Introduction à la Programmation par Contraintes

Marie Pelleau

Université Côte d'Azur, i3s, France

EJCP
Jeudi 6 juillet 2023

Crédits

Certains slides sont empruntés à

Charlotte Truchet
MCF U. Nantes

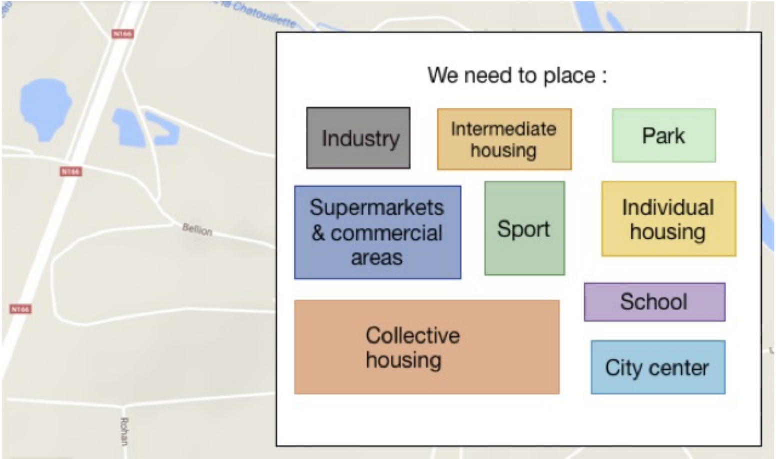
Ghiles Ziat
MCF Epita

Contenu

1. Par l'exemple
2. CSP et modélisation
3. Résolution
   - Cohérence
   - Branchement et heuristiques
4. Résolution abstraite
   - Interprétation Abstraite
5. AbSolute
   - Domaines abstraits en PPC
6. Conclusion

Urban planning

Urban planning

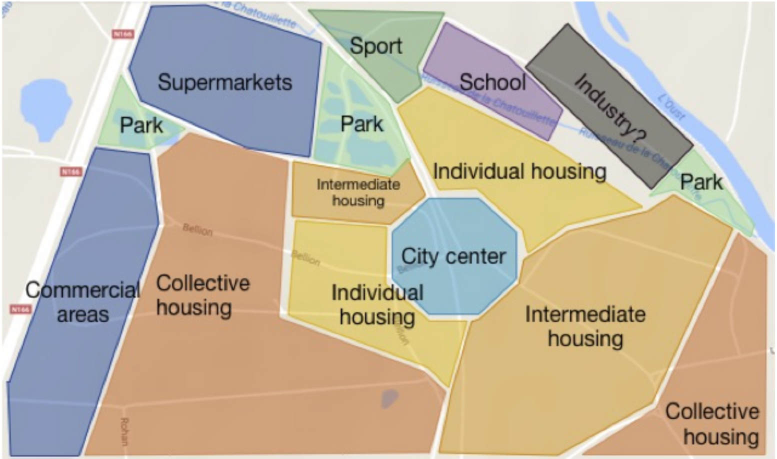


Urban planning

Urban planning

Urban planning

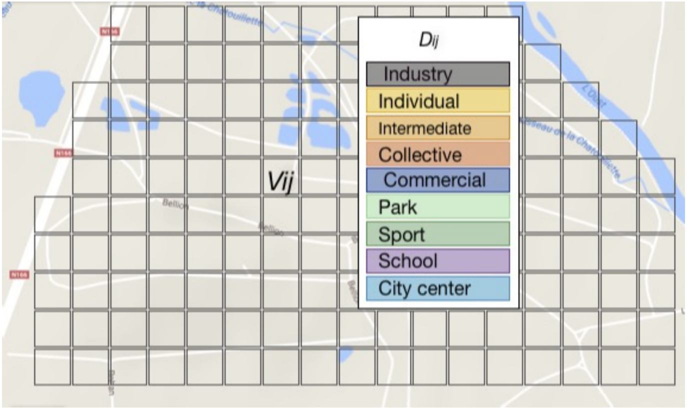
Urban planning



Urban planning

Urban planning

Une variable est une inconnue du problème. Elle a un domaine donné, l'ensemble des valeurs qu'elle peut prendre



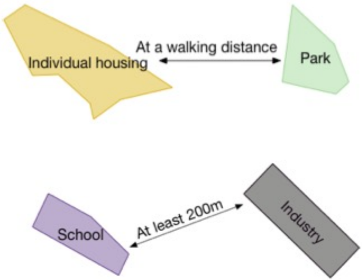
Urban planning

Une variable est une inconnue du problème. Elle a un domaine donné, l'ensemble des valeurs qu'elle peut prendre

Urban planning

Une variable est une inconnue du problème. Elle a un domaine donné, l’ensemble des valeurs qu’elle peut prendre

Une contrainte est une relation logique entre les variables



Urban planning

Une variable est une inconnue du problème. Elle a un domaine donné, l'ensemble des valeurs qu'elle peut prendre

Une contrainte est une relation logique entre les variables

Un solveur de contraintes cherche des valeurs pour les variables telles que les contraintes soient toutes satisfaites

Sustain project

Simulation sur Marne-le-Vallée, une ville française de 8728 hectares, avec ~230 000 habitants, ~10 000 cellules

Thèse de Bruno Belin : `https://www.theses.fr/2014NANT2084`

Programmation par Contraintes

En pratique

- Problème combinatoire
  - Des choix doivent être faits
  - Des choix peuvent avoir des conséquences longtemps après qu'ils soient faits
  - Les choix peuvent être changés
- Problème déclaratif
  - Vérifier est facile, en utilisant les règles ou les connaissances client
  - Construire éfficacement est difficile

Est-ce un problème pour les contraintes ?

Est-ce un problème pour les contraintes ?

Placement de VMs sur des machines réelles ?

Est-ce un problème pour les contraintes ?

Placement de VMs sur des machines réelles ?

Oui !
Projet Entropy, solveur Choco, thèse de Fabien Hermenier :
`https://www.theses.fr/2009NANT2081`

Est-ce un problème pour les contraintes ?

Faire l’emploi du temps d’un hôpital, avec différentes catégories de personnel médical et une règlementation spécifique ?

Est-ce un problème pour les contraintes ?

Faire l'emploi du temps d'un hôpital, avec différentes catégories de personnel médical et une règlementation spécifique ?

Oui !

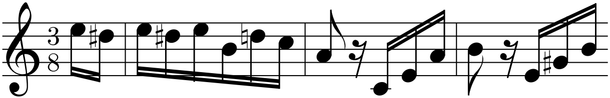
C'est le nurse roastering problem

Des tonnes d'applications

Par exemple, ce survey : : `https://citeseerx.ist.psu.edu/doc_view/pid/ea755c1bea1192b9ca5ea888d75d5d63c291f506`

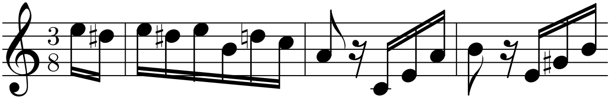
Est-ce un problème pour les contraintes ?

Écrire une partition musicale harmonieuse (au sens de la musique classique) ?



Est-ce un problème pour les contraintes ?

Écrire une partition musicale harmonieuse (au sens de la musique classique) ?



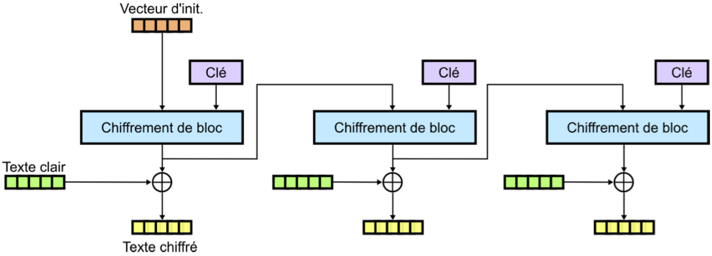
Oui !

C'est même, historiquement, un des premiers problèmes de contraintes

Références : [Ebcioglu, 1984, Pachet and Roy, 1999]

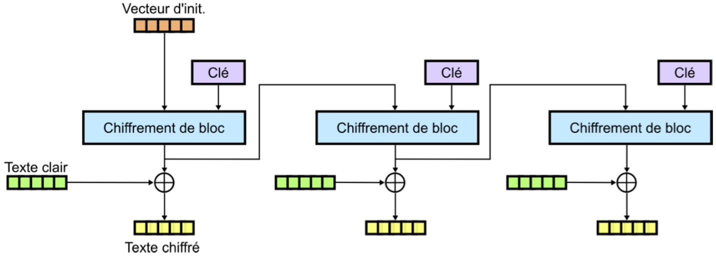
Est-ce un problème pour les contraintes ?

Améliorer ses chances de trouver la clef d'un système de crypto symétrique ?



Est-ce un problème pour les contraintes ?

Améliorer ses chances de trouver la clef d'un système de crypto symétrique ?



Oui !
En tous cas c'est en cours... Projet Décrypt, référence : [Gérault et al., 2020]

Est-ce un problème pour les contraintes ?

Générer des phrases pour des tests de lecture ?

Nous pouvons faire une centaine de pas dans cette direction

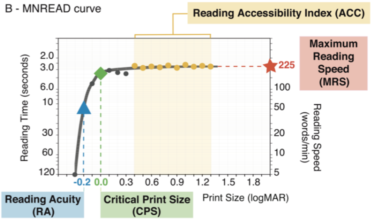
Une phrase standardisée

→

(Test MNREAD)

(Mansfield et al., 1993)

→



Est-ce un problème pour les contraintes ?

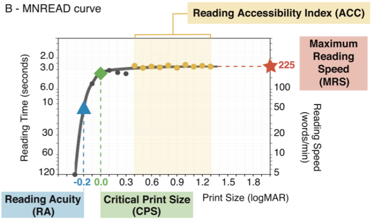
Générer des phrases pour des tests de lecture ?

Nous pouvons faire
une centaine de pas
dans cette direction

Une phrase standardisée

(Test MNREAD)

(Mansfield et al., 1993)



Oui !

Références : [Bonlarron et al., 2023]

Les points communs ?

À chaque fois, on a

- des problèmes combinatoires (choix)
- avec des contraintes diverses
- pour lesquels il ne semble pas malin de partir de zéro ni de faire un algorithme *ad hoc* (structures similaires)

PPC

La programmation par contraintes (PPC) est à la fois

- une technique d'Intelligence artificielle pour la programmation déclarative
- une série d'algorithmes efficaces pour résoudre les problèmes combinatoires

PPC et ses amis

Dans la même famille

- Programmation Linéaire : les contraintes sont toutes linéaires (ou linéarisées)

  Branch and Bound : même schéma de résolution, différentes contraintes
- Analyse Numérique : fonctions réelles

  Newton : algorithme numérique
- SAT/SMT : les contraintes sont des clauses logiques

  DPLL et l'apprentissage de clauses : même schéma de résolution, différents calculs sur les contraintes
- Plannification : optimisation du temps

  A : une heuristique commune*

Il ~~n'y a pas~~ ne devrait pas y avoir de compéttitions
En pratique, toutes ces techniques sont souvent combinées

Contenu

1. Par l'exemple
2. CSP et modélisation
3. Résolution
   - Cohérence
   - Branchement et heuristiques
4. Résolution abstraite
   - Interprétation Abstraite
5. AbSolute
   - Domaines abstraits en PPC
6. Conclusion

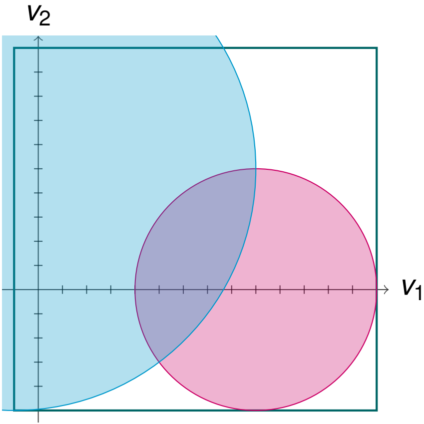
Constraint Satisfaction Problem (CSP)

Définition (CSP)

- un ensemble de variables $\{v_1, \ldots, v_n\}$ (n fixé)
- un ensemble de domaines $\{D_1, \ldots, D_n\}$
- un ensemble de contraintes $\{c_1, \ldots, c_p\}$

Exemple

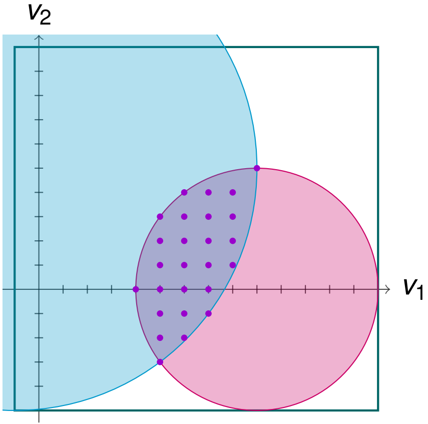
- $V = \{v_1, v_2\}$
- $D_1 = [-1, 14], D_2 = [-5, 10]$
- $C_1 : (v_1 - 9)^2 + v_2^2 \leq 25$
- $C_2 : (v_1 + 1)^2 + (v_2 - 5)^2 \leq 100$



Constraint Satisfaction Problem (CSP)

Définition (Solution)

Une solution est une instanciation de valeurs des domaines aux variables satisfaisant toutes les contraintes



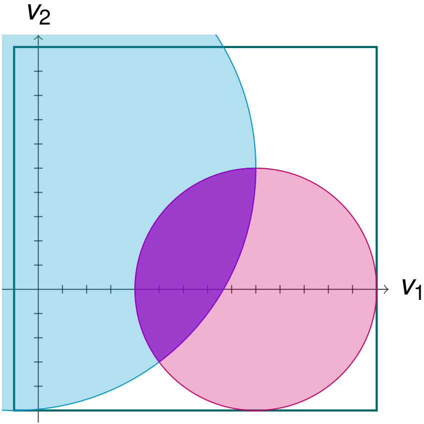
Constraint Satisfaction Problem (CSP)

Définition (Solution)

Une solution est une instanciation de valeurs des domaines aux variables satisfaisant toutes les contraintes

Dans le cas continu, si les solutions ne sont pas représentables en machine, une solution peut être donnée par une approximation :

- extérieure de l'ensemble des solutions (solveur complet)
- intérieure (solveur correct)



Constraint Satisfaction Problem (CSP)

Remarque

- Une contrainte restreint les valeurs que les variables peuvent prendre (généralement la seule chose qui relie les variables)
- Un CSP définit un espace de possibilités $D_1 \times \cdots \times D_n$ de taille d^n (si $\forall i, |D_i| = d$)
- On utilise la programmation par contraintes sur des problèmes NP-durs

Contraintes

Les langages de contraintes incluent en général

- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<, \leq, >, \geq, =, \neq$)

$$v_1 + 7 = v_2$$
$$v_1 \times v_3 < v_5$$
$$\sum_i v_i > M$$

Contraintes

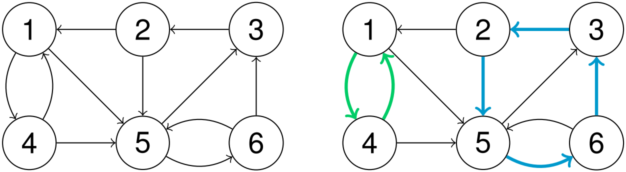
Les langages de contraintes incluent en général

- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<$, $\leq$, $>$, $\geq$, $=$, $\neq$)
- Contraintes globales

Contraintes

Les langages de contraintes incluent en général

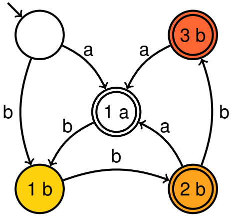
- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<, \leq, >, \geq, =, \neq$)
- Contraintes globales
  - sur des graphes : `tree`, `forest`, `circuit`...



Contraintes

Les langages de contraintes incluent en général

- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<, \leq, >, \geq, =, \neq$)
- Contraintes globales
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...



Contraintes

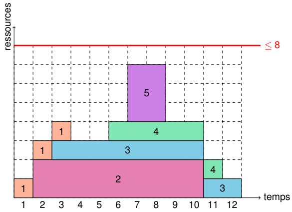
Les langages de contraintes incluent en général

- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<, \leq, >, \geq, =, \neq$)
- Contraintes globales
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`, ...

Contraintes

Les langages de contraintes incluent en général

- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<, \leq, >, \geq, =, \neq$)
- Contraintes globales
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`, ...
  - spécifiques : `cumulative`, `geost`, ...



Contraintes

Les langages de contraintes incluent en général

- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<, \leq, >, \geq, =, \neq$)
- Contraintes globales
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`, ...
  - spécifiques : `cumulative`, `geost`, ...
  - de cardinalité : `alldifferent`, `nvalue`, `atleast`, `gcc`, ...

Contraintes

Les langages de contraintes incluent en général

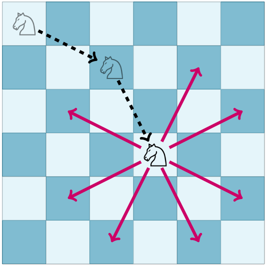
- Expressions arithmétiques, fonctions "raisonnables"
- Comparateurs usuels ($<, \leq, >, \geq, =, \neq$)
- Contraintes globales
  - sur des graphes : `tree`, `forest`, `circuit`...
  - sur des mots : `regular`, `cost-regular`, ...
  - pratiques : `element`, `table`, ...
  - spécifiques : `cumulative`, `geost`, ...
  - de cardinalité : `alldifferent`, `nvalue`, `atleast`, `gcc`, ...

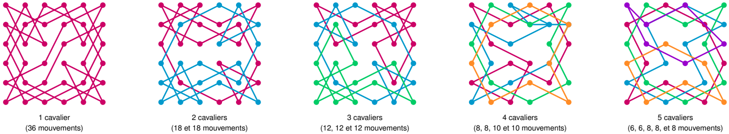
Presque toutes les contraintes globales sont référencées dans le *Global Constraint Catalog*, avec un format commun, et les références bibliographiques : `http://sofdem.github.io/gccat/`

Un exemple

Le Cavalier d'Euler

Un cavalier (pièce d'échecs) doit parcourir un échiquier
Il doit passer une et une seule fois par chaque case et revenir à son point de départ

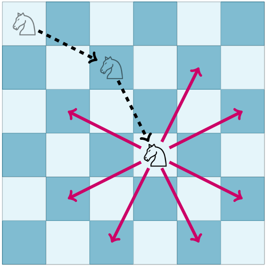




Un exemple

Le Cavalier d'Euler

Un cavalier (pièce d'échecs) doit parcourir un échiquier
Il doit passer une et une seule fois par chaque case et revenir à son point de départ

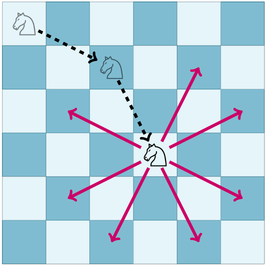


Comment modéliser ce jeu ?

Un exemple

Soient x_i, y_i les coordonnées du cavalier à l'étape i et f une fonction énumérant les cases en fonction des coordonnées
On a

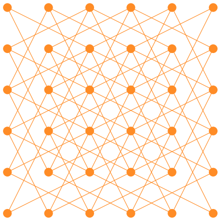
- $\forall i, j < n, x_i \neq x_{i+1} \wedge y_i \neq y_{i+1}$ et $|x_{i+1} - x_i| + |y_{i+1} - y_i| = 3$ (mouvement du cavalier)
- $\forall i \neq j, f(x_i, y_i) \neq f(x_{i+1}, y_{i+1})$ (pas 2 fois la même case)
- $1 \leq x_i, y_i \leq n$



Un exemple

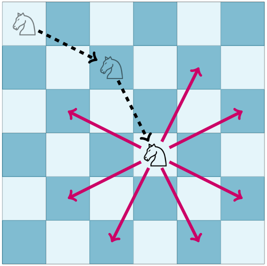
Un autre modèle

Considérons le graphe des mouvements possibles sur l'échiquier



Une seule contrainte nécessaire

`circuit(Sommets, 1)`



Contenu

1 Par l'exemple

2 CSP et modélisation

3 Résolution
- Cohérence
- Branchement et heuristiques

4 Résolution abstraite
- Interprétation Abstraite

5 AbSolute
- Domaines abstraits en PPC

6 Conclusion

Déduction

Pour résoudre un CSP, on cherche à réduire l'espace des possibles

Raisonnement sur les contraintes

Dès qu'on a placé la zone industrielle, on peut éliminer la valeur "école" tout autour

Cohérences sur les domaines finis

Une contrainte $c(v_1, \ldots, v_n)$ est generalized arc-consistent (GAC) pour des domaines $D_1, \ldots, D_n$ ssi pour toute variable v_i, pour toute valeur $v^i \in D_i$, il existe des valeurs $v^1 \in D_1$, …, $v^{i-1} \in D_{i-1}$, $v^{i+1} \in D_{i+1}$, …, $v^n \in D_n$ telles que $c(v^1, \ldots, v^n)$

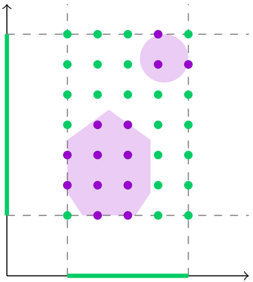
Cohérences sur les domaines finis

Une contrainte $c(v_1, \ldots, v_n)$ est generalized arc-consistent (GAC) pour des domaines $D_1, \ldots, D_n$ ssi pour toute variable v_i, pour toute valeur $v^i \in D_i$, il existe des valeurs $v^1 \in D_1$, ..., $v^{i-1} \in D_{i-1}$, $v^{i+1} \in D_{i+1}$, ..., $v^n \in D_n$ telles que $c(v^1, \ldots, v^n)$

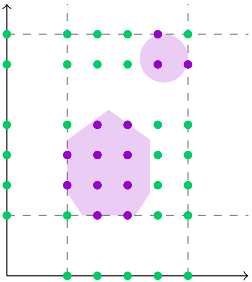
Une contrainte $c(v_1, \ldots, v_n)$ est bound-consistent (BC) pour des domaines $D_1, \ldots, D_n$ ssi les bornes de tous les domaines sont cohérentes (au sens ci-dessus)

Cohérences sur les domaines finis

En mauve les solutions, en vert les domaines cohérents



Bound-consistency



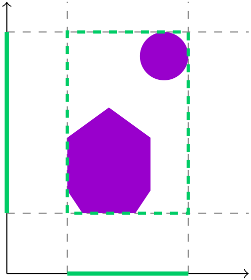
Generalized arc-consistency

Cohérences sur les domaines continus

Une contrainte c sur des variables $v_1, \ldots, v_n$ est de domaines $D_1, \ldots, D_n$ est Hull cohérent (HC) ssi $D_1 \times \cdots \times D_n$ est la plus petite boîte réelle, à bornes flottantes, contenant les solutions de c dans $D_1 \times \cdots \times D_n$

Cohérences sur les domaines continus

En mauve les solutions, en vert les domaines cohérents



Hull-consistency

Propagation

La propagation d'une contrainte c sur les domaines $D_1, \ldots, D_n$ consiste à enlever des domaines toutes les valeurs incohérentes pour cette contrainte

Exemple

Contrainte $x = y + 3 \times z$

- Si $x = 10$, $y = 4$, alors on en déduit que

Propagation

La propagation d'une contrainte c sur les domaines $D_1, \ldots, D_n$ consiste à enlever des domaines toutes les valeurs incohérentes pour cette contrainte

Exemple

Contrainte $x = y + 3 \times z$

- Si $x = 10$, $y = 4$, alors on en déduit que $z = 2$

Propagation

La propagation d'une contrainte c sur les domaines $D_1, \ldots, D_n$ consiste à enlever des domaines toutes les valeurs incohérentes pour cette contrainte

Exemple

Contrainte $x = y + 3 \times z$

- Si $x = 10$, $y = 4$, alors on en déduit que $z = 2$
- Si $D_x = [0, 10]$, $D_y = [0, 10]$ et $D_z = [1, 5]$, alors on en déduit que

Propagation

La propagation d'une contrainte c sur les domaines $D_1, \ldots, D_n$ consiste à enlever des domaines toutes les valeurs incohérentes pour cette contrainte

Exemple

Contrainte $x = y + 3 \times z$

- Si $x = 10$, $y = 4$, alors on en déduit que $z = 2$
- Si $D_x = [0, 10]$, $D_y = [0, 10]$ et $D_z = [1, 5]$, alors on en déduit que $D_y = [0, 10] \cap [0, 10] - 3 \times [1, 5] = [0, 10] \cap [-5, 7]$ donc $D_y = [0, 7]$

Propagation

La propagation d'une contrainte c sur les domaines $D_1, \ldots, D_n$ consiste à enlever des domaines toutes les valeurs incohérentes pour cette contrainte

Exemple

Contrainte $x = y + 3 \times z$

- Si $x = 10$, $y = 4$, alors on en déduit que $z = 2$
- Si $D_x = [0, 10]$, $D_y = [0, 10]$ et $D_z = [1, 5]$, alors on en déduit que $D_y = [0, 10] \cap [0, 10] - 3 \times [1, 5] = [0, 10] \cap [-5, 7]$ donc $D_y = [0, 7]$

Contrainte `alldifferent`(x_1, x_2, x_3) ; si on sait que $D_1 = D_2 = \{1, 2\}$, et $D_3 = \{0, 1, 2, 3, 5, 8\}$ on en déduit que

Propagation

La propagation d'une contrainte c sur les domaines $D_1, \ldots, D_n$ consiste à enlever des domaines toutes les valeurs incohérentes pour cette contrainte

Exemple

Contrainte $x = y + 3 \times z$

- Si $x = 10$, $y = 4$, alors on en déduit que $z = 2$
- Si $D_x = [0, 10]$, $D_y = [0, 10]$ et $D_z = [1, 5]$, alors on en déduit que $D_y = [0, 10] \cap [0, 10] - 3 \times [1, 5] = [0, 10] \cap [-5, 7]$ donc $D_y = [0, 7]$

Contrainte `alldifferent`(x_1, x_2, x_3) ; si on sait que $D_1 = D_2 = \{1, 2\}$, et $D_3 = \{0, 1, 2, 3, 5, 8\}$ on en déduit que les valeurs 1 et 2 peuvent être éliminées du domaine de x_3

Arithmétique des intervalles

Opérations arithmétiques

- $[a, b] + [c, d] = [a + c, b + d]$
- $[a, b] - [c, d] = [a - d, b - c]$
- $[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $[a, b] \div [c, d] = \left[\min\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right), \max\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right)\right]$ si $0 \notin [c, d]$

Évaluer une contrainte

$x \in [-2, 5]$
$y \in [-3, 7]$

$$2x - y \quad = \quad 0$$

Arithmétique des intervalles

Opérations arithmétiques

- $[a, b] + [c, d] = [a + c, b + d]$
- $[a, b] - [c, d] = [a - d, b - c]$
- $[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $[a, b] \div [c, d] = \left[\min\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right), \max\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right)\right]$ si $0 \notin [c, d]$

Évaluer une contrainte

$x \in [-2, 5]$
$y \in [-3, 7]$

$$2x - y \quad = \quad 0$$

Arithmétique des intervalles

Opérations arithmétiques

- $[a, b] + [c, d] = [a + c, b + d]$
- $[a, b] - [c, d] = [a - d, b - c]$
- $[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $[a, b] \div [c, d] = \left[\min\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right), \max\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right)\right]$ si $0 \notin [c, d]$

Évaluer une contrainte

$x \in [-2, 5]$
$y \in [-3, 7]$

$$\begin{aligned} 2x - y &= 0 \\ 2 \times [-2, 5] - [-3, 7] &= 0 \end{aligned}$$

Arithmétique des intervalles

Opérations arithmétiques

- $[a, b] + [c, d] = [a + c, b + d]$
- $[a, b] - [c, d] = [a - d, b - c]$
- $[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $[a, b] \div [c, d] = \left[\min\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right), \max\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right)\right]$ si $0 \notin [c, d]$

Évaluer une contrainte

$x \in [-2, 5]$
$y \in [-3, 7]$

$$\begin{aligned} 2x - y &= 0 \\ 2 \times [-2, 5] - [-3, 7] &= 0 \\ [-4, 10] - [-3, 7] &= 0 \\ [-11, 13] &= 0 \end{aligned}$$

Arithmétique des intervalles

Opérations arithmétiques

- $[a, b] + [c, d] = [a + c, b + d]$
- $[a, b] - [c, d] = [a - d, b - c]$
- $[a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$
- $[a, b] \div [c, d] = \left[\min\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right), \max\left(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}\right)\right]$ si $0 \notin [c, d]$

Évaluer une contrainte

$x \in [-2, 5]$
$y \in [-3, 7]$

$$\begin{aligned} 2x - y &= 0 \\ 2 \times [-2, 5] - [-3, 7] &= 0 \\ [-4, 10] - [-3, 7] &= 0 \\ [-11, 13] &= 0 \end{aligned}$$

$0 \in$ à l'intervalle résultat
$\Rightarrow$ Il existe peut-être une solution

$0 \notin$ à l'intervalle résultat
$\Rightarrow$ Pas de solution

Arithmétique des intervalles

Opérateurs ensemblistes

- $[a, b] \cap [c, d] = [\max(a, c), \min(b, d)]$
- $[a, b] \cup [c, d] = [\min(a, c), \max(b, d)]$

Opérateurs inverses

On considère 3 intervalles u, v et r

- $u + v = r$
  $\Rightarrow u = u \cap (r - v)$
  $\Rightarrow v = v \cap (r - u)$
- $u - v = r$
  $\Rightarrow u = u \cap (r + v)$
  $\Rightarrow v = v \cap (u - r)$

Exemples d'algorithme de propagation

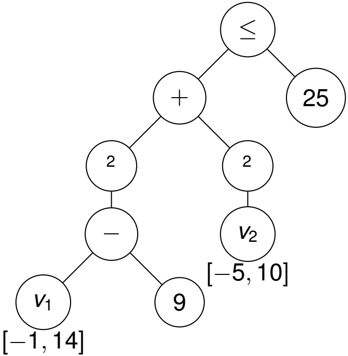
- HC4 [Benhamou et al., 1999] : un algorithme pour la hull consistency
- GAC pour la contrainte `alldifferent` [Régin, 1994]

Il existe de nombreux algoriithmes de propagation
`http://ktiml.mff.cuni.cz/~bartak/constraints/`,
`http://sofdem.github.io/gccat/`

HC4-Revise

[Benhamou, 1996]

$$(v_1 - 9)^2 + v_2^2 \leq 25, D_1 = [-1, 14], D_2 = [-5, 10]$$

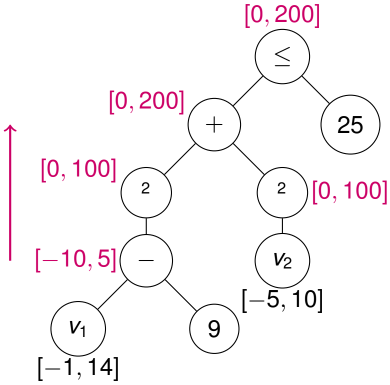


- Arbre syntaxique de la contrainte

HC4-Revise

[Benhamou, 1996]

$$(v_1 - 9)^2 + v_2^2 \leq 25, D_1 = [-1, 14], D_2 = [-5, 10]$$

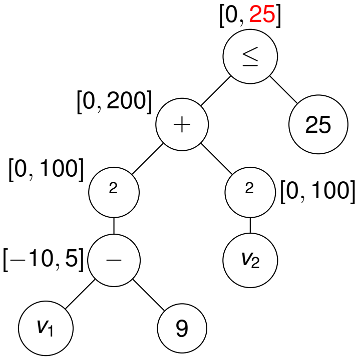


- Arbre syntaxique de la contrainte
- Arithmétique des intervalles

HC4-Revise

[Benhamou, 1996]

$$(v_1 - 9)^2 + v_2^2 \leq 25, D_1 = [-1, 14], D_2 = [-5, 10]$$

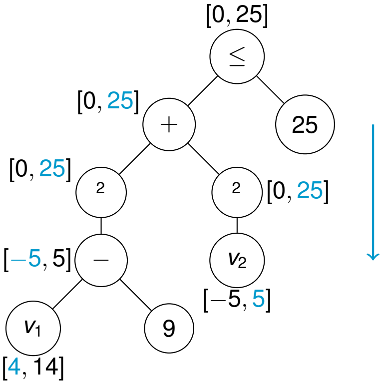


- Arbre syntaxique de la contrainte
- Arithmétique des intervalles

HC4-Revise

[Benhamou, 1996]

$$(v_1 - 9)^2 + v_2^2 \leq 25, D_1 = [-1, 14], D_2 = [-5, 10]$$

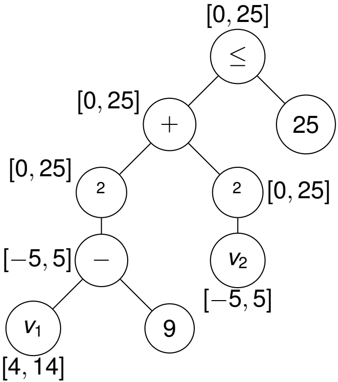


- Arbre syntaxique de la contrainte
- Arithmétique des intervalles

HC4-Revise

[Benhamou, 1996]

$$(v_1 - 9)^2 + v_2^2 \leq 25, D_1 = [-1, 14], D_2 = [-5, 10]$$

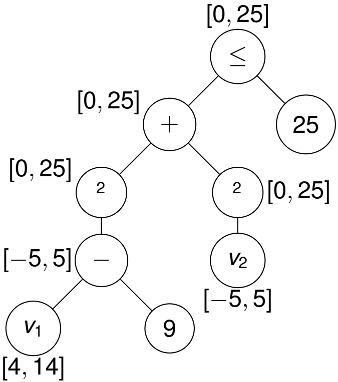


- Arbre syntaxique de la contrainte
- Arithmétique des intervalles
- Cet algorithme réalise HC sous certaines conditions

HC4-Revise

[Benhamou, 1996]

$$(v_1 - 9)^2 + v_2^2 \leq 25, D_1 = [-1, 14], D_2 = [-5, 10]$$



- Arbre syntaxique de la contrainte
- Arithmétique des intervalles
- Cet algorithme réalise HC sous certaines conditions

Remarque

Cet algorithme a aussi été défini indépendamment en IntAbs sous le nom de Bottom-Up Top-Down [Miné, 2004]

Cas des contraintes globales

Les algorithmes de propagation des contraintes globales sont donnés au cas par cas

Grande variété des méthodes utilisées

- souvent issues des graphes ou des flots, par exemple pour les contraintes de cardinalité
- méthodes *ad hoc* : `sweep` adapté pour la contrainte geost
- inspirées de domaines proches : relaxations et programmation linéaire, newton et consistances continues
- ...

Contrainte `alldifferent`

Présentée la première fois dans [Lauriere, 1978]
Retourne vrai si toutes les variables sont différentes deux à deux

Quelle différence entre $\texttt{alldifferent}(v_1, \ldots, v_n)$ et $\wedge_{i \neq j} v_i \neq v_j$?

Contrainte `alldifferent`

Présentée la première fois dans [Lauriere, 1978]
Retourne vrai si toutes les variables sont différentes deux à deux

Quelle différence entre $\texttt{alldifferent}(v_1, \ldots, v_n)$ et $\wedge_{i \neq j} v_i \neq v_j$?

- Sémantiquement : aucune

Contrainte `alldifferent`

Présentée la première fois dans [Lauriere, 1978]
Retourne vrai si toutes les variables sont différentes deux à deux

Quelle différence entre $\texttt{alldifferent}(v_1, \ldots, v_n)$ et $\wedge_{i \neq j} v_i \neq v_j$?

- Sémantiquement : aucune
- Opérationnellement : tout

Graphe des valeurs

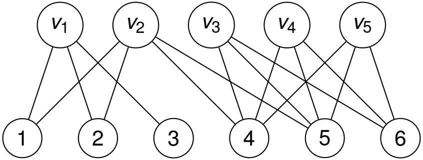
Définition (Graphe des valeurs)

À partir des variables et des domaines d'un CSP on peut créer un graphe biparti, appelé graphe des valeurs

- Les sommets correspondent aux variables et aux valeurs
- Une arête relie une variable v_i et une valeur x si $x \in D_i$

Exemple

- $\{v_1, v_2, v_3, v_4, v_5\}$
- $D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 4, 5\}$, $D_3 = D_4 = D_5 = \{4, 5, 6\}$

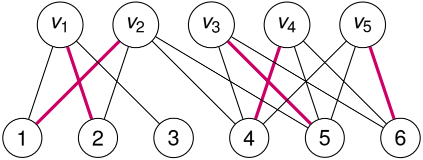


Contrainte `alldifferent`

Une contrainte `alldifferent`$(v_1, \ldots, v_n)$ est satisfaite ssi il existe un couplage maximal

Exemple

- $\{v_1, v_2, v_3, v_4, v_5\}$
- $D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 4, 5\}$, $D_3 = D_4 = D_5 = \{4, 5, 6\}$

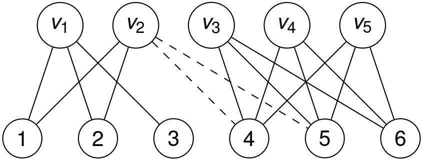


Contrainte `alldifferent`

Pour une variable v_i, une valeur $x \in D_i$ est incohérente ssi l'arête (v_i, x) n'appartient à aucun couplage maximal

Exemple

- $\{v_1, v_2, v_3, v_4, v_5\}$
- $D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 4, 5\}$, $D_3 = D_4 = D_5 = \{4, 5, 6\}$



Les limites

En théorie

Il s’agit d’un bon algorithme

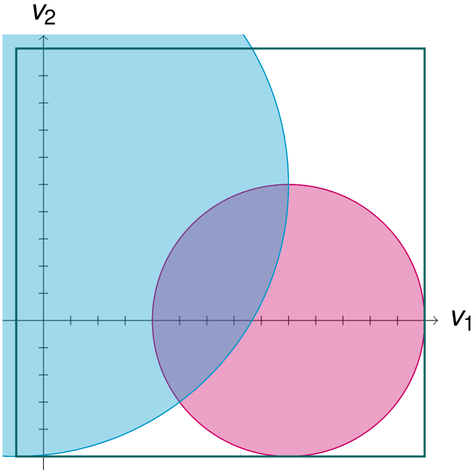
En pratique

- Re-calcul à chaque réveil (peut être coûteux)
- Implémentation de façon incrémentale
  - recalcule le couplage que si le précédent n’est plus valide
  - recalcule pas toujours les cfc

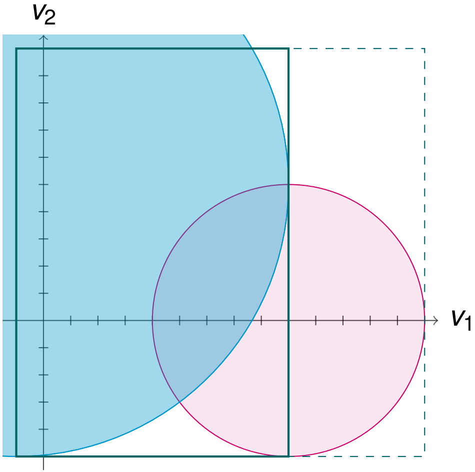
Boucle de propagation

- Pour une conjonction de contraintes, pour chaque contrainte la consistence est exécutée jusqu'au point fixe [Benhamou, 1996], [Apt, 1999]
- À chaque iteration, toutes les contraintes ne sont pas propagées [Mackworth, 1977]
- Le nombre d'iterations dépend :
  - Des contraintes
  - L'ordre dans lequels les contraintes sont propagées
- Concevoir une boucle de propagation efficace est encore un défi [Schulte and Tack, 2001]

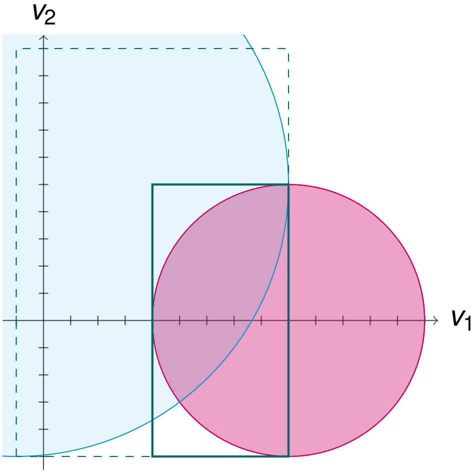
HC4



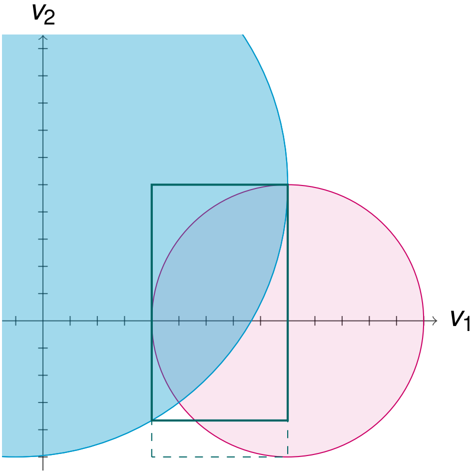
HC4



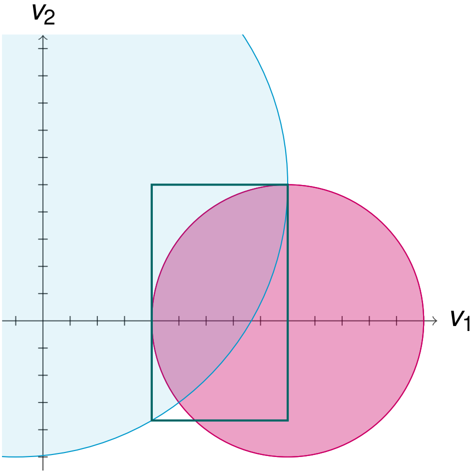
HC4



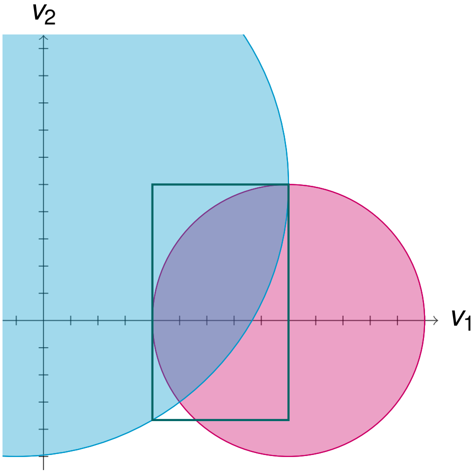
HC4



HC4



HC4



Accélérer la convergence au point fixe ?

Aujourd'hui, beaucoup de solveurs utilisent des boucles de propagations à base d'évènements comme

- une variable a été fixée
- un domaine a été modifié

Le tuning précis de la boucle de propagation, pour accélérer la convergence au point fixe, est encore un sujet de recherche

Méthode de résolution
Comment résoudre ce problème ?

Propagation

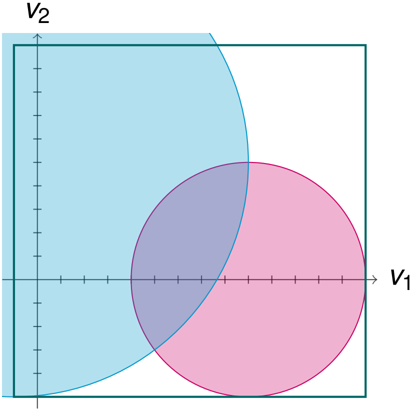
En utilisant les contraintes, supprime des domaines les valeurs ne pouvant être dans une solution $\Longrightarrow$ Pas assez

Exploration

Coupe une boîte en deux plus petites boîtes

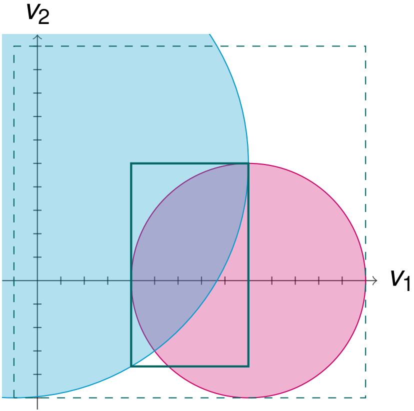
Méthode de résolution continue

```
Parametre: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e) then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and e2
         push e1 and e2 in toExplore
```



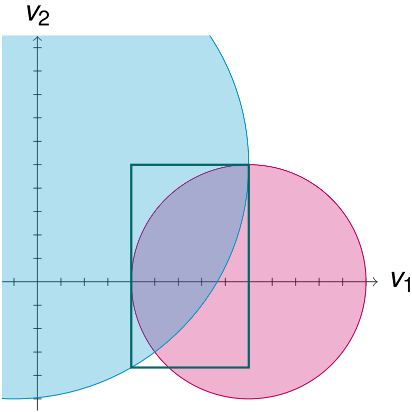
Méthode de résolution continue

```
Parametre: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Hull-Consistency(e)
  if e ≠ ∅ then
    if maxDim(e) ≤ r or isSol(e) then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and e2
      push e1 and e2 in toExplore
```



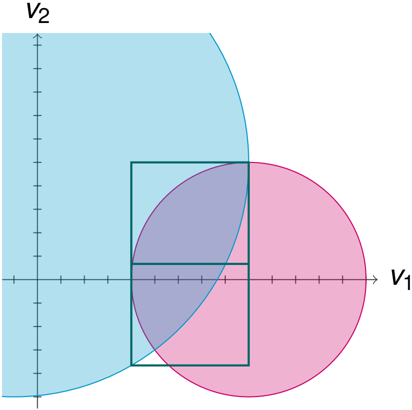
Méthode de résolution continue

```
Parametre: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Hull-Consistency(e)
  if e ≠ ∅ then
    if maxDim(e)≤ r or isSol(e) then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and e2
      push e1 and e2 in toExplore
```



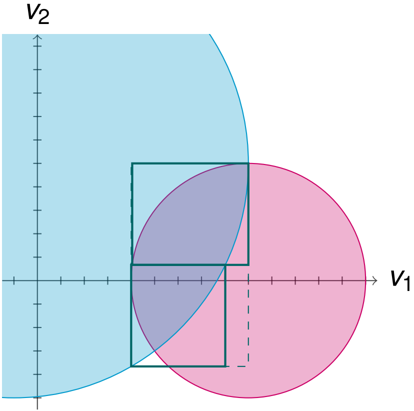
Méthode de résolution continue

```
Parametre: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e) then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and e2
         push e1 and e2 in toExplore
```



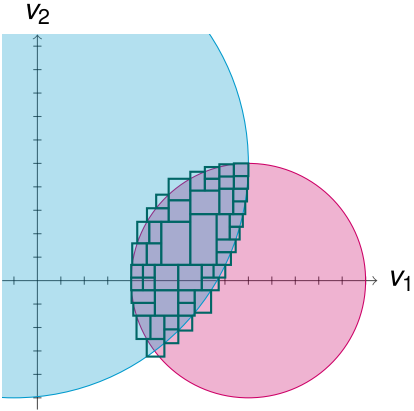
Méthode de résolution continue

```
Parametre: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e) ≤ r or isSol(e) then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and e2
         push e1 and e2 in toExplore
```



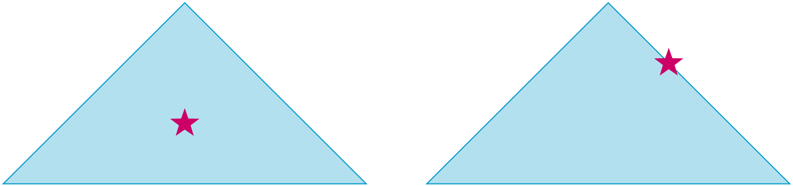
Méthode de résolution continue

```
Parametre: float r

list of boxes sols ← ∅
queue of boxes toExplore ← ∅
box e

e ← D
push e in toExplore

while toExplore ≠ ∅ do
   e ← pop(toExplore)
   e ← Hull-Consistency(e)
   if e ≠ ∅ then
      if maxDim(e)≤ r or isSol(e) then
         sols ← sols ∪ e
      else
         split e in two boxes e1 and e2
         push e1 and e2 in toExplore
```



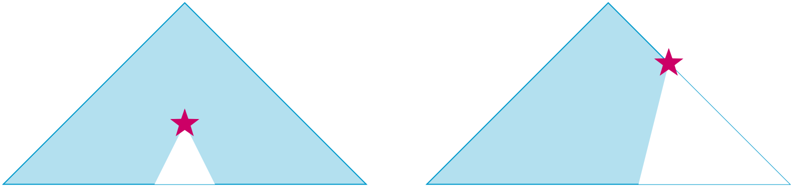
Heuristiques

On imagine l'arbre de recherche, dont les nœuds sont les décisions
À chaque nœud, on filtre, ce qui enlève des valeurs (étoile rose)
Pour une même valeur enlevée, que vaut-il mieux ?

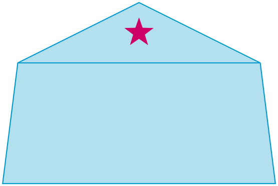


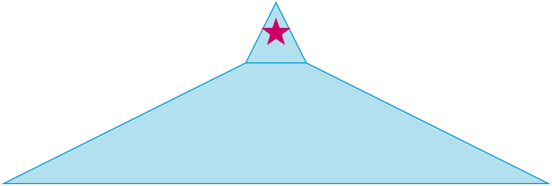
Heuristiques

On imagine l'arbre de recherche, dont les nœuds sont les décisions
À chaque nœud, on filtre, ce qui enlève des valeurs (étoile rose)
Pour une même valeur enlevée, que vaut-il mieux ?

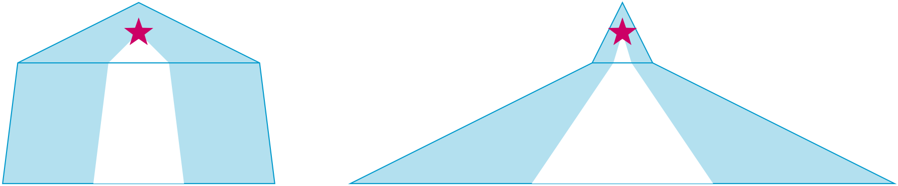


Heuristiques





Heuristiques



Stratégie d'exploration

Choisir la variable

- Ayant le plus petit domaine (dom), First-fail [Haralick and Elliott, 1979]
- Apparaissant dans le plus grand nombre de contraintes (deg)
- $\mathrm{dom} + \mathrm{deg}$ [Brélaz, 1979]
- $\mathrm{dom}/\mathrm{deg}$ [Bessière and Régin, 1996]
- $\mathrm{dom}/w\mathrm{deg}$ [Boussemart et al., 2004]
- $\ldots$

Solveur

- Choco : java library, free
- gecode : C++ library, free
- ORTools : C++, interface in Python, free
- Oscar : Scala, free,
- Prolog family : ECLiPSe, Sicstus
- AbSolute : OCaml, free
- De très nombreux autres

2 compétitions annuelles

- XCSP3 `http://www.cril.univ-artois.fr/XCSP18/`
- MiniZinc Challenge `http://www.minizinc.org/challenge2018/challenge.html`

Contenu

1 Par l'exemple

2 CSP et modélisation

3 Résolution
- Cohérence
- Branchement et heuristiques

4 Résolution abstraite
- Interprétation Abstraite

5 AbSolute
- Domaines abstraits en PPC

6 Conclusion

Interprétation Abstraite

- Interprétation Abstraite (IntAbs) est une théorie des approximations de sémantiques [Cousot and Cousot, 1976]
- Utilisée pour l'analyse statique et la vérification des logiciels
- Example d'application : prouver automatiquement qu'un programme ne contient pas d'erreurs d'exécution

Interprétation Abstraite

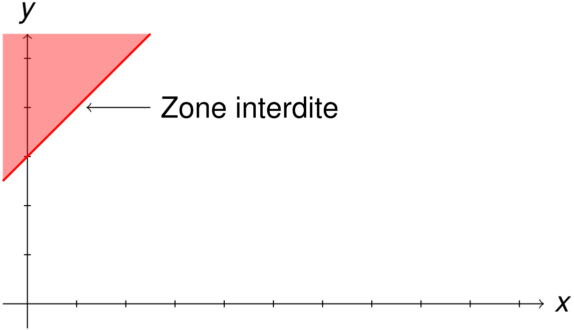
Étudie les valeurs des variables

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```

Interprétation Abstraite

Étudie les valeurs des variables

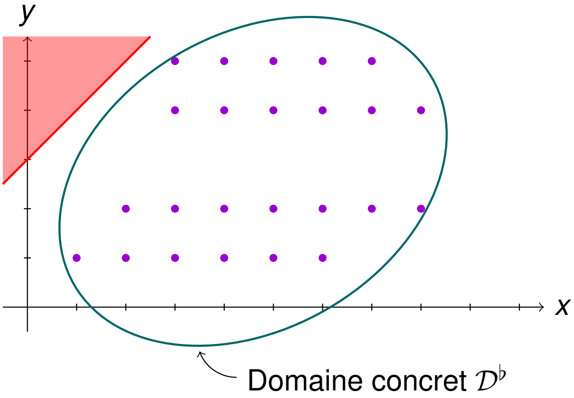
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2∗y
x ← x−1
y ← y+1
```



Interprétation Abstraite

Étudie les valeurs des variables

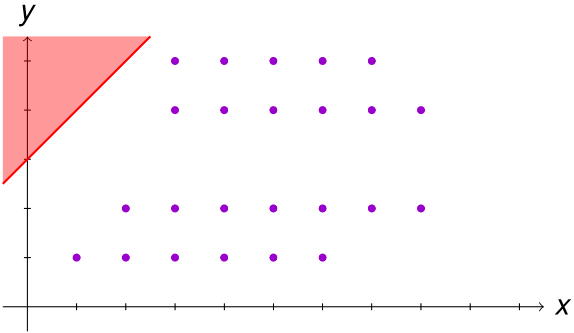
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



Interprétation Abstraite

Étude les valeurs des variables

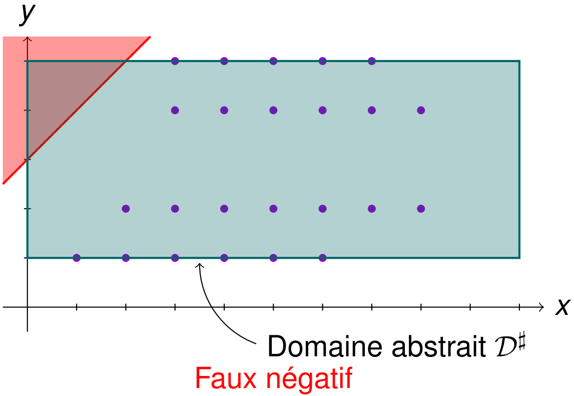
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



Interprétation Abstraite

Étudie les valeurs des variables

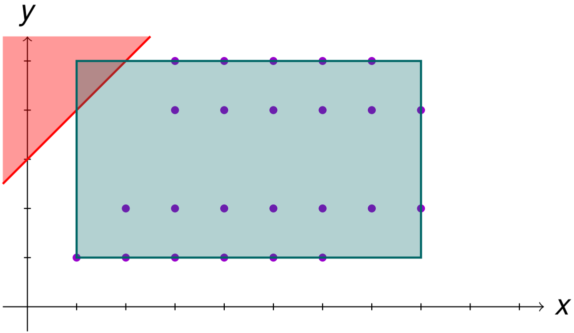
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



Interprétation Abstraite

Étudie les valeurs des variables

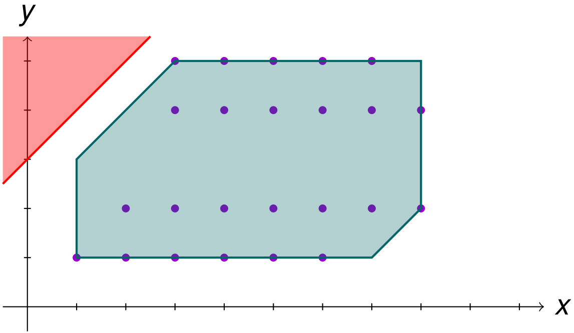
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



Interprétation Abstraite

Étudie les valeurs des variables

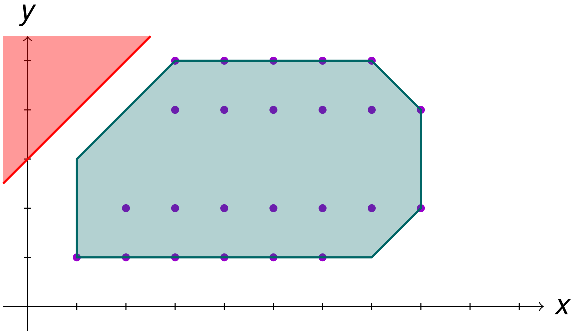
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
   x ← x+y
   y ← 2*y
x ← x-1
y ← y+1
```



Interprétation Abstraite

Étudie les valeurs des variables

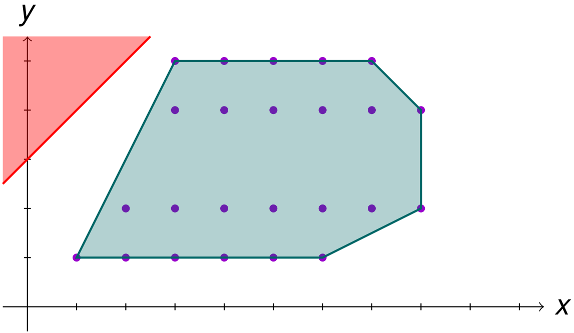
```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



Interprétation Abstraite

Étudie les valeurs des variables

```
int x, y
y ← 1
x ← random(1, 5)
while y<3 and x≤8 do
  x ← x+y
  y ← 2*y
x ← x-1
y ← y+1
```



IntAbs ? PPC ?

En IntAbs

Peu importe de savoir où on va, du moment qu'on sait où on ne va pas

En PPC

Si on sait où on ne va pas, on le coupe

Liens

PPC $\cap$ IntAbs

Approximations d'espaces compliqués ou impossibles à calculer exactement

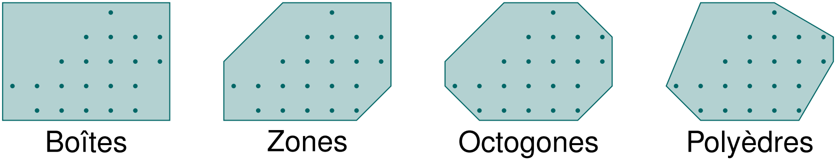
IntAbs $\setminus$ PPC

- Nombreux domaines abstraits
- Cadre théorique commun

PPC $\setminus$ IntAbs

- Nombreuses heuristiques
- Précision

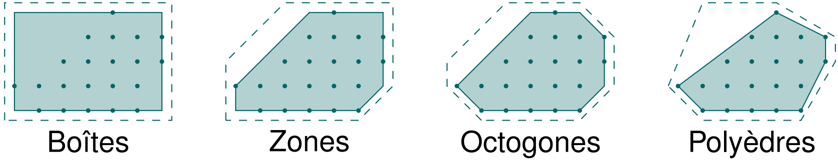
Ce qui existe en IntAbs



Domaines abstraits avec

- des fonctions de transfert $\rho^\sharp$ (affectation, test, ...)
- intersection $\cap^\sharp$ et union $\cup^\sharp$
- widening $\nabla^\sharp$ et narrowing $\triangle^\sharp$

Ce qui existe en IntAbs



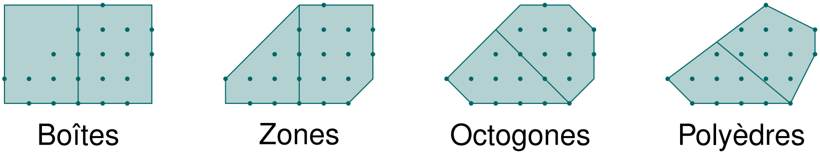
Domaines abstraits avec

- des fonctions de transfert $\rho^\sharp$ (affectation, test, ...)
- intersection $\cap^\sharp$ et union $\cup^\sharp$
- widening $\nabla^\sharp$ et narrowing $\triangle^\sharp$

Ce dont on a besoin

- une consistence

Ce qui existe en IntAbs



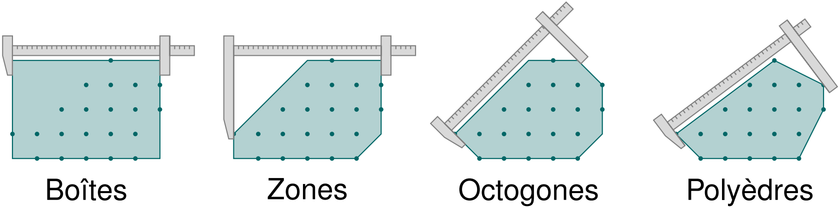
Domaines abstraits avec

- des fonctions de transfert $\rho^\sharp$ (affectation, test, ...)
- intersection $\cap^\sharp$ et union $\cup^\sharp$
- widening $\nabla^\sharp$ et narrowing $\triangle^\sharp$

Ce dont on a besoin

- une consistence
- un opérateur de coupe

Ce qui existe en IntAbs



Domaines abstraits avec

- des fonctions de transfert $\rho^\sharp$ (affectation, test, ...)
- intersection $\cap^\sharp$ et union $\cup^\sharp$
- widening $\nabla^\sharp$ et narrowing $\triangle^\sharp$

Ce dont on a besoin

- une consistence
- un opérateur de coupe
- une fonction de précision

Méthode de résolution abstraite

Propagation

- Consistence : les fonctions de transfert de test
- Boucle de propagation : itérations locales [Granger, 1992]

Exploration

- Opérateur de coupe dans une complétion disjonctive

  Définition (Complétion disjonctive [Cousot and Cousot, 1992])

  Soit $\mathcal{D}^\sharp$un domaine abstrait, une complétion disjonctive $\mathcal{E}^\sharp = \mathcal{P}_{finite}(\mathcal{D}^\sharp)$ est un sous-ensemble de $\mathcal{D}^\sharp$dont les éléments ne sont pas comparables

- Fonction de précision

Méthode de résolution continue

```
Parameter: float r

list of boxes  sols ← ∅
queue of boxes  toExplore ← ∅
box  e ← D

push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← Hull-Consistency(e)
  if e ≠ ∅ then
    if maxDim(e)  ≤ r or isSol(e) then
      sols ← sols ∪ e
    else
      split e in two boxes e1 and e2
      push e1 and e2  in toExplore
```

Méthode de résolution abstraite

```
Parameter: float r

~~list of boxes~~ disjunction sols ← ∅
~~queue of boxes~~ disjunction toExplore ← ∅
~~box~~ abstract domain e ← ~~D~~ ⊤♯

push e in toExplore

while toExplore ≠ ∅ do
  e ← pop(toExplore)
  e ← ~~Hull-Consistency(e)~~ ρ♯(e)
  if e ≠ ∅ then
    if ~~maxDim(e)~~ τ(e) ≤ r or isSol(e) then
      sols ← sols ∪ e
    else
      ~~split e in two boxes e1 and e2~~
      push ~~e1 and e2~~ ⊕(e) in toExplore
```

Suivant les conditions des opérateurs, cet résolution abstraite termine, est correcte ou complète

Implantation

Implantation avec Apron [Jeannet and Miné, 2009], une bibliothèque OCaml de domaines abstraits

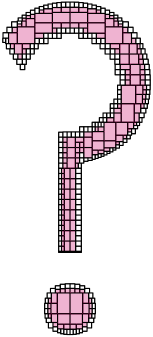
- Écrit en OCaml
- Résolution paramétrable (domaine, précision, profondeur maximum ...)
- Mode visualisation

`https://github.com/mpelleau/AbSolute`

Conclusion

La CP offre un cadre générique pour utiliser de nombreux algorithmes d'optimisation combinatoire avec

- Des modèles faciles à modifier / améliorer incrémentalement
- Des outils efficaces en pratique sur beaucoup de problèmes
- Des liens très naturels avec l'analyse de programmes



Implantation de HC4-Revise ou Bottom-up Top-down

Dans votre langage préféré écrire une classe/module `itv` représentant un intervalle par ses bornes et pouvant représenter l'intervalle "vide"

Écrire l'arithmétique des intervalles

- addition (`itv, itv -> itv`)
- soustraction (`itv, itv -> itv`)
- multiplication (`itv, itv -> itv`)
- division (`itv, itv -> itv`)

Écrire les opérateurs ensemblistes

- intersection (`itv, itv -> itv`)
- union (`itv, itv -> itv`)

Avec ces opérateurs vous pouvez effectuer la montée (Bottom-up)

Implantation de HC4-Revise ou Bottom-up Top-down

Écrire les opérateurs "inverses"
Étant donné un intervalle résultat `r`, un opérateur binaire `op` et les deux opérandes `u` et `v`, l'opérateur "inverse" calcule les intervalles pour `u` et `v` en fonction de `r`

- addition (`itv, itv, itv -> itv, itv`)
  `u, v, r -> u ∩ (r - v), v ∩ (r - u)`
- soustraction (`itv, itv, itv -> itv, itv`)
  `u, v, r -> u ∩ (r + v), v ∩ (u - r)`
- multiplication (`itv, itv, itv -> itv, itv`)
  `u, v, r -> u ∩ (r / v), v ∩ (r / u)`
- division (`itv, itv, itv -> itv, itv`)
  `u, v, r -> u ∩ (r * v), v ∩ (u / r)`

Avec ces opérateurs vous pouvez effectuer la descente (Top-down)

Implantation de HC4-Revise ou Bottom-up Top-down

Écrire une classe/module pour les arbres syntaxiques dans lesquels les nœuds ont un nom et sont annotés d'un intervalle
À l'aide des fonctions sur les intervalles écrire, HC4-Revise ou Bottom-up Top-down

Quand il y a plusieurs contraintes/arbres syntaxiques, il faut faire une boucle afin d'éxecuter HC4-Revise pour chacune des contraintes/arbres jusqu'à atteindre un point fixe

Références I

Apt, K. R. (1999).
The essence of constraint propagation.
Theoretical Computer Science, 221.

Benhamou, F. (1996).
Heterogeneous constraint solvings.
In Proceedings of the 5th International Conference on Algebraic and Logic Programming, pages 62–76.

Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F. (1999).
Revisiting hull and box consistency.
In Proceedings of the 16th International Conference on Logic Programming, pages 230–244.

Références II

Bessière, C. and Régin, J.-C. (1996).
Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems.
In Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, volume 1118 of Lecture Notes in Computer Science. Springer.

Bonlarron, A., Calabrèse, A., Kornprobst, P., and Régin, J.-C. (2023).
Constraints first : A new mdd-based model to generate sentences under constraints.
In 32nd Interrnational Joint Conference on Artificial Intelligence, IJCAI 2023.

Références III

Boussemart, F., Hemery, F., Lecoutre, C., and Sais, L. (2004).
Boosting systematic search by weighting constraints.
In Proceedings of the 16th Eureopean Conference on Artificial Intelligence, (ECAI'2004), pages 146–150. IOS Press.

Brélaz, D. (1979).
New methods to color the vertices of a graph.
Communications of the ACM, 22(4) :251–256.

Cousot, P. and Cousot, R. (1976).
Static determination of dynamic properties of programs.
In Proceedings of the 2nd International Symposium on Programming, pages 106–130.

Références IV

Cousot, P. and Cousot, R. (1992).
Abstract interpretation frameworks.
Journal of Logic and Computation, 2(4) :511–547.

Ebcioglu, K. (1984).
An expert system for schenkerian synthesis of chorales in the style of j. s. bach.
In 1984 International Computer Music Conference, ICMC 1984.

Gérault, D., Lafourcade, P., Minier, M., and Solnon, C. (2020).
Computing AES related-key differential characteristics with constraint programming.
Artificial Intelligence, 278.

Références V

Granger, P. (1992).
Improving the results of static analyses of programs by local decreasing iterations.
In Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science.

Haralick, R. M. and Elliott, G. L. (1979).
Increasing tree search efficiency for constraint satisfaction problems.
In Proceedings of the 6th International Joint Conference on Artificial intelligence (IJCAI'79), pages 356–364. Morgan Kaufmann Publishers Inc.

Références VI

Jeannet, B. and Miné, A. (2009).
Apron : A library of numerical abstract domains for static analysis.
In Proceedings of the 21th International Conference Computer Aided Verification (CAV 2009).

Lauriere, J.-L. (1978).
A language and a program for stating and solving combinatorial problems.
Artificial Intelligence, 10(1) :29 – 127.

Mackworth, A. K. (1977).
Consistency in networks of relations.
Artificial Intelligence, 8(1) :99–118.

Références VII

Miné, A. (2004).
Relational abstract domains for the detection of floating-point run-time errors.
In Proceedings of the European Symposium on Programming (ESOP'04), volume 2986 of Lecture Notes in Computer Science, pages 3–17. Springer.

Pachet, F. and Roy, P. (1999).
Automatic generation of music programs.
In 5th International Conference on Principles and Practice of Constraint Programming - CP'99, pages 331–345.

Références VIII

- Régin, J.-C. (1994).
  A filtering algorithm for constraints of difference in csps.
  In Proceedings of the 12th National Conference on Artificial Intelligence (Vol. 1), pages 362–367.

- Schulte, C. and Tack, G. (2001).
  Implementing efficient propagation control.
  In Proceedings of the 3rd workshop on Techniques for Implementing Constraint Programming Systems.