# Introduction à la programmation par Contraintes

**Michel RUEHER**

**Université de Nice - Sophia Antipolis**

**November 27, 2010**

## Plan du cours “Contraintes”

1. Introduction – Michel Rueher
2. Contraintes globales et modélisation – Jean-Charles Régin
3. Stratégie de recherche et méthodes heuristiques – Gilles Trombettoni
4. Arithmétique d’intervalles, algèbre linéaire, algèbre non-linéaire, applications de l’analyse par intervalles – Jean Pierre Merlet

- Conférences :
  **CP** (Int. Conf. on Principles and Practice of Constraint Programming)
  **CPAIOR** (Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems)
- Revues: Constraint Journal, Journal of AI
- Ouvrages:
  - Handbook of Constraint Programming. Edited by Francesca Rossi, Peter van Beek, Toby Walsh. Elsevier, 978 pages, 2006
  - Constraint and Integer Programming: Toward a Unified Methodology, Michela Milano (Ed.), Kluwer Academic Pub. (2003)
  - Kim Marriott and Peter J. Stuckey . Programming with Constraints: an Introduction . MIT Press, 1998.
  - P. Van Hentenryck, Y. Deville, and L Michel. Numerica. A modelling language for global optimization. MIT Press, 1997.
  - François Fages Programmation logique par contraintes, Editions Ellipses, 1996.

## Plan du "Introduction"

1. Présentation informelle des *concepts de base*
2. La *programmation en logique avec contraintes (CLP)*
3. Les *techniques de satisfaction de contraintes (CSP) – Domaines finis*
4. Les contraintes dans le domaine continu: *la propagation d'intervalles*
5. Application des techniques de programmation par contraintes *à la génération de jeux de test et à la vérification de programmes*

# Partie I : présentation informelle des concepts de base

*"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."*

Eugene C.Freuder, Constraints, April 1997

## Origines

- **IA** : “Scene Labelling” - Waltz, résolution de problèmes,..
- **Programmation Logique** : unification $\rightarrow$ constraint solving
- **Recherche Opérationelle**: problème combinatoires NP Difficiles

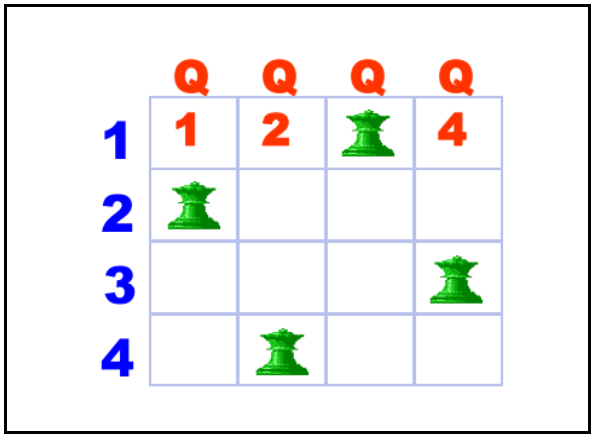
## Intuitions

- La complexité dans le pire des cas est en général trop pessimiste
- Une analyse de la complexité en moyenne est en général impossible

$\rightarrow$ CP: Etude des problèmes comme des phénomènes naturels

# Intuitions (suite)

- **SEND+MORE=MONEY** : comment inférer l'information pertinente ?
- **N Reines** : comment "prévoir" les impasses ?




- **Chemin hamiltonien du cavalier**: comment trouver les bonnes heuristiques ?

→ **Applications**

- L'ordonnancement
- La planification
- La gestion de ressources
- L'optimisation de systèmes non-linéaires
- ...

→ **Apports**

- Plus *grande souplesse* pour énoncer et résoudre les problèmes
- Utilisation dans un cadre uniforme des techniques combinatoires, de recherche opérationnelle, d'analyse numérique

## Exemple 1 : coloriage d'un graphe

Le coloriage des arêtes (ou des sommets) d'un graphe consiste à affecter une couleur à chaque arête (resp. sommet) tel que deux arêtes (resp. sommets) adjacentes ne soient pas coloriées de la même manière.

**Problèmes**:

- Trouver le nombre minimal de couleurs pour colorier un graphe
- Rechercher un coloriage effectif pour un nombre donné de couleurs

APPLICATIONS :

- **Le coloriage d'une carte** : il s'agit de colorier une carte (e.g. la carte de France) tel que deux régions voisines soient coloriées avec des couleurs différentes.

  **Modélisation** :

  – les sommets du graphe correspondent aux régions,

  – les arcs relient deux sommets qui correspondent à des régions ayant une frontière commune

  Quatre couleurs sont suffisantes lorsque le graphe est planaire mais il n'existe pas d'algorithme polynomial permettant d'effectuer le coloriage.

- **Planification des examens** : il s'agit de définir un planning des examens tel que le temps requis pour l'ensemble des examens soit minimal, et qu'il n'y ait pas d'incompatibilités pour les étudiants.
  **Modélisation** :
  - les sommets du graphe correspondent aux épreuves,
  - les arcs relient deux sommets qui correspondent à des épreuves que doivent passer les mêmes étudiants (le nombre de couleurs correspond alors au nombre minimal de créneaux horaires requis)

  Si $k$ est le nombre de salles dont on dispose, alors chaque couleur peut au plus colorier $k$ noeuds. Le problème se complique lorsqu'il faut prendre en compte les contraintes de préférence (e.g. éviter qu'un étudiant ne passe plus de 3 épreuves en deux jours).

## Exemple 2 : problème de balistique

Le problème consiste à trouver le point d'impact sur le sol d'un objet lancé dans un champ gravitationnel uniforme $\vec{g}$. La vitesse initiale $\vec{V_i}$ de l'objet a une incidence $\alpha$ avec le sol.

De plus les valeurs de ces deux dernières variables ne sont pas connues avec précision (pour cause de tolérance mécanique ou d'imprécision dans les mesures, etc).

La solution sera donc un segment couvrant les points d'impact possibles pour le projectile.

# Problème de balistique

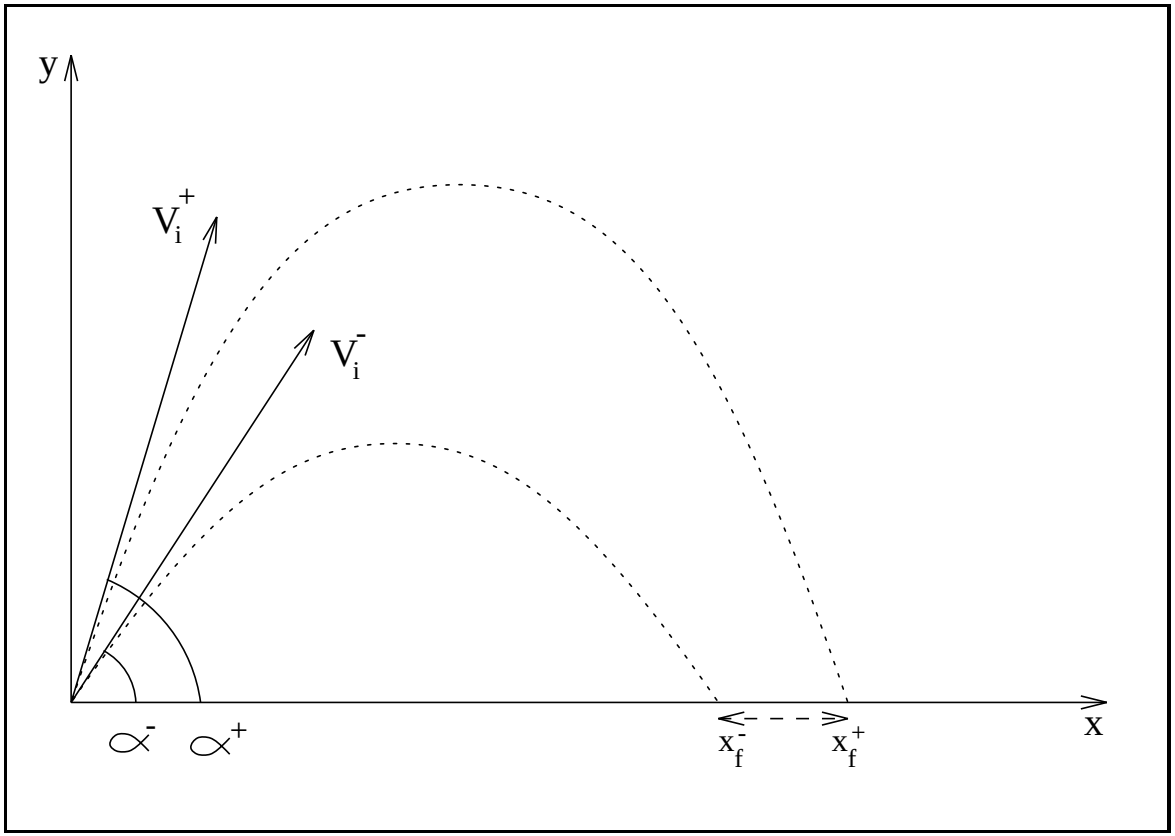



Figure 1: Trajectoires possibles du projectile

# Exemple 4: modélisation avec des contraintes booléennes

$$\neg FO \Rightarrow (O \Leftrightarrow (X \vee Y)) \wedge$$
$$\neg FA \Rightarrow (A \Leftrightarrow (X \wedge Y)) \wedge$$
$$\neg FN \Rightarrow (N \Leftrightarrow \neg A) \wedge$$
$$\neg FG \Rightarrow (Z \Leftrightarrow (N \wedge O))$$

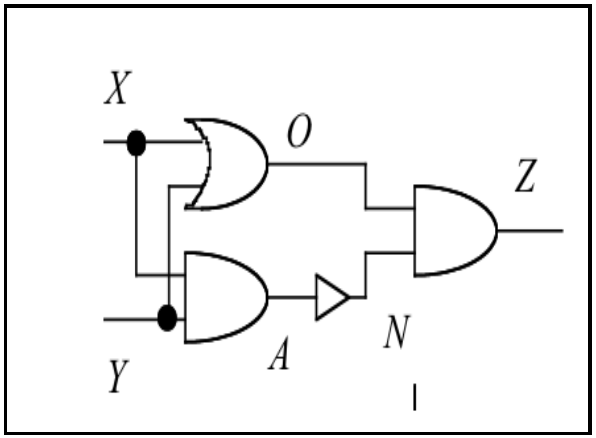
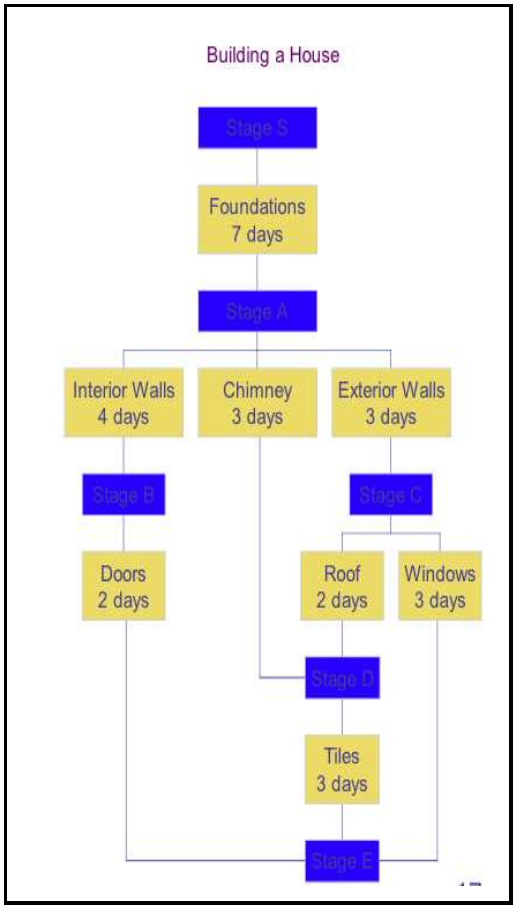



Figure 2: Porte XOR

# Exemple 5: Modélisation d'un problème d'ordonnancement de tâches

Début: $T_s \geq 0$
Fondations: $T_A \geq T_s + 7$
Murs intérieurs: $T_B \geq T_A + 4$
Murs extérieurs: $T_c \geq T_A + 3$
Cheminée: $T_D \geq T_A + 3$
Toit: $T_D \geq T_C + 2$
Portes: $T_E \geq T_B + 2$
Tuiles: $T_E \geq T_D + 3$
Fenêtres: $T_E \geq T_C + 3$

Est-ce que $B \geq C$ si la durée totale est de 15 jours ?
$(CH \wedge T_E = 15) \Rightarrow B \geq C$



# Stratégie de recherche naives : GT & BT

**Generate & test** :

Tant que toutes les contraintes ne sont pas satisfaites

- Nouvelle instanciation de toutes les variables
- Évaluation des contraintes

**Backtracking** :

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)*

# Backtracking vs Generate & Test : exemple

Contraintes:
$D_X = D_Y = D_Z = \{1, 2\}$, $X = Y$, $X \neq Z$, $Y > Z$

**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 |
| | | 2 | fail |
| | 2 | | fail |
| 2 | 1 | | fail |
| | 2 | 1 | succes |

# Partie II : la programmation en logique avec contraintes (CLP)

# Le traitement relationnel de l'arithmétique en programmation logique

Instruction $\equiv$ Relation

$\rightarrow$ Spécification d'une solution en terme de relations (produits cartésiens) entre entités (ou classes d'entités)

$\rightarrow$ Réversibilité
Exemple de `append/3`:

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

$\|$ concaténation mais aussi décomposition de listes

$\rightarrow$ Programme en logique $\simeq$ spécification exécutable

## Le problème de l'arithmétique en programmation logique(1)

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 Programmation en Logique avec Contraintes était de donner un statut relationnel à l'arithmétique.*

## LE PROBLEME DE L'ARITHEMETIQUE EN PL (2)

`?- 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).
```

**et** ?- 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)).
```

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

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

`?- sort(X1,X2,X3).` ⇒ FAIL

## LE PROGRAMME SEND + MORE = MONEY EN PROLOG

```
mm0([S,E,N,D,M,O,R,Y]) :-
        generate0([S,E,N,D,M,O,R,Y]),      % generate
        S>0, M >0,                         % test
        alldiff([S,E,N,D,M,O,R,Y]),        % test
        test_sum(S,E,N,D,M,O,R,Y).         % test
generate0([X|Xs]):-                        generate0([]).
        member(X,[0,1,2,3,4,5,6,7,8,9]),
        generate0(Xs).
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.
```

## LE PROGRAMME SEND + MORE = MONEY EN PROLOG (SUITE)

```
alldiff([]).
alldiff([X|L]):-
        diff(X,L),
        alldiff(L).
diff(X,[]).
diff(X,[V|L]):-
        X \= V,
        diff(X,L).
```

## LE PROGRAMME SEND + MORE = MONEY AVEC DES CONTRAINTES

```
mm([S,E,N,D,M,O,R,Y]) :-
        fd_domain([S,E,N,D,M,O,R,Y], 0, 9),      %  Définition des domaines
        S#>0, M#>0,                              % Pose
        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]).        % 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.
?- mm([S,E,N,D,M,O,R,Y]).
D = 7, E = 5, M = 1, N = 6, O = 0, R = 8, S = 9,  Y = 2
```

## LE PROGRAMME SEND + MORE = MONEY (SUITE)

**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**

## LE PROGRAMME SEND + MORE = MONEY (SUITE)

$$\mathbf{900 \times M = 1000 \times S + 91 \times E + D - 900 \times O - 90 \times N - Y}$$

D'ou

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

Et:

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

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

## LE PROGRAMME SEND + MORE = MONEY (SUITE)

**Par propagation on obtient :**

**D(S) = {9}  D(E) = {4, 5, 6, 7}  D(N) = {5, 6, 7, 8}  D(D) = {2, 3, 4, 5, 6, 7, 8}**

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

**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 $10^8$**

**La programmation en logique avec contraintes**

$=$

**théorie du premier ordre qui préserve les propriétés de calcul associées aux clauses de Horn**

$\rightarrow$ **des classes de langages de programmation en logique avec contraintes dont les variables peuvent prendre des valeurs sur des domaines variés**

# Machine Abstraite d'un langage PLC

$$(1) \qquad \sigma = (W, t_0 t_1 \ldots t_n, S)$$

$$(2) \qquad s_0 \rightarrow s_1 \ldots s_m, R \ (m \geq 0)$$

$$(3) \qquad \sigma^{'} = (W, s_1 \ldots s_m t_1 \ldots t_n, S \wedge R \wedge (s0 = t0))$$

$W$: liste des variables de la question initiale,

$t_0 t_1 \ldots t_n$: liste de termes (buts) à effacer,

$S$ : liste des contraintes courantes (contraintes satisfiables)

$s_1 \ldots s_m$ : une suite de termes

$R$ : les contraintes de la règle d'inférence

$\sigma^{'}$ représente le nouvel état de la machine après application de la règle (2) si $S \cup R \cup (s0 = t0)$ est satisfiable

**Solution du système** : $\sigma_f$ = (W,$\epsilon$,Contraintes Finales).

**Réponse d'un système CLP:**

*ensemble de contraintes satisfiables*
(éventuellement un ensemble infini de solutions,
e.g., $-1 < x < 1$)

**Apport des contraintes** :

La stratégie "generate and test" de Prolog est remplacée par une stratégie "prune and search"

$\rightarrow$ l'arbre de recherche est élagué a priori

## Résolution de systèmes de contraintes linéaires

Il existe des *algorithmes polynomiaux* pour résoudre des systèmes de contraintes linéaires sur $\mathcal{R}$

→ Les contraintes linéaires furent introduites dans des langages PLC dès le milieu des années 80

Résolution d'un système de contraintes linéaires:

- Algorithme de Gauss–Jordan
- Algorithme de Fourrier
- Simplexe (résultats justes que sur les nombres rationnels, pour les nombres flottants, le simplexe peut donner des résultats faux)

Une **expression linéaire** est de la forme

$$a_1x_1 + a_2x_2 + \ldots a_nx_n$$

avec :

- $a_i \quad (1 \leq i \leq n)$ est un nombre réel;
- $x_i \quad (1 \leq i \leq n)$ est une variable.

Une **équation linéaire** est de la forme $e_1 = e_2$ où $e_1$ et $e_2$ sont des expressions linéaires

Une **inéquation linéaire** est de la forme $e_1 < e_2$ ou $e_1 \leq e_2$ où $e_1$ et $e_2$ sont des expressions linéaires

Une **contrainte linéaire** est une conjonction d'équations et inéquations linéaires

## Algorithme de Gauss–Jordan

Le principe de cet algorithme consiste à réécrire un système d'équations linéaires en un système équivalent jusqu'à l'obtention d'une forme trivialement solvable ou insovable

L'itération de base, de l'algorithme de Gauss–Jordan est consistuée de deux opérations:

1. réécriture d'une contrainte $c$ sous la forme $x = e$ où $x$ n'apparaît pas dans $e$,
2. remplacement de $x$ par $e$ dans toutes les autres contraintes.

La complexité temporelle de cet algorithme est en $\theta(m \times n^2)$ pour un système de $m$ équations avec $n$ variables.

```
% r : constante réelle(non nulle) x : variable, c: equation
% e : expression linéaire
% C, C', S : conjonctions d'équations

1 function GAUSS-JORDAN(in C, return BOOL)
2     S ←true
3     while C ≠ empty conjonction true
4         let C = c ∧ C0
5         C ← C0
6         if c can be written as 0 = r
7             then      return false
8             else      write c as x = e (x does not occur in e)
9                 substitute e for x throughout C and S
10                S ← S ∧ x = e
11        endif
12    endwhile
```

Figure 3: Algorithme de Gauss–Jordan

# Partie III : les problèmes de satisfaction de contraintes (CSP)

# Les techniques de satisfaction de contraintes en intelligence artificielle

Parallèlement aux travaux sur les CLP qui visaient avant tout à donner un statut “logique” à l’arithmétique, les techniques CSP ont été développées en intelligence artificielle pour résoudre des problèmes combinatoires.

# PRINCIPES DE LA PROGRAMMATION AVEC CONTRAINTES

- **Contrainte**

  Une **contrainte** est une relation logique (une propriété qui doit être vérifiée) entre différentes inconnues. Elle restreint les valeurs que peuvent prendre simultanément les variables.

  Exemple : $x + 3 * y = 12$ restreint les valeurs que l'on peut affecter simultanément aux variables x et y

Les **inconnues**, appelées variables, prennent leurs valeurs dans un ensemble donné, appelé **domaine**

Une contrainte est **relationnelle, déclarative** et peut être définie en **extension** ou en **intension** :

- Énumération des tuples de valeurs de la relation : $(x = 0 \wedge y = 1) \vee (x = 0 \wedge y = 2) \vee (x = 1 \wedge y = 2)$
- Définition des propriétés mathématiques connues: $x < y$ ou encore $A \vee B \Rightarrow \neg C$

- **Types de contraintes :**
  - Les contraintes numériques, entières ou réelles (linéaires ou non linéaires)
  - Les contraintes ensemblistes, contraintes booléennes, univers de Herbrandt
  - ...

# Définition d'un CSP

Un CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R})$ est défini par :

- Un ensemble de *variables* $\mathcal{X} = \{x_1, \ldots, x_n\}$
- Un ensemble de *domaines* $\mathcal{D} = \{D_1, \ldots, D_n\}$ où $D_i$ est le domaine associé à la variable $x_i$
- Un ensemble de *contraintes* $\mathcal{C} = \{C_1, \ldots, C_m\}$ où chaque contrainte $C_i$ est définie sur un ensemble de variables $\{x_{i_1}, \ldots, x_{i_p}\} \subset \mathcal{X}$
- Un ensemble de *relations* $\mathcal{R} = \{R_1, \ldots, R_m\}$ où chaque relation $R_i$ est la projection des solutions de la contrainte $C_I$ sur le domaine de ses variables; $R_i$ est un sous ensemble du produit cartésien $Dx_{i_1} \times \ldots \times D_{x_{i_p}}$.

**Coloriage de graphe:**

- $\mathcal{X} = \{x_1, x_2, x_3\}$
- $D_1 = D_2 = D3 = \{rouge, vert\}$
- $\mathcal{C} = \{x_1 \neq x2, x_1 \neq x_3\}$
- $R_1 = R_2 = R_3 = \{< rouge, vert >, < vert, rouge >\}$

Solutions :

$< x_1 = rouge, x_2 = x_3 = vert >, < x_1 = vert, x_2 = x_3 = rouge >$

**Problème des $n$ reines avec $n = 4$:**

$\mathcal{X} = \{x_1, x_2, x_3, x_4\}$

$D_1 = D_2 = D_3 = D_4 = \{1, 2, 3, 4\}$

$\mathcal{C} = \{x_i \neq x_j$ pour $1 \leq i < n, x_i \neq x_j + (j - i)$ pour $1 \leq i < j \leq n, x_i \neq x_j - (j - i)$ pour $1 \leq i < j \leq n\}$

Solutions : $< 2, 4, 1, 3 >, < 3, 1, 4, 2 >$

## 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*

## Notion de CSP surcontraint ou sous-contraint

- Un CSP qui n'a pas de solution est **surcontraint** $\rightarrow$ max-CSP (préférences entre les contraintes)
- Un CSP qui admet beaucoup de solutions différentes est **sous-contraint**

## Exemple : modélisation des n-reines

**Les *inconnues*** : les positions des reines sur l'échiquier

En numérotant les lignes et les colonnes de l'échiquier :

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

-> association à chaque reine *i* de deux variables *Li* et *Ci* correspondant respectivement à la ligne et la colonne sur laquelle placer la reine

….. ou une variable Xi par colonne dont la valeur sera la position de la reine (numéro de ligne )

# Exemple : modélisation des n-reines (suite)

## Les contraintes

- les reines doivent être sur des lignes différentes:
  $\mathbf{X_i} \neq \mathbf{X_j}$ pour tout $\mathbf{i}$ dans $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}$
- Les reines doivent être sur des colonnes différentes : **aucune contrainte** avec la deuxième modélisation !
- les reines doivent être sur des diagonales différentes :
  $\mathbf{X_i} \neq \mathbf{X_j} + / - (\mathbf{i} - \mathbf{j})$pour $\mathbf{i}, \mathbf{j}$ dans $\{\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}\}$ et $\mathbf{i} \neq \mathbf{j}$

## Résolution d'un CSP

La résolution d'un CSP est un problème NP-complet dans le cas général. Le mécanisme de résolution consiste (schématiquement) à répéter les deux étapes:

- Réduction des problèmes par *filtrage,*
- Parcours de l'arbre de recherche.

La réduction des problèmes par filtrage repose sur des propriétés de consistance (ou cohérence) partielle: degré de compatibilité entre valeurs des domaines et contraintes.

# Résolution d'un CSP (suite)

## Filtrage par consistance locale et utilisation d'heuristiques

**Principes:**

- **Élimination a priori** des valeurs qui ne peuvent être dans aucune solution (travail local à chaque contrainte)

- **Choix des variables et valeurs** (lors de l'énumération) qui devraient permettre de découvrir rapidement les impasses

# Consistance d'arc

**Consistance d'arc**: un domaine $D_x$ est consistant d'arc ssi pour chaque contrainte de $\mathcal{C}$, toute valeur de $D_x$ a un support dans les domaines des autres variables de cette contrainte. Plus formellement:

$$\forall C_i \in \mathcal{C} \text{ tel que } Var(C_i) = (x, x_1, \ldots, x_n, ) \quad \forall v \in D_x$$

$$\exists v_1 \times \ldots \times v_k \in D_1 \times \ldots \times D_k \text{ tel que}$$
$$< v, v_1, \ldots, v_k > \in R_i$$

Un CSP est consistant d'arc si tous ses domaines sont consistants d'arc:

$$\forall C_k \in \mathcal{C}, \forall x_i \in Var(C_k), \Pi_{x_i}(R_k) = D_i \neq \emptyset$$

# Consistance d'arc (suite)

- La consistance d'arc est une *consistance locale* qui **ne garantit pas l'existance d'une solution**

- Le filtrage par consistance d'arc consiste à **éliminer les éléments qui ne peuvent trivialement figurer dans aucune solution** (valeurs ou n-uplets de relations).
  Il s'agit d'une application:
  $P = (\mathcal{X}, \mathcal{D}, \mathcal{C}, \mathcal{R}) \longrightarrow P' = (\mathcal{X}, \mathcal{D}', \mathcal{C}', \mathcal{R})$

# Consistance d'arc (suite)

**IN**(**in** $\mathcal{C}$, **inout** $\overrightarrow{\mathcal{D}}$)

Queue $\leftarrow \mathcal{C}$ ;

**while** Queue $\neq \emptyset$

$c \leftarrow$ POP Queue;

$\mathcal{D}' \leftarrow$ narrow$(c, \mathcal{D})$;

**if** $\mathcal{D}' \neq \mathcal{D}$ **then** $\mathcal{D} \leftarrow \mathcal{D}'$;

Queue $\leftarrow$Queue $\cup \{c' \in \mathcal{C} \mid Var(c) \cap Var(c') \neq \emptyset\}$

**endif**

**endwhile**

Figure 4: Algorithme générique de filtrage AC3

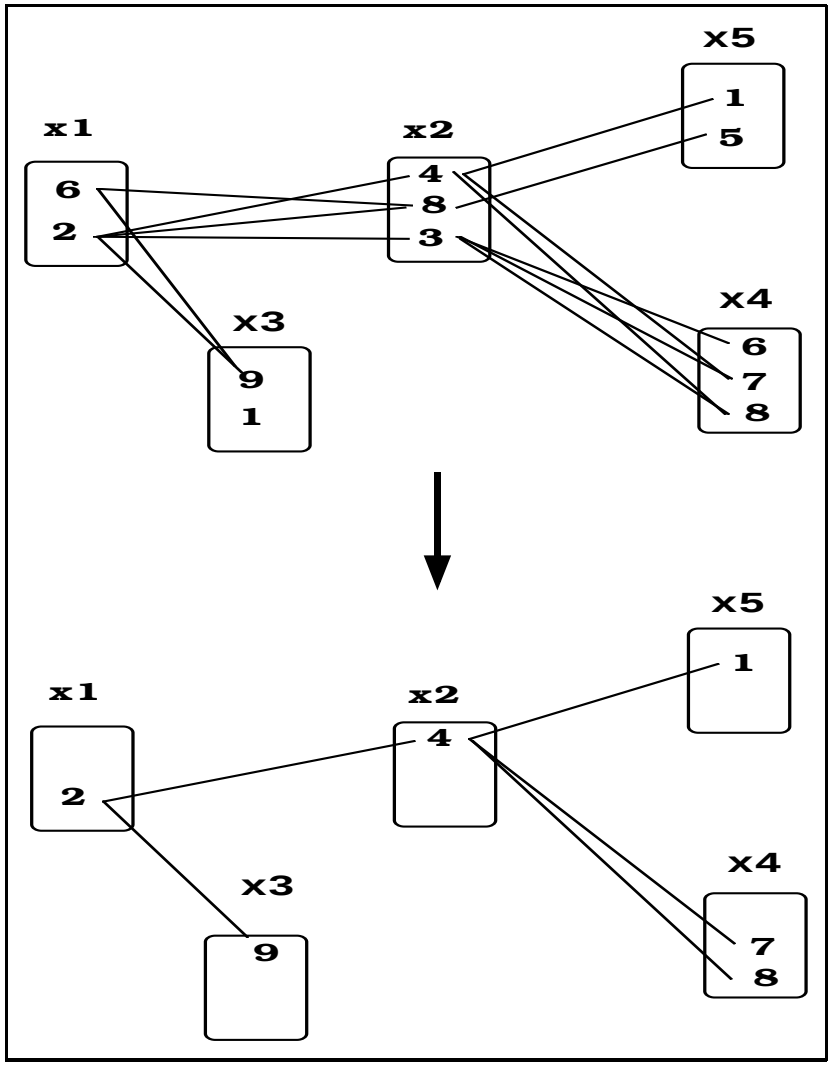
# AC3 (suite)

**Exemple**

Soit le système suivant:

$$C = \{x_1 < x_2, x_5 + 3 = x_2, x_3 > x_1, x_4 \geq x_2 + 3\}$$
$$d_{x_1} = \{2, 6\}, d_{x_2} = \{3, 4, 8\},$$
$$d_{x_3} = \{1, 9\}, d_{x_4} = \{6, 7, 8\}, d_{x_5} = \{1, 5\}$$

# Illustration du filtrage par AC3



# Limites de la consistance d'arc

L'exemple ci-dessous est consistant d'arc alors qu'il ne possède pas de solution:

- $\mathcal{X} = \{x_1, x_2, x_3\}$
- $D_1 = D_2 = D3 = \{rouge, vert\}$
- $\mathcal{C} = \{x_1 \neq x2, x_1 \neq x_3\, x_2 \neq x_3\}$

# Contrainte globale $\rightarrow$ algorithmes de filtrage efficaces pour des contraintes spécifiques

**Exemple: *alldiff***

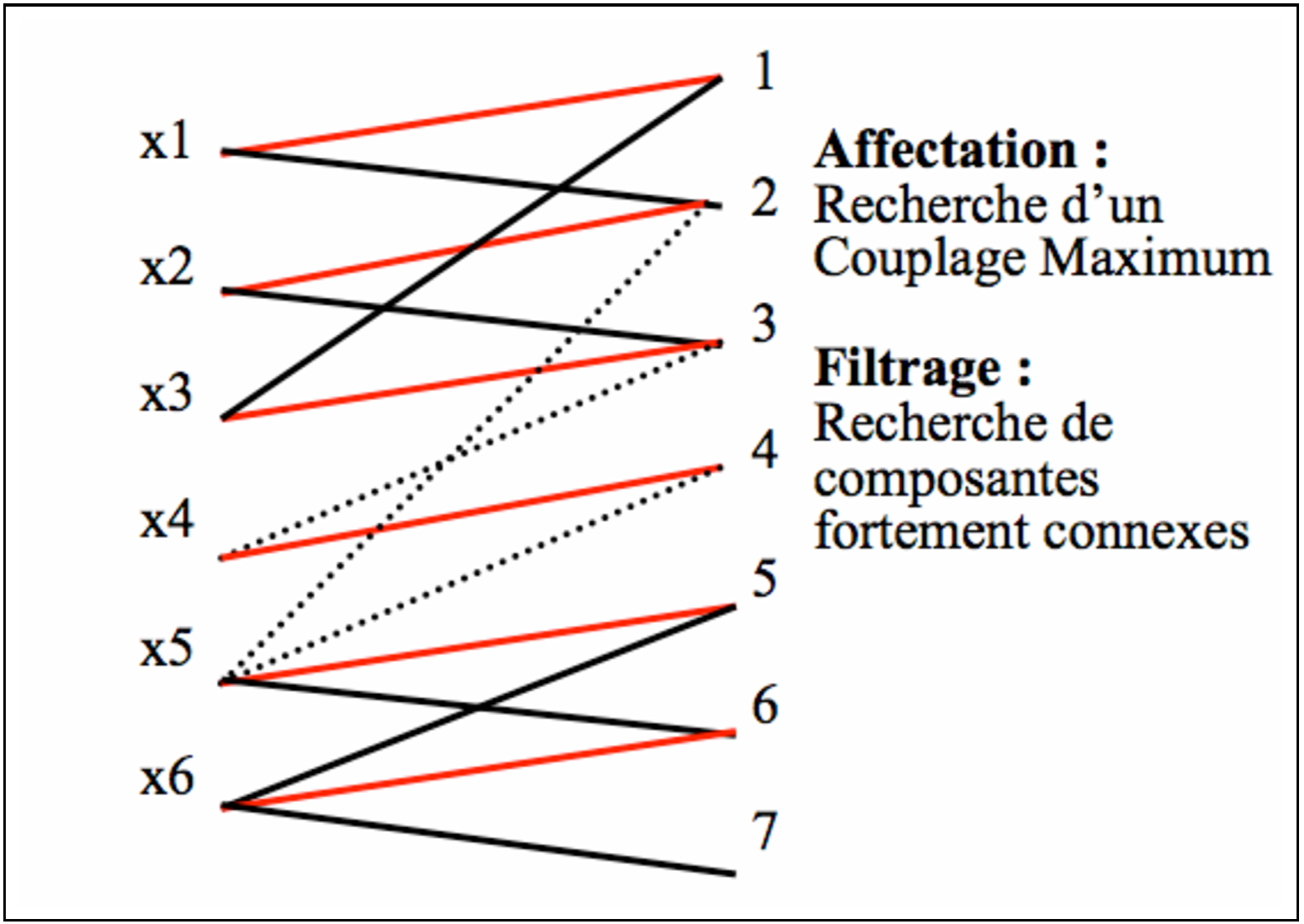
Les variables doivent prendre deux à deux des valeurs différentes

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

$D(x_1) = \{1, 2\}$ $D(x_2) = \{2, 3\}$ $D(x_3) = \{1, 3\}$

$D(x_4) = \{3, 4\}$ $D(x_5) = \{2, 4, 5\}$ $D(x_6) = \{5, 6, 7\}$

# Contraintes Globales (suite)



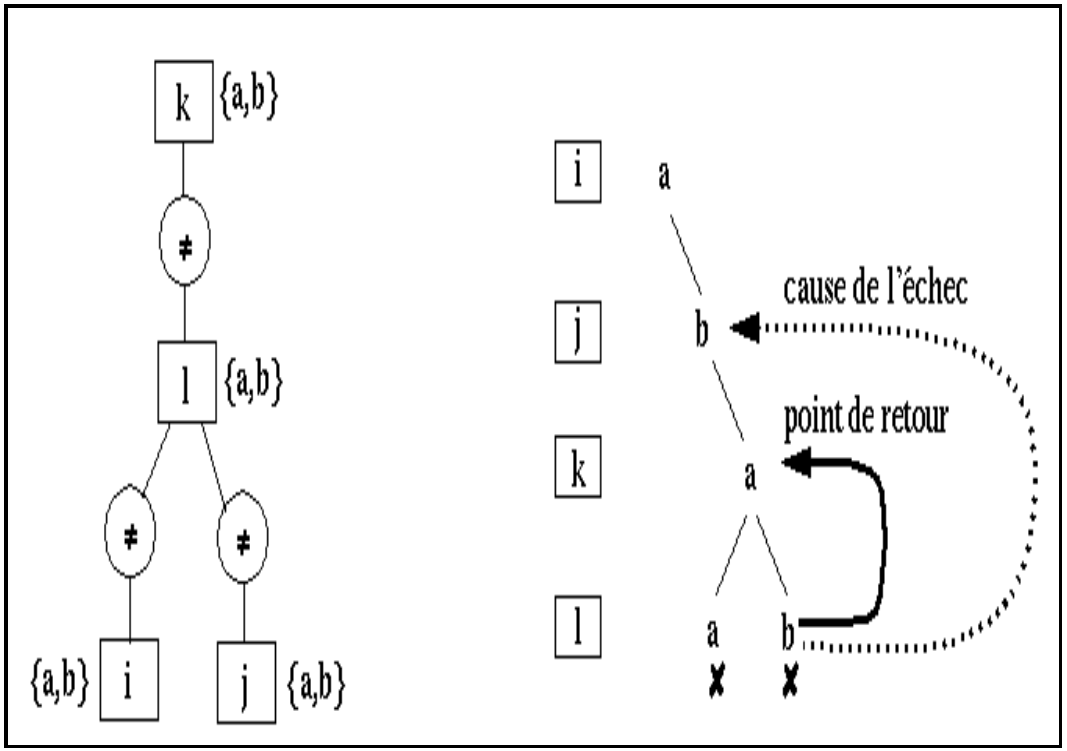
## Parcours de l'arbre de recherche

- **Heuristiques** qui permettent dans de nombreux cas d'éliminer a priori certaines branches
- Combinaison filtrage & heuristiques:
  "forward checking", "looking ahead" ou "MAC" effectuent des opérations de filtrage après chaque étape de recherche (i.e., chaque instanciation de variable).

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

# Défauts du backtrack chronologique

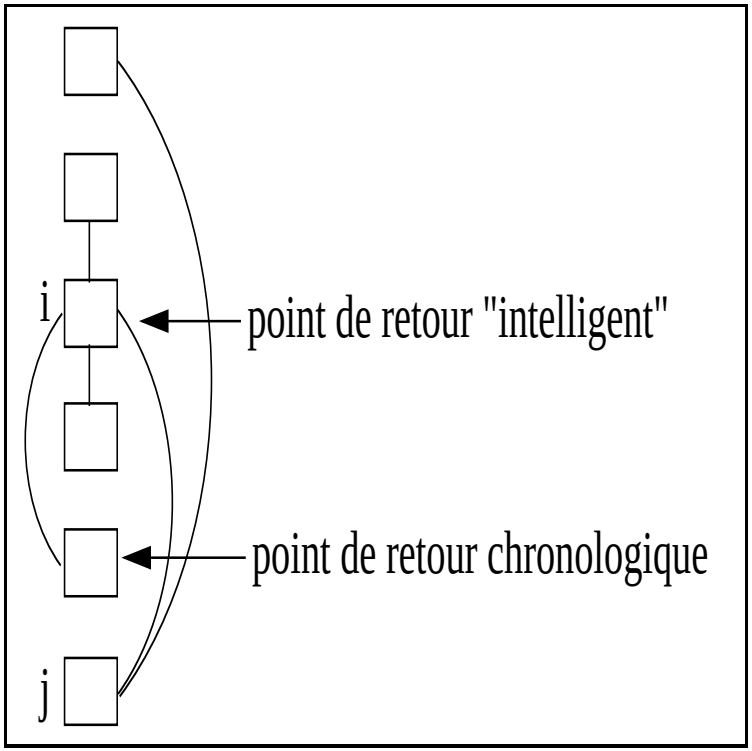
Le backtrack chronologique peut entrainer la répétition d'échecs causés par une même raison.



# Backtrack dirigé par le graphe de contraintes : GBJ

En cas d'échec sur une variable $j$, retour sur la variable $i$ connectée à $j$ par une contrainte et la plus récemment instanciée.

Intéressant pour les graphes de contraintes peu denses.



# Partie IV : Les contraintes dans le domaine continu — la propagation d'intervalles

1. Arithmétique des Intervalles
2. 2–B–Consistance et 2–B($w$)–Consistance
3. 3B–Consistance
4. Box–Consistance et Bound–Consistance

La propagation d'intervalles est un cas particulier de CSP dont la spécificité réside essentiellement au niveau du domaine des variables qui représentent ici **un ensemble infini de valeurs**

$\rightarrow$ Problèmes définis sur des variables réelles par un ensemble d'équations et d'inéquations non linéaires voire non polynomiales

## Problèmes sur variables réelles

1. **Problèmes physiques**
   - Données imprécises → problèmes sous-contraints (infinité de solutions)
   - Contraintes non linéaires → il n'existe pas d'algorithme symbolique efficace
2. **Exemple**

   $U = R \times I$

   $10.5 \leq R \leq 11 \qquad \rightarrow 10.5 \leq U \leq 22$

   $1 \leq I \leq 2$

## Propagation d'intervalles : caractéristiques

- Raisonnement numérique sur les domaines de variation (une solution numérique est plus pertinente qu'une formule algébrique dans de nombreuses applications)
- Approximation des domaines de variation
- Réduction de domaines à l'aide de consistances partielles
- Simplification, efficacité sur les problèmes en variables entières

# 1 Arithmétique des Intervalles

## Notations

- $x, y, z$ : variables définies sur les réels ; $X, Y, Z$ : variables définies sur les intervalles ;
- $u, v$ : constantes dans $\mathcal{R}$ ;
- $f, g$ : fonctions définies sur les réels ;
- $F, G$ : fonctions définies sur les intervalles ;
- $c$ : contrainte définie sur les réels ;
- $C$ : relation définie sur les intervalles ;
- $\Phi_{cstc}(P)$ : fermeture (ou le filtrage) par la consistance $cstc$ (où $cstc$ est $2B$, $Box$, $3B$, $Bound$) de $P$ .

# **Notations** (suite)

- $\mathcal{R}^{\infty} = \mathcal{R} \cup \{-\infty, +\infty\}$ : ensemble des réels étendu aux symboles des infinis.
- $\overline{\mathbb{F}}$ : un sous-ensemble fini de $\mathcal{R}^{\infty}$ contenant en particulier $\{-\infty, +\infty\}$ ;
  ($\overline{\mathbb{F}}$ : un ensemble de nombres flottants utilisés dans une implémentation machine)
- $a, b$ : constantes dans $\overline{\mathbb{F}}$

- $a^{+}$ (resp. $a^{-}$) : plus petit (resp. plus grand) nombre de $\overline{\mathbb{F}}$ strictement plus grand (resp. plus petit) que $a$.

## Bases du calcul d'intervalles

- ◇ Un **intervalle** $[a, b]$ avec $a, b \in \overline{\mathbb{F}}$ est l'ensemble de nombres réels $\{r \in \mathcal{R} \mid a \leq r \leq b\}$

- ◇ $\tilde{r}$ : plus petit intervalle de $\overline{\mathbb{F}}$ contenant un nombre réel $r$

- ◇ $\square S$ est le plus petit intervalle $I$ tel que $S \subseteq I$
  ($S$ est un sous ensemble de $\mathcal{R}$)

- ◇ $\mathcal{I}$ : ensemble de tous les intervalles
- ◇ $\mathcal{U}(\mathcal{I})$ : ensemble de toutes les unions d'intervalles

# Incidence de la précision des calculs

Le terme plus petit (w.r.t. inclusion) sous-ensemble doit être interprêté en fonction de la précision des calculs lors des opérations en virgule flottante.

**Hypothèses :**

1. Tous les résultats des calculs sont arrondis vers l'extérieur
2. Erreur maximale lors du calcul d'une borne d'un intervalle est toujours inférieure à un flottant
   (peut nécessiter de recourir à des "grands flottants")

# Extension aux intervalles

• Une **fonction** sur les intervalles $F : \mathcal{I}^n \rightarrow \mathcal{I}$ est une extension aux intervalles de $f : \mathcal{R}^n \rightarrow \mathcal{R}$ ssi :

$$\forall I_1, \ldots, I_n \in \mathcal{I} : r_1 \in I_1, \ldots, r_n \in I_n \Rightarrow f(r_1, \ldots, r_n) \in F(I_1, \ldots, I_n).$$

• Une **relation** sur les intervalles $C : \mathcal{I}^n \rightarrow \mathcal{B}ool$ est une extension aux intervalles de la relation $c : \mathcal{R}^n \rightarrow \mathcal{B}ool$ ssi :

$$\forall I_1, \ldots, I_n \in \mathcal{I} : r_1 \in I_1, \ldots, r_n \in I_n \Rightarrow [c(r_1, \ldots, r_n) \Rightarrow C(I_1, \ldots, I_n)]$$

**Exemple :**

Soit $I_1 \doteq I_2 \Leftrightarrow (I_1 \cap I_2) \neq \emptyset$

La relation $\doteq$ est une extension aux intervalles de la relation d'égalité sur les réels.

# Extension naturelle aux intervalles

Une fonction sur les intervalles $F : \mathcal{I}^n \rightarrow \mathcal{I}$ est une extension naturelle aux intervalles de $f : \mathcal{R}^n \rightarrow \mathcal{R}$ ssi $F$ est l'extension aux intervalles de $f$ obtenue en remplaçant dans $f$ :

- ◇ chaque constante $k$ par son extension naturelle aux intervalles $\tilde{k}$
- ◇ chaque variable par une variable sur les intervalles
- ◇ chaque opération arithmétique par son extension optimale aux intervalles (opération qui calcule le plus petit intervalle qui conserve l'ensemble des solutions)

## Extensions optimales aux intervalles pour les opérations de base sur les intervalles (Moore)

Soit $\odot$ un opérateur binaire parmi $\{+, -, *, /\}$ et
$[a, b] \odot [c, d] = \{x \odot y$ tel que $a \leq x \leq b$ et $c \leq y \leq d\}$ alors :

- $[a, b] \ominus [c, d] = [a - d, b - c]$
- $[a, b] \oplus [c, d] = [a + c, b + d]$
- $[a, b] \otimes [c, d] = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]$
- $[a, b] \oslash [c, d] = [min(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}), max(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d})]$ *if* $0 \notin [c, d]$

◇ Les extensions optimales aux intervalles peuvent être définies pour quasiment toutes les fonctions élémentaires

◇ **la forme syntaxique d'une fonction a une très forte influence** sur la précision de l'extension naturelle aux intervalles

# Extension naturelle aux intervalles : exemples

Soit les fonctions sur les réels :

- $f_1(x) : x^2 - x$
- $f_2(x) : x(x-1)$

Extension naturelle aux intervalles :

- $F_1([0,5]) = [-5, 25]$
- $F_2([0,5]) = [-5, 20]$

Valeur de l'extension optimale aux intervalles de $f_1$ et $f_2$ sur $[0,5]$
est $[-0.25, 20]$
(la dérivée de $f_1$ et $f_2$ s'annulle pour $x = 0.5$).

# Limites de la sous-distributivité

**Exemple** :

a) $[1,2] * ([1,2] - [1,2]) = [-2,2]$

$[1,2] * [1,2] - [1,2] * [1,2] = [-3,3]$

b) $f_1 : \mathcal{R} \rightarrow \mathcal{R}$ $\qquad$ $F_1 : I \rightarrow I$

$x \rightarrow x - x$ $\qquad$ $i \rightarrow i - i$

Si $I_0 = [10,20]$ alors

$F_1(I_0) = [-10,10]$

# CSP sur les domaines continus

- $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ où $\mathcal{X} = \{x_1, \ldots, x_n\}$ est un ensemble de variables
- $\mathcal{D} = \{D_{x_1}, \ldots, D_{x_n}\}$ est un ensemble de domaines ($D_{x_i}$ est le domaine contenant toutes les valeurs potentielles de la variable $x_i$)
- $\mathcal{C} = \{c_1, \ldots, c_m\}$ est un ensemble de contraintes

On note $Var(C)$ le sous ensemble de $\mathcal{X}$ des variables de $C$.

## Approximation d'un CSP

Il est en général impossible de calculer toutes les solutions d'un CSP $\rightarrow$ calcul d'une approximation de cet ensemble de solutions.

Soit $A$ un domaine d'approximation sur $\mathcal{R}$, $\rho$ une relation n-aire sur $\mathcal{R}$. La fonction $N : A^n \rightarrow A^n$ est un opérateur de narrowing pour la relation $\rho$ ssi pour tout $u, v \in A^n$ on a :

1. $N(u) \subset u$ (*contractance*)
2. $u \cap \rho \subset N(u)$ (*correction*)
3. $u \subset v \Rightarrow N(u) \subset N(v)$ (*monotonie*)

## Algorithme standard de *narrowing*

1 **IN–1 (in** $\mathcal{C}$**, inout** $\mathcal{D}$**)**
2 $Q \leftarrow \{< x_i, C_j > \mid C_j \in \mathcal{C}$ and $x_i \in Var(C_j)\}$
3 **while** $Q \neq \emptyset$
4 extract $< x_i, C_j >$ from $Q$
5 $\mathcal{D}' \leftarrow$ `narrowing`$(\mathcal{D}, x_i, C_j)$
6 **if** $\mathcal{D}' \neq \mathcal{D}$ **then**
7 $\mathcal{D} \leftarrow \mathcal{D}'$
8 $Q \leftarrow Q \cup \{< x_l, C_k > |(x_l, x_i) \in Var(C_k)\}$
10 **endif**
11 **endwhile**

## Consistances partielles

◇ les consistances **strictement locales**
*ne travaillent que sur une seule contrainte*

◇ les consistances **partielles** qui ne sont pas strictement locales
→ vérifient certaines propriétés sur un sous-ensemble du système de contraintes

# 2 2–B–Consistance et 2–B($w$)–Consistance

*La 2–B–Consistance : approximation de la consistance d'arc*

**Intuitions** :

◇ La contrainte $c$ est 2B–Consistante pour la variable $x$, de domaine $D_x = [a, b]$, s'il existe des valeurs dans les domaines de toutes les autres variables qui satisfont $c$ lorsque $x$ est instancié avec *a* et lorsque $x$ est instancié avec $b$.

◇ si $c$ peut s'écrire sous la forme $f(x) = 0$, alors $a$ et $b$ sont respectivement le plus petit et le plus grand zéro de $f(x)$

# 2B–Consistance

Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et $c \in \mathcal{C}$ une contrainte k-aire sur les variables $(x_1, \ldots, x_k)$. La contrainte $c$ est 2B–Consistante ssi :

$$\forall i, D_{x_i} = \square\{v_i \in D_{x_i} \mid \exists v_1 \in D_{x_1}, \ldots, \exists v_{i-1} \in D_{x_i-1}, \exists v_{i+1} \in D_{x_i+1}, \ldots \exists v_k \in D_{x_k} \text{ tel que } c(v_1, \ldots, v_{i-1}, x_i, v_{i+1} \ldots, v_k)\}$$

Un CSP est 2B–Consistant ssi toutes ses contraintes sont 2B–Consistantes

## Consistance d'arc versus 2–B–Consistance

La 2B–Consistance est une consistance plus faible que la consistance d'arc :

- ◇ La consistance d'arc impose une condition sur tous les éléments des domaines
- ◇ La 2–B–Consistance impose une condition aux seules bornes de l'intervalle qui contient les domaines

**Exemple**

$P = \{\{x, y\}, \{D_x, D_y\}, \{x = y^2\}\}$ avec $D_x = [1, 4],\ D_y = [-2, +2]$

$P$ est 2–B–Consistant, mais pas consistant d'arc car la valeur 0 de $D_y$ n'a pas de support dans $D_x$

## Calcul de la 2–B–Consistance

$\rightarrow$ Approximation des fonctions de projection (utilisées dans `narrowing`$(\mathcal{D}, x, C)$ de `IN-1`)

$\diamond$ la portée des vérifications effectuées par la 2B–Consistante est réduite par la décomposition : si une variable $x$ possède des occurrences multiples dans une contrainte $c \in \mathcal{C}$, alors les différentes occurrences de $x$ dans $decomp(c)$ pourront varier indépendamment les unes des autres
$\rightarrow$ le filtrage par 2B–Consistance de $decomp(\mathcal{C})$ sera plus faible que celui de $\mathcal{C}$

## Algorithme de calcul de la 2–B–Consistance

$\rightarrow$ Calcul des fonctions extremum dans la fonction `narrowing` de l'algorithme `IN-1`

1 **function narrow** $(\mathcal{D}, x_i, C_j)$ : set of domains
2 $\quad m \leftarrow Min_{x_i}(C, D_{x_i})$
3 $\quad M \leftarrow Max_{x_i}(C, D_{x_i})$
4 $\quad$ **return** $\mathcal{D}[D_{x_i} \leftarrow [m, M]]$

FIG. 1 – Fonction `narrowing` pour le calcul de la 2–B–Consistance

## Arrêt prématuré de l'algorithme de propagation

En cas de convergences asymptotiques, il n'est pas réaliste de vouloir réduire tous les intervalles jusqu'à ce qu'aucun élément de $\overline{\mathbb{F}}$ (i.e., aucun flottant) ne puisse être enlevé

$\rightarrow$ **arrêt de la propagation** avant d'atteindre le point fixe.

**Exemple 2.1.** *Soit :*

$X = 2 \times Y$

$Y = X$

$D_X = [-10, 10], D_Y = [-10, 10]$

*La 2B-consistance va opérer les réductions suivantes :*

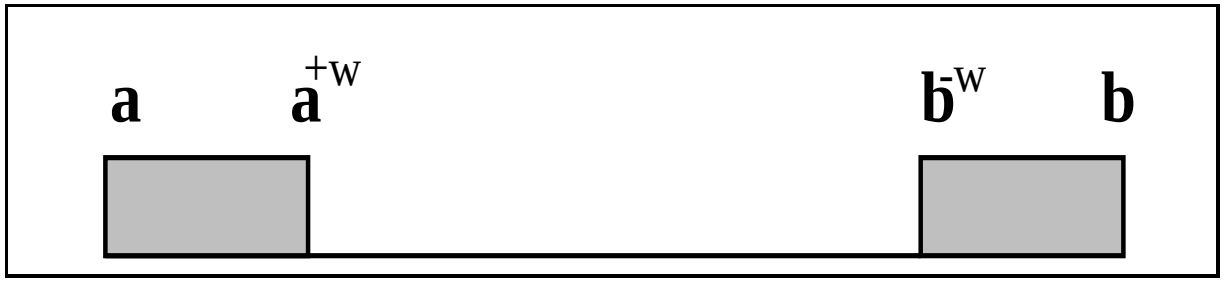
| | |
|---|---|
| $D_Y = [-5, 5]$ | $D_X = [-5, 5]$ |
| $D_Y = [-2.5, 2.5,$ | $D_X = [-2.5, 2.5]$ |
| $D_Y = [-1.25, 1.25]$ | $D_X = [-1.25, 1.25]$ |
| $D_Y = [-0.625, 0.625]$ | $D_X = [-0.625, 0.625]$ |
| ..... | |

*il est préférable d'arrêter la propagation avant d'atteindre le point fixe ...*

# "Largeur" de borne

$a^{+w}$ dénote le $(w+1)^{ieme}$ floattant après $a$.

$a^{-w}$ dénote le $(w+1)^{ieme}$ floattant avant $a$.



# 2B(w)–Consistance

Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP, $x \in \mathcal{X}$, $D_x = [a, b]$, $w$ un entier positif. $D_x$ est 2–B($w$)–Consistant si pour toute contrainte $c(x, x_1, \ldots, x_k)$ de $\mathcal{C}$ :
1) $\exists v \in [a, a^{+w}),\ \exists v_1, \ldots, v_k \in D_{x_1} \times \cdots \times D_{x_k} \mid c(v, v_1, \ldots, v_k)$
2) $\exists v' \in (b^{-w}, b],\ \exists v'_1, \ldots, v'_k \in D_{x_1} \times \cdots \times D_{x_k} \mid c(v', v'_1, \ldots, v'_k)$

Un CSP est 2–B($w$)–Consistant ssi tous ses domaines sont 2–B($w$)–Consistants

Algorithme de filtrage par 2B(w)–Consistance : modification de la fonction `Narrow` pour que les réductions de domaine effectuées soient supérieures à $w$.

## Problèmes de la 2B(w)–Consistance

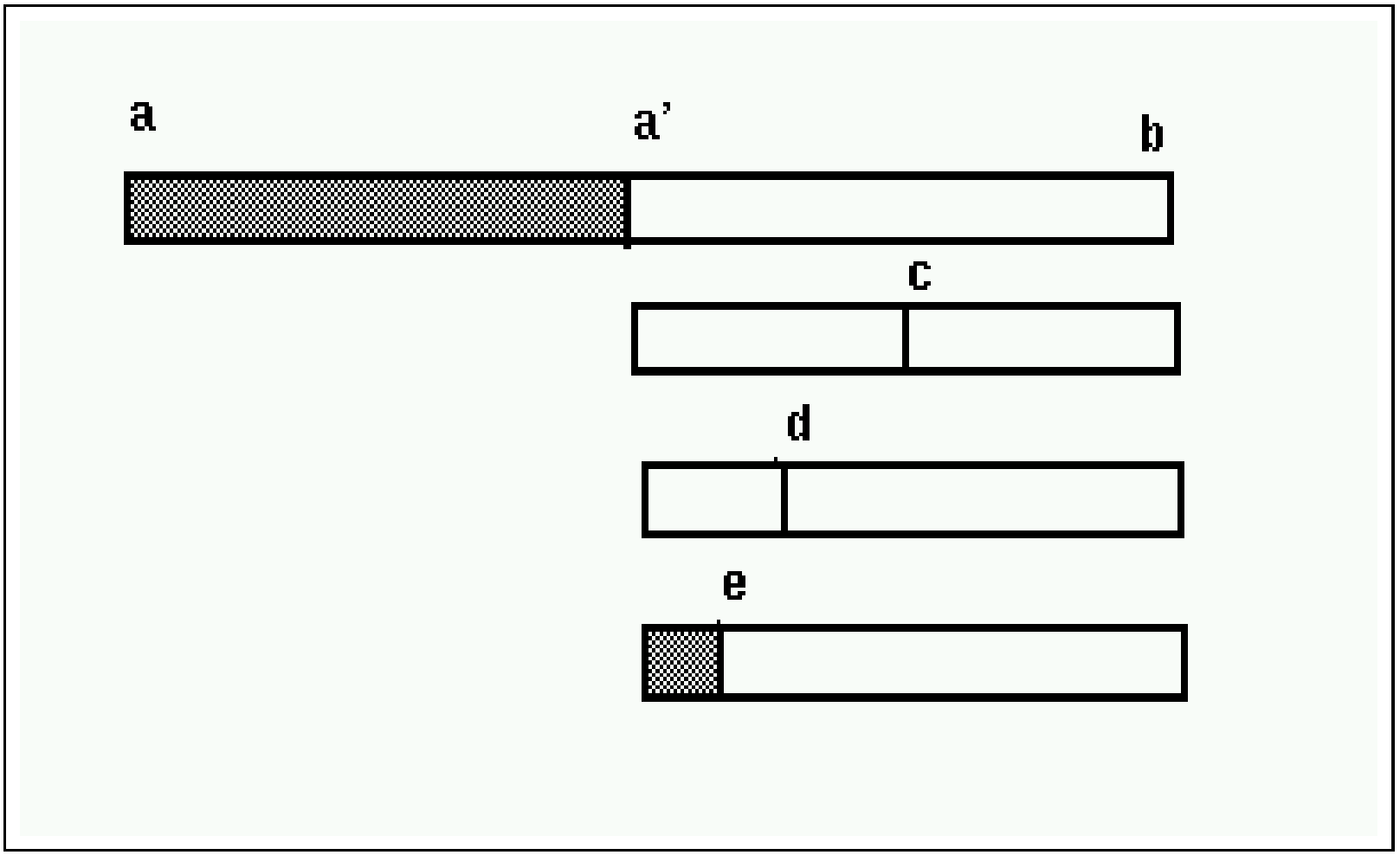
- ◇ Le filtrage par 2–B($w$)–Consistance dépend de l'ordre d'évaluation des fonctions de projection (pas de point fixe)
- ◇ Il n'y a pas de rapport direct entre la valeur de $w$ et la précision du filtrage
- ◇ Le mode de décomposition d'un système de contraintes influence l'ordre d'évaluation des fonctions de projection et peut donc affecter le résultat d'un filtrage par 2B($w_1$)–Consistance
- ◇ Si les nombres manipulés ont des valeurs très hétérogènes, il est indispensable d'utiliser une valeur relative pour $w$

# 3 3B–Consistance

Principe de la 3B–Consistance : vérifier si le système de contraintes ne devient pas trivialement inconsistant lorsqu'une variable est instanciée par la valeur d'une des bornes de l'intervalle qui lui est associée

$\rightarrow$ la 3B–Consistance cherche à vérifier si une borne peut effectivement être solution

# 3B–Consistance : intuition



# 3B–Consistance (définition)

Soit $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et $x$ une variable de $\mathcal{X}$ de domaine $[a, b]$ et soit :

- $P_{D_x^1 \leftarrow [a, a^+)}$ le CSP derivé de $P$ en substituant $D_x$ dans $\mathcal{D}$ par $D_x^1 = [a, a^+)$ ;
- $P_{D_x^2 \leftarrow (b^-, b]}$ le CSP derivé de $P$ en substituant $D_x$ dans $\mathcal{D}$ par $D_x^2 = (b^-, b]$.

$D_x$ est 3B–Consistant ssi $\Phi_{2B}(P_{D_x^1}) \neq P_\emptyset$ et $\Phi_{2B}(P_{D_x^2}) \neq P_\emptyset$.

Un CSP est 3–B–Consistant ssi tous ses domaines sont 3–B–Consistants

**k–B–Consistance** : généralisation de la 3–B–Consistance

## Algorithme de filtrage par 3B–consistance = une suite de tentatives de refutation

Soit $D_x = [a, b]$ dans un CSP $P$, si $\Phi_{3B}(P_{D_x \leftarrow [a, \frac{a+b}{2}]}) = \emptyset$,

- alors la portion $[a, \frac{a+b}{2})$ de $D_x$ est éliminée et le filtrage se poursuit sur l'intervalle $[\frac{a+b}{2}, b]$ ;
- sinon le filtrage se poursuit avec l'intervalle $[a, \frac{a+b}{4}]$.

L'algorithme s'arrête lorsque le point fixe est atteint ou lorsque l'intervalle qu'on essaye de refuter devient plus petit qu'une valeur absolue (ou relative) $w$ fixée.

# 4 La Box–Consistance et Bound–Consistance

– La Box–Consistance est une approximation plus grossière de la consistance d'arc que la 2B–Consistance

– La Box–Consistance génère un système de fonctions univariables qui peut être résolu par la méthode de Newton

– La Box–Consistance n'amplifie pas le problème de localité des consistances partielles

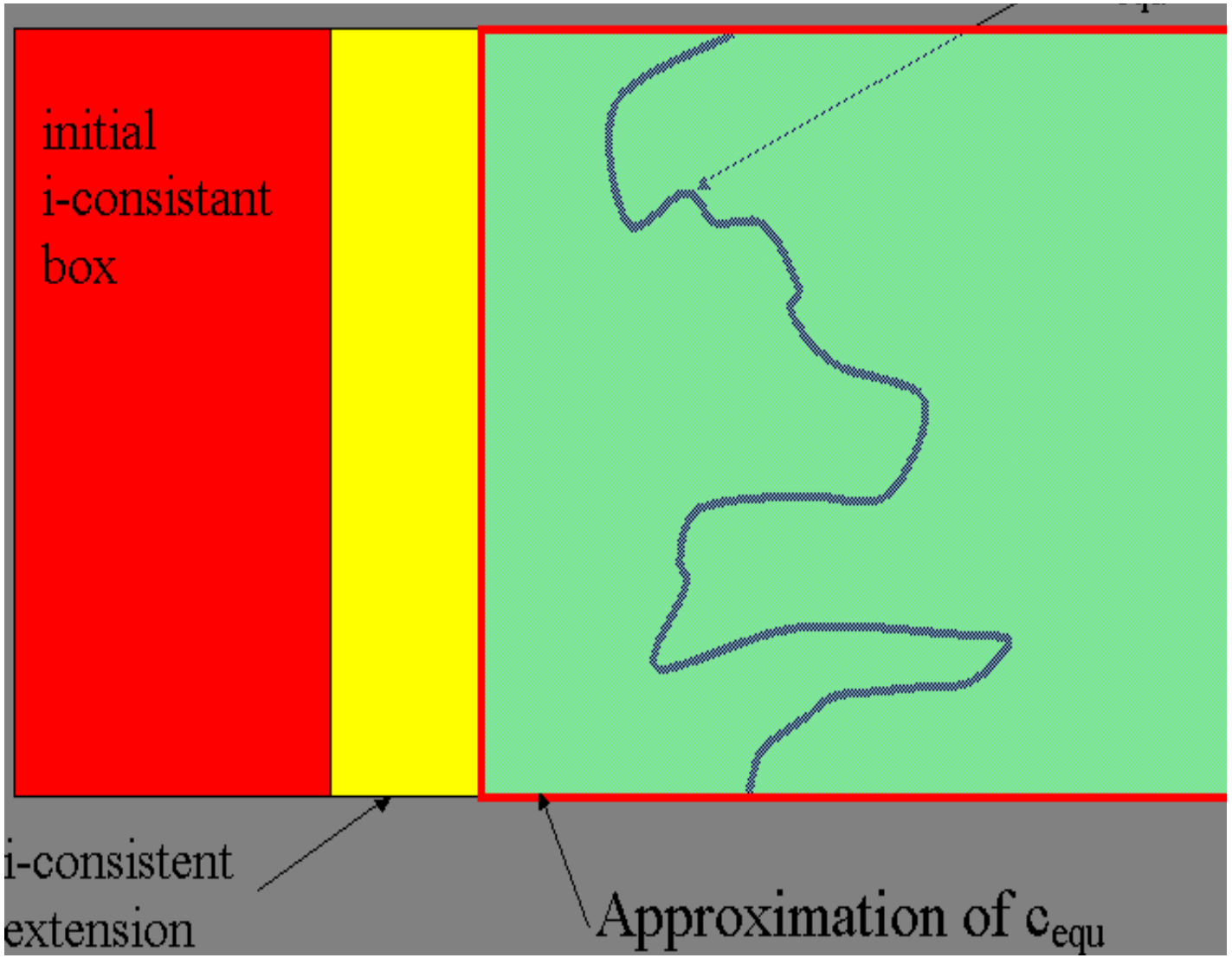
## Définition et propriétés de la Box–Consistance

Soit $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ un CSP et $c \in \mathcal{C}$ une contrainte k–aire définie sur les variables $(x_1, \ldots, x_k)$. $c$ est Box–Consistant si, pour tout $x_i$ dans $\{x_1, \ldots, x_k\}$ tel que $D_{x_i} = [a, b]$, les relations ci-dessous sont vérifiées :

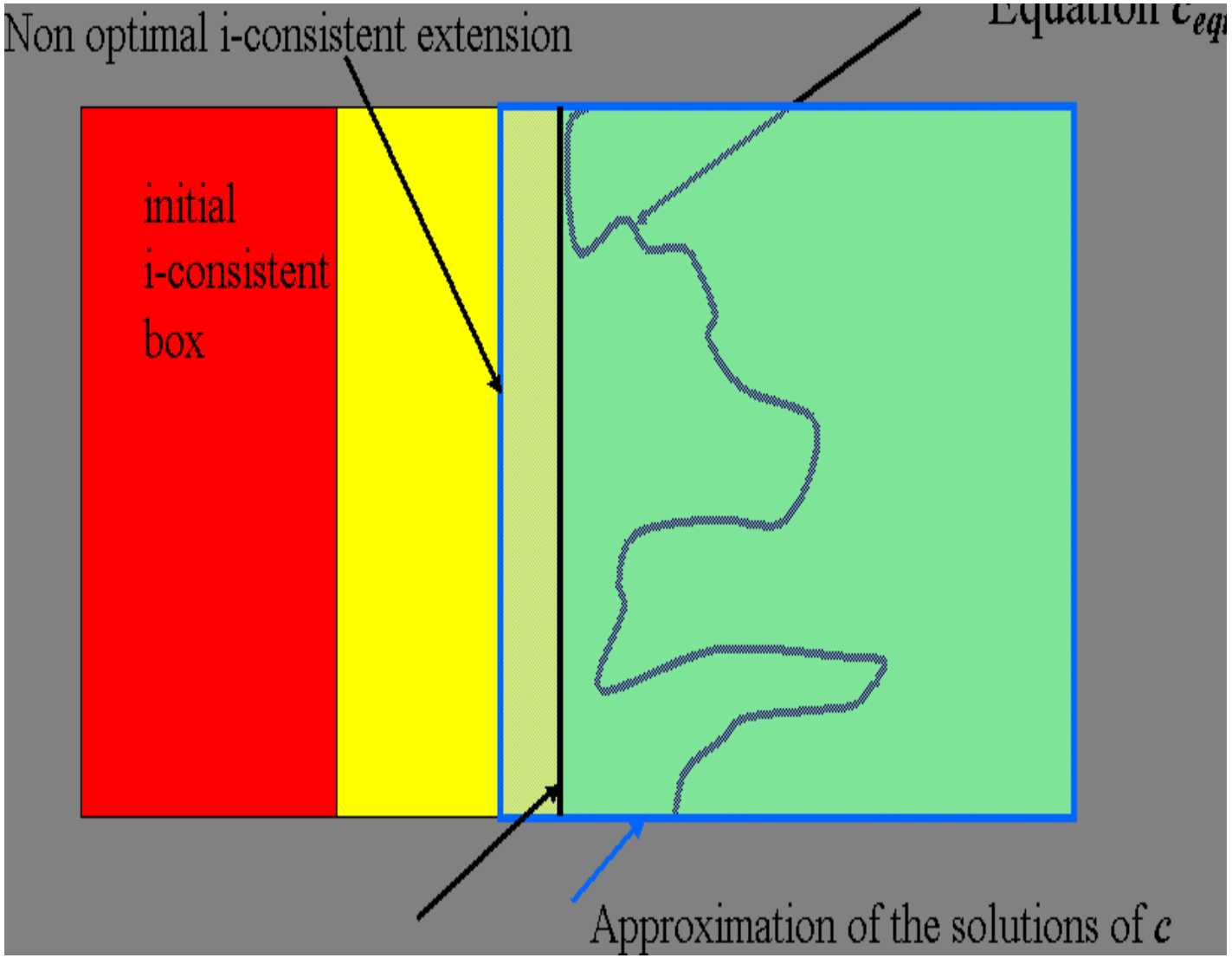
1. $C(D_{x_1}, \ldots, D_{x_{i-1}}, [a, a^+), D_{x_{i+1}}, \ldots, D_{x_k})$,
2. $C(D_{x_1}, \ldots, D_{x_{i-1}}, (b^-, b], D_{x_{i+1}}, \ldots, D_{x_k})$.

Le filtrage par Box–Consistance de $P$ est défini de manière similaire au filtrage par 2B–Consistance de $P$, et est noté $\Phi_{Box}(P)$.

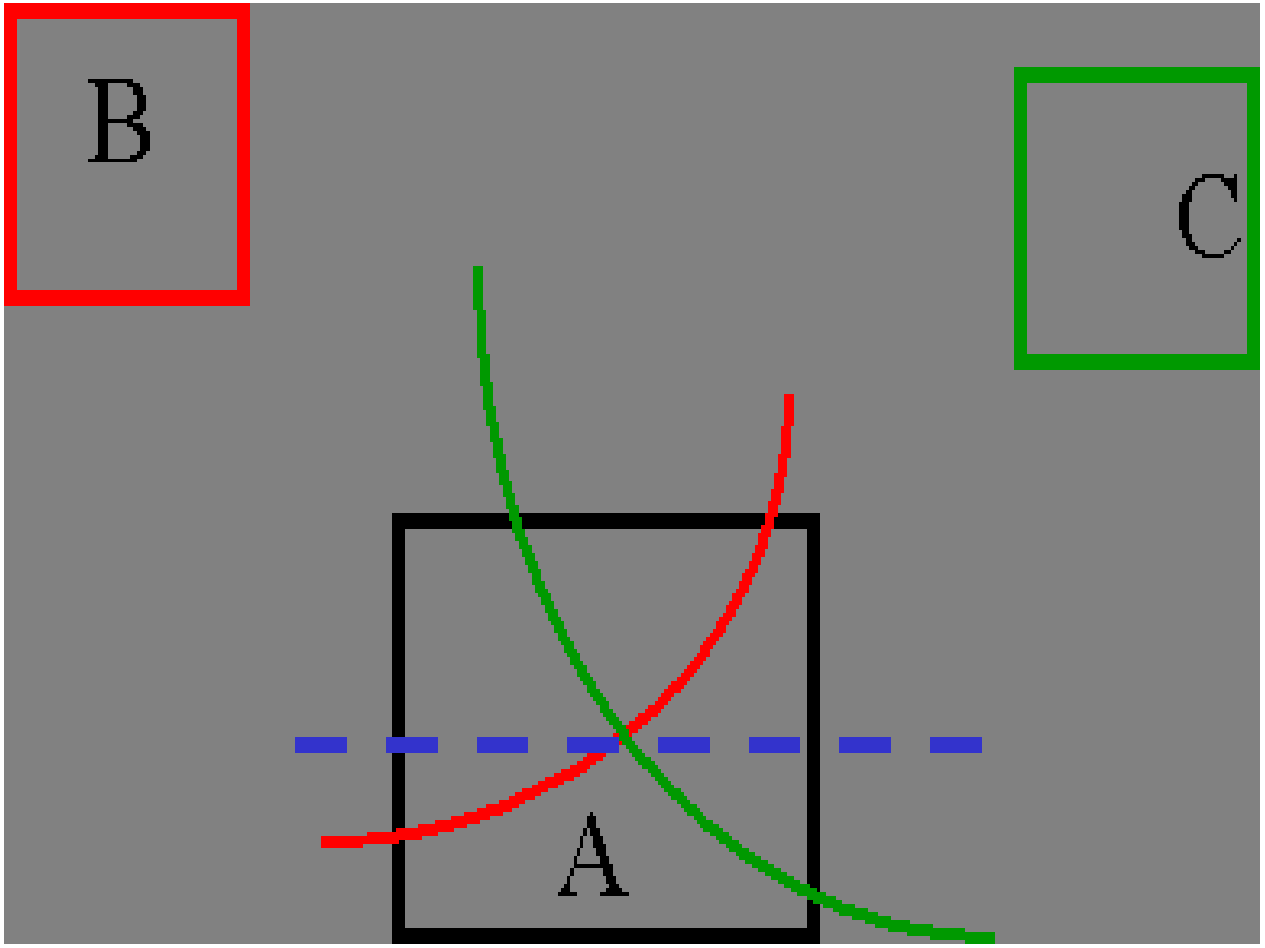
## Intervalle quantifiés (1)



## Intervalle quantifiés (2)



## Contrainte globale



# Quelques limites de la programmation par contraintes

## Exemple des limites liées au choix des domaines

Nombre d'oiseaux et de chats qui totalisent 12 têtes et 34 pattes.

Contraintes : {p=4c+2o,t=c+o}

Résolution dans $Q$ avec un algorithme de Gauss:

p = 12, t =34 $\rightarrow$ c = 5, o =7

p= 6, t= 4 $\rightarrow$ c = -1, o =5

p= 7, t= 3 $\rightarrow$ c = 1/2, o =5/2

Il est possible d'ajouter des contraintes linéaires pour obliger $c$ et $o$ à prendre des valeurs positives, mais il n'est pas possible d'exprimer avec des contraintes linéaires le fait que ces variables doivent prendre des valeurs entières.

## Conséquences des erreurs de calculs

L'algorithme du simplexe est numériquement instable sur les flottants et la correction d'un système $\mathcal{CLP}(\mathcal{R}_{Lin})$qui approxime les réels par des flottants ne peut être assurée.

## Problèmes insolubles

Un problème avec 4 variables :

$\mathcal{C} = \{x^n + y^n = z^n\}$ avec $x, y, z, n \in N$ et $n > 2$

Le théorème d'incomplétude de Goedel limite les possibilités dans $N$...