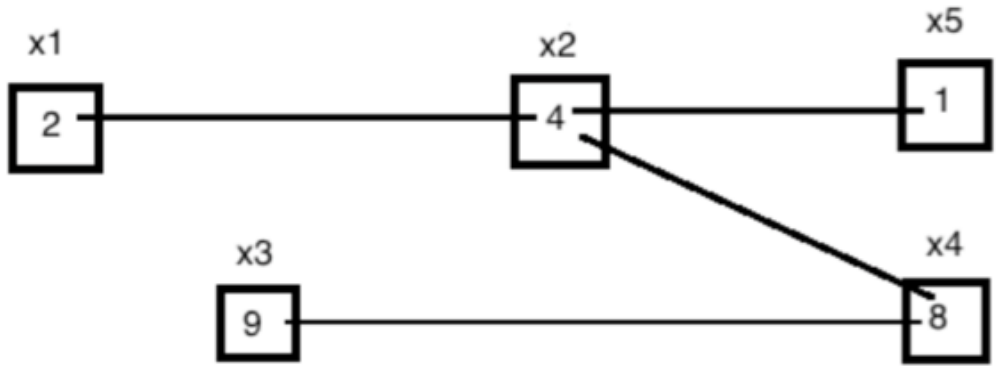
Dx2 = {3, 4, 8},

Dx3 = {1, 9},

Dx4 = {6, 7, 8},

Dx5 = {1, 5}





PPC: ALGORITHME DE FILTRAGE (1)

- ***Chaque*** contrainte est considérée séparément
- Les valeurs ***trivialement inconsistantes*** sont éliminées

IN(**in** C, **inout** D) % C et D ensemble des contraintes et domaines

```
Queue← C ;
while Queue ≠ Ø
    c ←POP Queue;
    D' ←narrow(c,D);
    If D' ≠ D    then   D ← D';
                        Queue ← Queue ∪
                        {c' ∈ C tel que V ar(c) ∩ V ar(c')  ≠ Ø}
    endif
endwhile
```

LIMITE DE LA CONSISTANCE D'ARC

Contraintes :

$X \neq Y$

$Y \neq Z$

$Z \neq X$

Domaines :

$X, Y, Z \in \{1,2\}$

CONTRAINTES GLOBALES

Contraintes globales :
algorithmes de filtrage efficaces pour des contraintes spécifiques

Exemple: *alldiff*

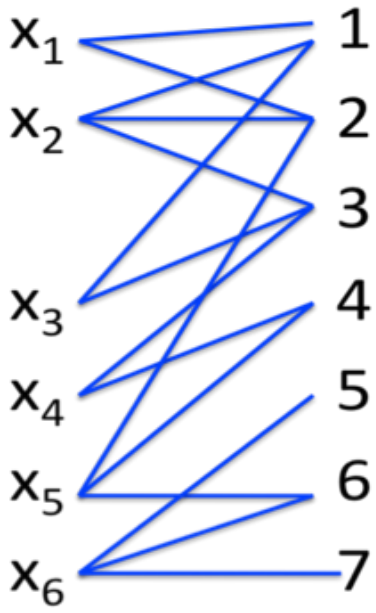
Les variables doivent prendre deux à deux des valeurs différentes

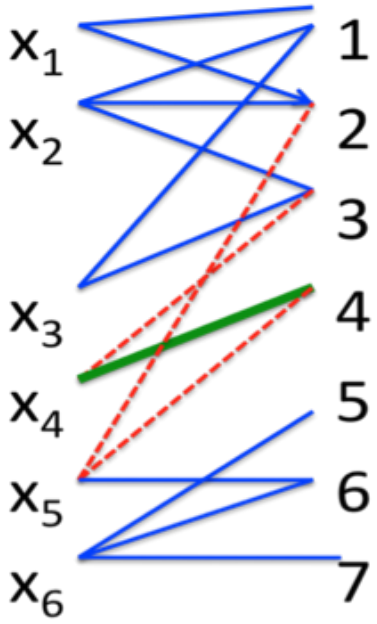
Problème: identifier les groupes de k variables qui ne peuvent pendre que k valeurs ... 2^k groupes possibles

Solution: utiliser des algorithmes de matching ou de flow

CONTRAINTES GLOBALES - AllDiff (1)

- alldifferent(x_1, x_2, x_3, x_4, x_5, x_6)
- $x_1 \in \{1, 2\}$ $x_2 \in \{1, 2, 3\}$ $x_3 \in \{1, 3\}$ $x_4 \in \{3, 4\}$
  $x_5 \in \{2, 4, 6\}$ $x_6 \in \{5, 6, 7\}$





CONTRAINTES GLOBALES - AllDiff (2)

alldifferent(x_1, x_2, x_3, x_4, x_5, x_6)

$x_1 \in \{1, 2\}$

$x_2 \in \{1, 2, 3\}$

$x_3 \in \{1, 3\}$

$x_4 \in \{3, 4\}$

$x_5 \in \{2, 4, 6\}$

$x_6 \in \{5, 6, 7\}$

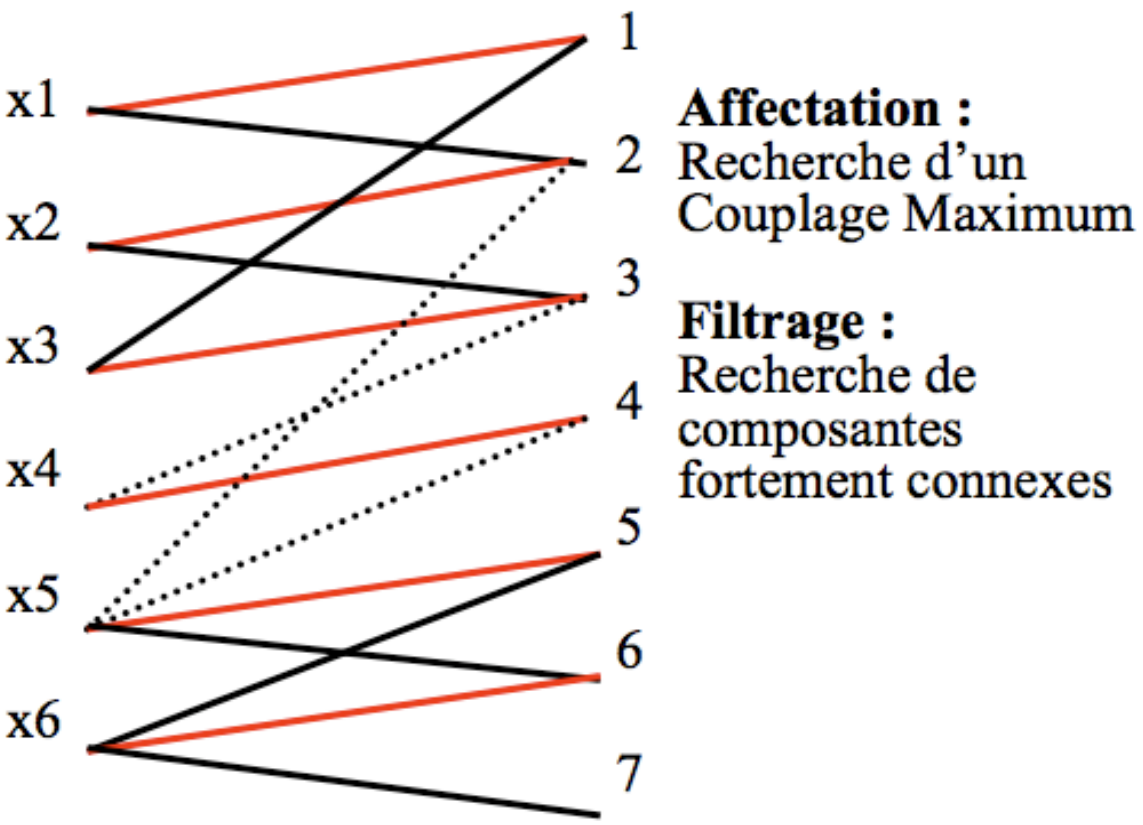


Schéma général de la PLC

Filtrage & Recherche

1. **Pose des contraintes**
2. **Tant qu'il existe une variable non-instanciée faire**
   a. **Réduction des domaines** *(à l'aide des contraintes)*
   b. **Choix d'une variable à instancier**

PLC : filtrage & résolution (exemple)

Variables / domaines : $x_1 \in \{1, 2\}$, $x_2 \in \{0, 1, 2, 3\}$, $x_3 \in \{2, 3\}$

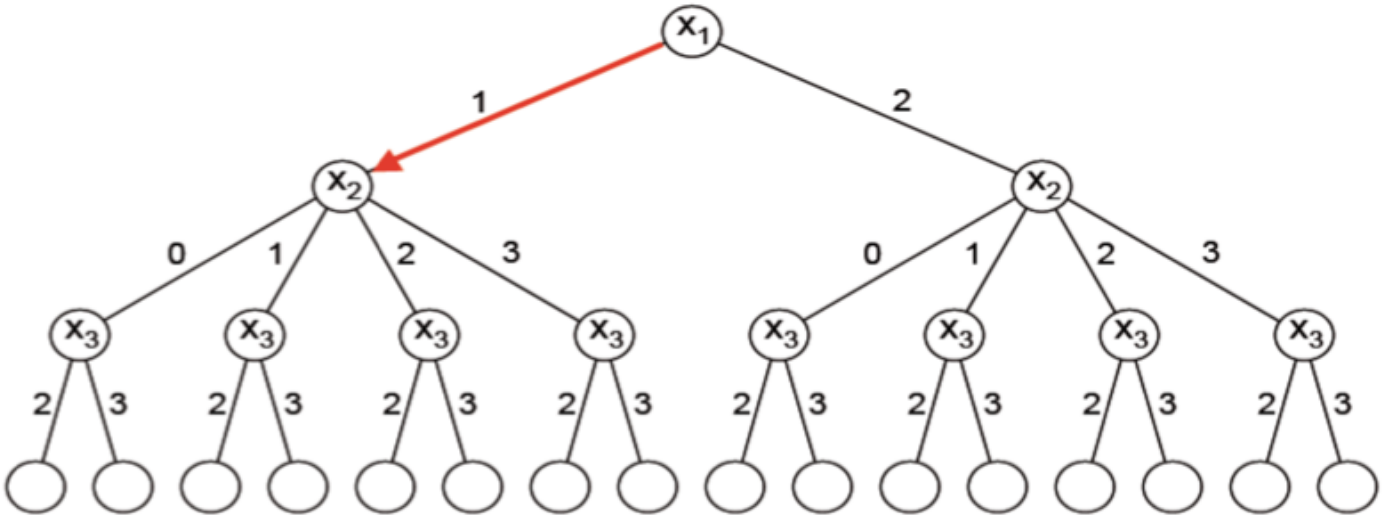
Contraintes:

$x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1, x_2, x_3)

Arbre de recherche :



PLC : filtrage & recherche (exemple, suite)

Variables / domaines : $x_1 \in \{1, 2\}$, $x_2 \in \{0, 1, \sout{2, 3}\}$, $x_3 \in \{2, 3\}$

Contraintes:

$x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1, x_2, x_3)

Pas d'autre filtrage possible

→ *Choix de valeur pour x_1*

PLC : filtrage & recherche (exemple, suite)

Variables / domaines : $x_1 \in \{1\}$, $x_2 \in \{0, \cancel{1}\}$, $x_3 \in \{\cancel{2, 3}\}$

Contraintes:

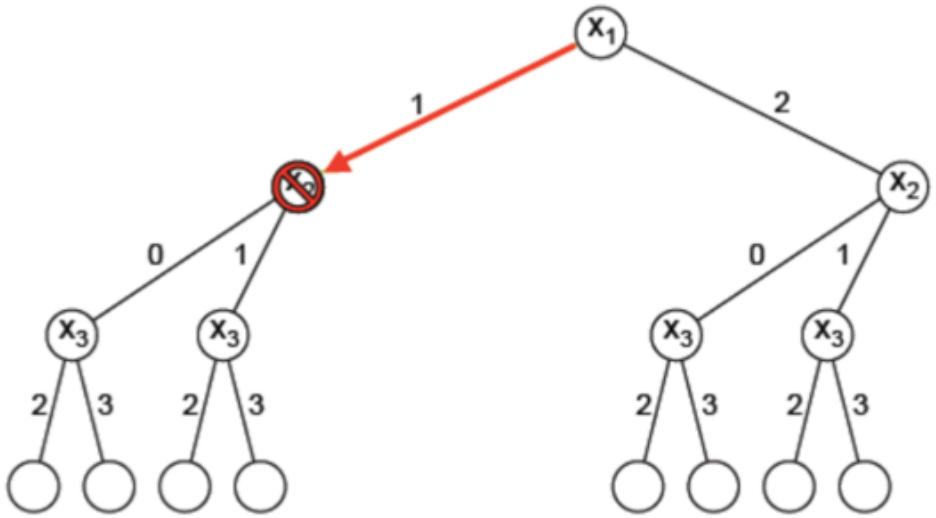
$x_1 > x_2$

$x_1 + x_2 = x_3$

alldifferent(x_1, x_2, x_3)

Domaine de x_3 vide

→ Nouveau choix de valeur pour x_1



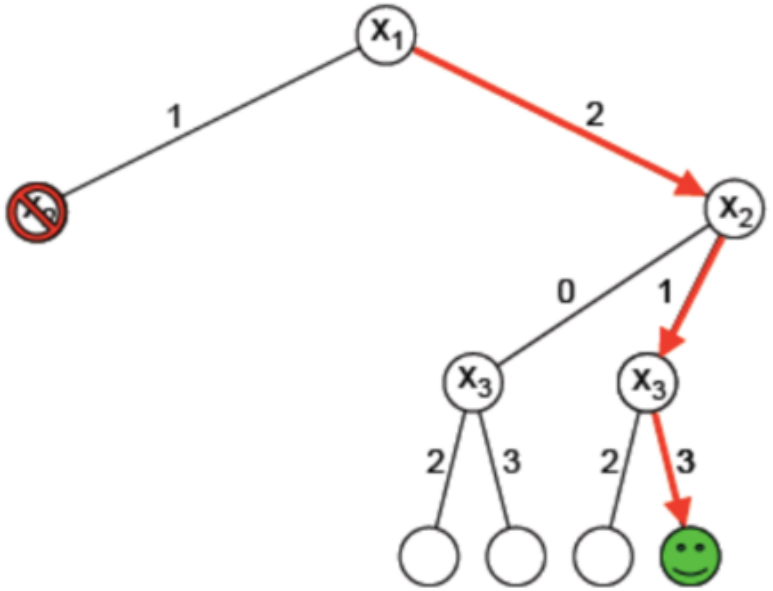
PLC : filtrage & recherche (exemple, suite)

Variables / domaines : $x_1 \in \{2\}$, $x_2 \in \{\cancel{0}, 1, \cancel{2}, \cancel{3}\}$, $x_3 \in \{\cancel{2}, 3\}$

Contraintes: $x_1 > x_2$

$x_1 + x_2 = x_3$ (après alldifferent)

alldifferent(x_1, x_2, x_3)



Techniques de Recherche pour la PLC(1)

Parcours de l'arbre de recherche

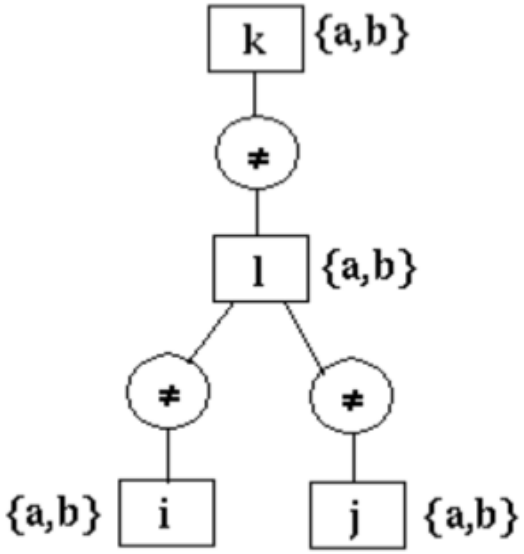
- **Heuristiques** qui permettent dans de nombreux cas d'éliminer ***a priori*** certaines branches

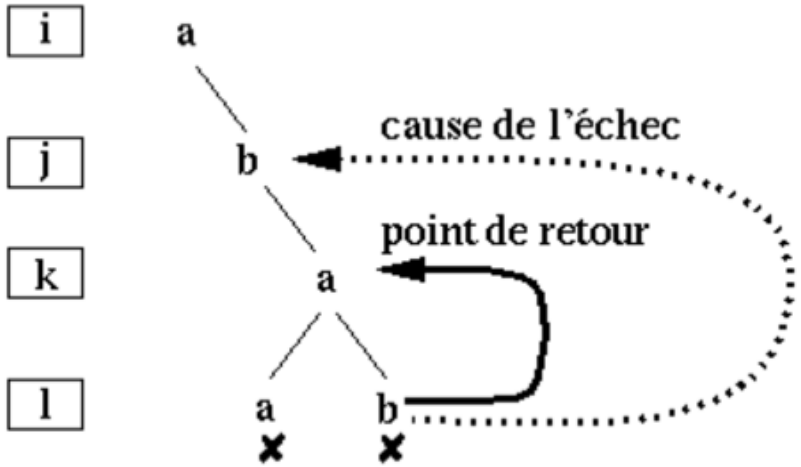
- **Combinaison filtrage & heuristiques:**
  → " **MAC** "(Maintien de la Consistance d'Arc)
  filtrage après chaque chaque instanciation de variable

L'élimination des solutions symétriques est un problème majeur dans de nombreuses applications

Techniques de Recherche pour la PLC (2)

Le ***backtrack chronologique*** peut entrainer la répétition d'échecs causés par une même raison.



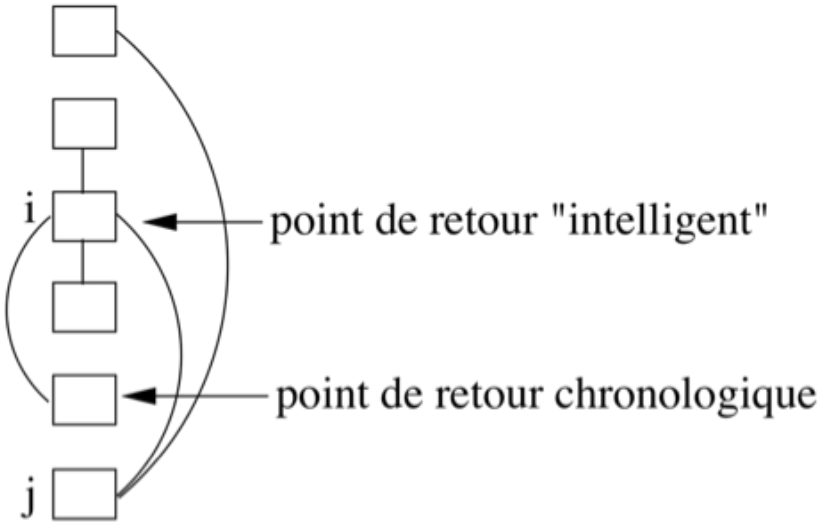


Techniques de Recherche pour la PLC (3)

Backtrack dirigé par le graphe de contraintes : GBJ

Si échec sur une variable **j**

→ retour sur la variable **i** la plus récemment instanciée et connectée à j par une contrainte



PLC : limites (1)

Solution de : $a^n + b^n = c^n$

avec a,b,c, n **entiers** et **n > 2**