Au-delà des produits cartésiens de domaines : l'exemple des octogones

Marie Pelleau, Charlotte Truchet, Frédéric Benhamou

Laboratoire d'Informatique de Nantes-Atlantique (LINA), UMR 6241
Université de Nantes, CNRS
2 rue de la Houssinière, Nantes, France
firstName.lastName@univ-nantes.fr

Résumé

Cet article propose une nouvelle représentation pour les domaines des variables continues s'inspirant des travaux de l'Interprétation Abstraite (IA). Il existe de nombreux domaines abstraits et à l'inverse des domaines en Programmation par Contraintes (PPC) certains prennent en compte des dépendances linéaires entre les variables. Les domaines abstraits peuvent être vus comme une implantation de sous-langages de contraintes. Nous montrons que les outils basiques de résolution en PPC (consistance, split, ...) se traduisent naturellement sur les domaines abstraits. Nous illustrerons cette adaptation avec un exemple détaillé, celui des octogones. Les octogones font partie de la famille des domaines abstraits faiblement relationnels (non cartésien) et sont une restriction des polyèdres où seules les contraintes de la forme $\pm V_i \pm V_j \leq c$ sont autorisées.

1 Introduction

Pour les problèmes à variables continues, le formalisme des contraintes par intervalles repose sur une encapsulation des réels dans l'ensemble des intervalles à borne flottante. Le manque de précision des intervalles peut générer une surapproximation des solutions (*e.g.* plusieurs occurrences d'une même variable, contraintes non linéaires). Plusieurs méthodes ont été proposées pour réduire cette surestimation utilisant des évaluateurs pour les contraintes non linéaires, une relaxation sûre des contraintes quadratiques, ou plus récemment la monotonie des contraintes.

Plutôt que d'essayer d'améliorer les méthodes par intervalles, nous avons décidé d'étudier l'utilisation des domaines abstraits de l'Interprétation Abstraite (IA) comme représentation en Programmation par Contraintes (PPC). L'IA traite de très gros programmes et repose notamment sur la notion de domaines abstraits, qui sont des surapproximations des traces de programmes, pouvant être relationnels (non cartésien). L'idée est d'utiliser ces domaines abstraits en PPC afin d'avoir une représentation plus précise que les intervalles lors de la résolution de problèmes continus. Les domaines des variables étant plus réduits, les solutions sont plus rapidement trouvées. En d'autres termes, améliorer la contraction pour avoir une résolution plus rapide. Afin de résoudre un problème avec une représentation des domaines différente, il est nécessaire de redéfinir certains outils basiques de la PPC. Ce papier donne des définitions générales pour ces outils.

Afin de répondre à ces questions, nous présenterons section 2, une manière générique d'étendre les domaines abstraits à la PPC. Section 3, nous illustrerons cette adaptation avec un exemple détaillé de domaine abstrait, celui des octogones. Nous définirons les outils nécessaires à la résolution de problèmes continus à l'aide des octogones, tel que la consistance octogonale. La section 4 présente les résultats obtenus avec les octogones.

1.1 Notations et rappels

On se contentera de rappeler les notations et définitions nécessaires pour la suite.

On considère un Problème de Satisfaction de Contraintes (CSP) sur les variables $\mathcal{V} = (V_1...V_n)$, prenant leurs valeurs dans les domaines $\mathcal{D} = (D_1...D_n)$, et avec les contraintes $(C_1...C_p)$. L'ensemble des instanciations possibles pour les variables est $D = D_1 \times ... \times D_n$. Les solutions du CSP sont les éléments de D qui satisfont les contraintes. On note $\mathcal{S}$ l'ensemble des solutions, $\mathcal{S} = \{(v_1...v_n) \in \mathcal{D}, \forall i \in \{1 \dots p\}, C_i(v_1...v_n)\}$,

et $\mathcal{S}_{C_i}$ les solutions pour la contrainte C_i.

Définition Soient $V_1...V_n$ des variables de domaines discrets finis $D_1...D_n$, et C une contrainte.
Les domaines $D_1...D_n$ sont dits *arc-consistants généralisés* (GAC) pour C ssi $\forall i \in \{1...n\}, \forall v \in D_i, \forall j \neq i, \exists v_j \in D_j$ tel que $C(v_1, v_2, ..., v_{i-1}, v, v_{i+1}, ...v_n)$.
Les domaines sont dits *bornes-consistants* (BC) pour C ssi $\forall i \in \{1...n\}$, D_i est un intervalle entier, $D_i = \{a_i...b_i\}$ avec $a_i, b_i \in \mathbb{N}$, et la condition précédente tient pour $v = a_i$ and $v = b_i$.

Définition Soient $V_1...V_n$ des variables de domaines continus représentés par les intervalles $D_1...D_n \in \mathbb{I}$, et C une contrainte. Les domaines $D_1...D_n$ sont dits *Hull-consistant* ssi $D_1 \times ... \times D_n$ est la plus petite boite flottante contenant les solutions.

La prochaine partie introduit brièvement l'IA et les notions utiles afin d'utiliser les domaines abstraits en PPC. Pour une présentation plus complète voir [6]

1.2 Interprétation Abstraite

L'IA est un domaine de la sémantique traitant la preuve de programmes. La correction de programmes ne pouvant être prouvée génériquement, l'IA met en place des méthodes permettant d'analyser automatiquement certaines propriétés d'un programme. Cette analyse est appelée *statique* car elle est effectuée au moment de la compilation. L'analyse repose sur l'approximation de toutes les exécutions possibles d'un programme. Si l'intersection de cette approximation et des zones dites dangereuses est vide, on sait que le programme est correct.

L'idée principale est d'abstraire les valeurs des variables. Plus précisément, l'ensemble des valeurs prises par une variable tout au long d'une exécution, appelé *sémantique concrète*, est remplacé par une approximation, appelée *sémantique abstraite*. Par exemple, les valeurs possibles pour une variable flottante peuvent être remplacées par un intervalle à bornes flottantes les contenant. Plusieurs opérateurs sont définis sur les domaines abstraits permettant de modéliser les effets des instructions du programme sur les variables. Cela entraine souvent une perte de précision notamment lors de l'analyse d'une boucle.

L'outil théorique sur lequel repose les domaines abstraits est la notion de treillis :

Définition Une relation $\sqsubseteq$ sur un ensemble non vide E est un *ordre partiel* (po ou poset) ssi elle est réflexive, antisymétrique et transitive. Un ordre partiel $(E, \sqsubseteq)$ est un *treillis* ssi pour $a, b \in E$, le couple $\{a, b\}$ a une plus petite borne supérieure et une plus grande borne inférieure. Il est *complet* ssi tout sous-ensemble a une plus petite borne supérieure.

Un treillis complet a un plus petit élément et un plus grand élément. Les domaines abstraits doivent être des treillis ou un ordre partiel complet. Une caractéristique importante des domaines abstraits est qu'ils sont reliés par des connexions de Gallois :

Définition Soient deux domaines abstraits E_1 et E_2 (chacun étant un po), une connexion de Gallois est définie par deux morphismes $\alpha : E_1 \to E_2$ et $\gamma : E_2 \to E_1$ tels que

- α et γ sont monotones,
- $\forall X_1 \in E_1, X_2 \in E_2, \alpha(X_1) \sqsubseteq X_2 \iff X_1 \sqsubseteq \gamma(X_2)$.

Remarque Les connexions de Gallois ne perdent pas d'éléments : $(\alpha \circ \gamma)(X_2) \sqsubseteq X_2$ et $X_1 \sqsubseteq (\gamma \circ \alpha)(X_1)$.

De nombreux domaines abstraits de formes différentes ont été définis. Ils peuvent prendre en compte plusieurs variables. Ils peuvent être non-relationnels, c'est-à-dire être un produit cartésien de domaines abstraits. Le produit cartésien d'intervalles est un exemple de domaine abstrait non-relationnel.

Mais ils peuvent aussi être relationnels, *e.g.* non cartésiens, comme le domaine abstrait Polyèdre qui prend en compte des relations linéaires entre les variables. Définir une connexion de Gallois entre un polyèdre et un domaine concret revient à résoudre des contraintes linéaires. Un autre exemple de domaine abstrait relationnel est celui des Octogones, une restriction des polyèdres, où seules les contraintes de la forme $\pm V_i \pm V_j \leq c$ avec c une constante, sont autorisées. Ce domaine est moins précis mais moins coûteux en terme de temps de calcul [9]. Beaucoup d'autre domaines abstrait ont été définis et étudiés, voir [8]. La raison pour laquelle il existe tant de domaines abstraits est la recherche d'un compromis entre précision et temps de calcul : un domaine abstrait précis offre une analyse plus fine mais requiert un plus grand temps de calcul.

La PPC et l'IA sont deux domaines différents, pourtant dans ces deux domaines les théories sous-jacentes reposent sur des notions de treillis et de point fixe. De plus, ces deux domaines calculent une sur-approximation d'un ensemble, à la différence qu'en IA on ne cherche pas toujours à être le *plus* précis possible mais à être *assez* précis afin d'éviter des zones dangereuses. En pratique le point fixe est rarement atteint en IA, ce qui permet d'analyser de très gros programmes. Notre idée est de profiter des points communs pour utiliser les domaines abstraits en PPC, et ainsi pouvoir s'abstraire des produits cartésiens lors de la résolution de problèmes continus.

2 Domaines abstraits pour la PPC

Un domaine abstrait peut être vu comme un sous-langage de contraintes permettant de définir des formes géométriques (*e.g.* les contraintes linéaires définissent les polyèdres). Des solveurs spécifiques sont utilisés pour implanter les domaines abstraits (*e.g.* la programmation linéaire pour les polyèdres). Ces sous-langages, adaptés à la PPC, peuvent calculer une approximation complète des solutions si une consistance dédiée est définie. En IA, le choix du domaine abstrait dépend des propriétés du programme à analyser [5]. La conclusion de [5], est qu'il faut considérer plusieurs domaines abstraits en même temps pour permettre un passage à l'échelle. Ainsi chaque domaine abstrait traite les expressions pour lesquelles il est le plus adapté. Ces travaux montrent une comparaison instructive du compromis entre expressivité et complexité de chaque langage de contraintes. La PPC devrait bénéficier de ce résultat.

Nous donnons ici une définition généralisée de la consistance et de l'opérateur de coupe, puis nous proposons une généralisation des domaines en PPC en domaines abstraits.

2.1 Consistance

Soit E un sous-ensemble de $\mathcal{P}(\mathcal{D})$, avec l'inclusion comme ordre partiel. On notera E^f pour $E\backslash\emptyset$. Par la suite, nous restreindrons les définitions aux cas où E est clos par intersection, car cela est suffisant pour les cas classiques.

Exemple Notons $\mathbb{F}$ l'ensemble des nombres flottants suivant la norme IEEE [7]. Soient $a, b \in \mathbb{F}$, on peut définir $[a, b] = \{x \in \mathbb{R}, a \leq x \leq b\}$ l'intervalle réel délimité par a et b, et $\mathbb{I} = \{[a, b], a, b \in \mathbb{F}\}$ l'ensemble de tous les intervalles à bornes flottantes. $(\mathbb{I}^n, \subset)$ est clos par intersection et forme un treillis. Comme montré par la suite, ceci est l'ensemble de base pour la résolution de contraintes continues.

Définition (et proposition) Soit C une contrainte. Un élément e est E-consistant pour C ssi il est le plus petit élément pour $\mathbf{C_{E,C}} = \{\mathbf{e'} \in \mathbf{E}, \mathcal{S}_\mathbf{C} \subset \mathbf{e'}\}$. Si E est clos par intersection, $\mathbf{C_{E,C}}$ est un treillis complet et il existe un unique élément E-consistant pour C dans E. S'il existe, cet élément est noté $\mathcal{C}_{E,C}$.

Preuve Soit $\mathcal{C}_{E,C} = \cap_{e \subset E, \mathcal{S}_C \subset e} e$. L'existence et l'unicité se déduisent directement du fait que E soit clos par intersection.

Définition Soit $e \in E$, e est E-consistant pour $C_1...C_p$ (ou un sous-ensemble) ssi c'est le plus petit élément de $\{e' \in E, \mathcal{S}_{E,C_1 \wedge ... \wedge C_p} \subset e'\}$. S'il existe, il est noté $\mathcal{C}_{E,C_1 \wedge ... \wedge C_p}$ ou $\mathcal{C}_E$ s'il n'y a pas d'ambiguïtés.

Proposition 2.1 *Si E est clos par intersection, alors c'est un treillis complet et $\mathcal{C}_E$ existe et est unique. De plus, l'ensemble de tous les $\mathcal{C}_{E,C_{i_1} \wedge ... \wedge C_{i_k}}$ pour $i_1...i_k \in \{1...p\}$ forme un treillis par inclusion et $\mathcal{C}_{E,C_1...C_p}$ est le plus petit élément.*

Preuve Soient $i, j \in \{1..p\}$, $\mathcal{C}_{E,C_i}$, $\mathcal{C}_{E,C_j}$ admet un plus petit élément $\mathcal{C}_{E,C_i \wedge C_j} \subset \mathcal{C}_{E,C_i} \cap \mathcal{C}_{E,C_j}$ et un plus grand élément $\cap_{e \in E, \mathcal{C}_{E,C_i} \subset e, \mathcal{C}_{E,C_j} \subset e} e$.

La proposition est plutôt formelle, mais elle assure que, si chaque contrainte d'un CSP vient avec un propagateur, il suffit d'appliquer ces propagateurs de manière récursive, peu importe l'ordre, jusqu'à ce que le point fixe soit atteint, pour obtenir la consistance de ce CSP. Une proposition similaire a déjà été définie pour les consistances existantes dans [1] ou [2].

Nous montrons ci-dessous que les principales consistances existantes, discrètes ou continues, sont incluses dans la définition.

Proposition 2.2 *Soit $S(\mathbb{N}) = \mathcal{P}(\mathbb{N})^n$. La $S(\mathbb{N})$-consistance correspond à la consistance d'arc généralisée (GAC).*

Preuve Nous prouvons d'abord que GAC$\Longrightarrow S(\mathbb{N})$-consistant. Soit $D = D_1 \times ... \times D_n$ GAC pour une contrainte C. D contient évidemment toutes les solutions, il reste donc à prouver que c'est le plus petit élément de $S(\mathbb{N})$. Soit $D' \in S(\mathbb{N})$ strictement plus petit que D, il existe donc un i tel que $D'_i \subsetneq D_i$ et un $v \in D_i \backslash D'_i$. Puisque $v \in D_i$ et D est GAC, il existe aussi $v^j \in D_j$ pour $j \neq i$ tel que $(v^1...v...v^n)$ soit une solution. Cette solution n'est pas dans D' et donc $D' \subsetneq D$ perd des solutions.
Nous prouvons maintenant que $S(\mathbb{N})$-consistant $\Longrightarrow$ GAC. Soit $D = D_1 \times ... \times D_n$ $S(\mathbb{N})$-consistant pour une contrainte C. Soient $i \in \{1...n\}$ et $v \in D_i$. Supposons par l'absurde que pour chaque $v^j \in D_j$, $(v^1...v...v^n)$ ne soit pas une solution : on peut construire l'ensemble $D_1 \times ... \times (D_i \backslash \{v\}) \times ...D_n$ strictement plus petit que D contenant toutes les solutions. Donc D n'est pas le plus petit élément. Il ne peut donc pas exister de tels i et v, et D est GAC.

Proposition 2.3 *Soit $\mathcal{I}(\mathbb{N}) = \{\{\underline{d_1}...\overline{d_1}\} \times ... \times \{\underline{d_n}...\overline{d_n}\}, \underline{d_i}, \overline{d_i} \in \mathbb{N}\}$. La $\mathcal{I}(\mathbb{N})$-consistance est la consistance de bornes.*

Proposition 2.4 *La $\mathbb{I}^n$-consistance est la hull-consistance.*

La définition de consistance abstraite généralise ainsi les consistances existantes. Pour tout $E \subset \mathcal{P}(D)$ clos par intersection, la consistance est bien définie.

2.2 Opérateur de coupe

La consistance n'étant pas suffisante pour résoudre un CSP, il faut aussi un opérateur de coupe.

Définition Soit $(E, \subset)$ un poset. Un *opérateur de coupe* est un opérateur $\oplus : E \to \mathcal{P}(E)$ tel que $\forall e \in E$,
- $|\oplus(e)|$ est fini, $\oplus(e) = \{e_1...e_k\}$,
- $\cup_{1 \leq j \leq k} e_j = e$,
- $\forall j \in \{1...k\}$, $e \neq \emptyset \Longrightarrow e_j \neq \emptyset$,
- $e_j = e \Longrightarrow e$ est le plus petit élément de E^f.

La première condition est nécessaire pour que la résolution termine (largeur de l'arbre de recherche finie). La deuxième condition assure que la coupe ne perd pas de solutions. $\oplus(e)$ peut être une partition de e mais ceci n'est pas nécessaire pour prouver la complétude de la résolution. La troisième condition est purement technique, elle permet de garder l'ensemble vide pour caractériser l'inconsistance. La quatrième condition assure que l'opérateur de coupe coupe effectivement : il est interdit de ne pas couper.

Remarque Il est important de noter que la définition implique : si e n'est pas le plus petit élément de E^f, $e_k \subsetneq e$.

Nous montrons ci-dessous que l'instanciation discrète et la bissection en continu sont incluses dans la définition. Nous utiliserons les notations suivantes pour les domaines abstraits cartésiens : soit $\oplus_1 : E_1 \to \mathcal{P}(E_1)$ un opérateur pour un domaine abstrait E_1. Soient E_2 un autre domaine abstrait et Id la fonction identité pour E_2. On note $\oplus_1 \times Id$ l'opérateur sur $E_1 \times E_2$ tel que $\oplus_1 \times Id(e_1, e_2) = \cup_{e \in \oplus_1(e_1)} e \times e_2$. On notera aussi Id^i pour le produit cartésien de i fois Id.

Exemple L'instanciation d'une variable discrète est un opérateur de coupe sur $\mathcal{P}(\mathbb{N})$: $\oplus_{\mathcal{P}(\mathbb{N})}(d) = \cup_{v \in d}\{v\}$. Pour chaque $i \in \{1...n\}$, l'opérateur $\oplus_{S(\mathbb{N}),i}(d) = Id^{i-1} \times \oplus_{\mathcal{P}(N)} \times Id^{n-i-1}$, avec $\oplus_{\mathcal{P}(N)}$, à la $i$ème place, un opérateur de coupe. Cela revient à choisir une variable V_i et une valeur v dans D_i.

Exemple La même construction permet de définir la coupe en continu en prenant comme opérateur de base sur $\mathbb{I}$: $\oplus_{\mathbb{I}}(I) = \{I^{\vdash}, I^{\dashv}\}$, avec $I^{\vdash}$ et $I^{\dashv}$ deux sous intervalles non vides de I avec n'importe quel moyen de gérer les erreurs d'arrondi sur les flottants, sous réserve que $I^{\vdash} \cup I^{\dashv} = I$.

2.3 Domaines abstraits

Définition Un *domaine abstrait* $D^\sharp$ est défini par :
- un treillis complet E,
- une séquence d'opérateurs de coupe sur E,
- une concrétisation $\gamma : D^\sharp \to D^\flat$, et une abstraction $\alpha : D^\flat \to D^\sharp$ formant une connexion de Gallois *avec un domaine non-relationnel*,
- une fonction strictement décroissante $\tau : D^\sharp \to \mathbb{R}$.

La connexion de Gallois relie le domaine abstrait à une représentation représentable en machine des domaines du CSP. La fonction τ correspond à une mesure du domaine abstrait. C'est un artifice technique pour exprimer la précision du domaine abstrait utilisé pour la terminaison de la résolution.

Les domaines abstraits peuvent être définis indépendamment des domaines d'un CSP. Ils sont destinés à déterminer la forme de la représentation des domaines. Ils peuvent être cartésiens, mais cela n'est plus obligatoire. Notons que nous n'avons pas défini les propagateurs, en effet ils ne peuvent être défini génériquement car ils dépendent des contraintes et de la forme du domaine abstrait choisi. Ils doivent être *ad hoc*.

Il est maintenant possible de définir un solveur abstrait comme une alternance de propagations et de coupes, voir figure 1. Notons que le choix de l'opérateur de coupe à une certaine profondeur doit être stocké afin de garder le solveur cohérent.

Proposition 2.5 *Si E est clos par intersection (H1), n'a pas de chaine infinie décroissante (H2) et si $r \in \tau(E^f)$, alors l'algorithme de résolution de la figure 1 termine et est complet.*

La preuve est assez technique, nous ne donnerons ici que l'intuition. Supposons que l'arbre de recherche soit infini. Par la définition de l'opérateur de coupe, sa largeur est finie, cela signifie qu'il existe une branche infinie. Le long de cette branche, les domaines abstraits sont strictement décroissants tant que e n'est pas le plus petit élément de E^f. L'algorithme converge donc vers un élément e^∞. Si $e^\infty = \emptyset$, l'algorithme s'arrête (cas de l'échec). Si $e^\infty \neq \emptyset$, comme $r > \tau(E^f)$, $r > \tau(e^\infty)$ et l'algorithme termine (cas du succès). La complétude du solveur est directement prouvée grâce à la définition des opérateurs de coupe.

Cette proposition définit un solveur générique pour n'importe quel domaine abstrait E, à condition que les hypothèses (H1) et (H2) soient vérifiées. L'efficacité de ce solveur dépend bien sûr des algorithmes de consistance de E.

Exemple Soit n fixé, l'ensemble $S(\mathbb{N})$ avec l'opérateur de coupe $\oplus_{S(\mathbb{N}),i}$ pour $i \in \{1...n\}$ est un domaine abstrait vérifiant (H1) et (H2). Afin de modéliser le fait que la résolution termine quand toutes les variables sont instanciées, on peut prendre $\tau_1(e) = max(|D_i|, i \in \{1...n\})$ et $r = 1$. Comme vu précédemment, la $S(\mathbb{N})$-consistance est GAC.

int $j \leftarrow 0$ */*profondeur de l'arbre de recherche*/*
int op[] */*à une profondeur j, stocke l'index de l'opérateur de coupe choisi, initialisé uniformément à 0*/*
int width[] */*à une profondeur j, stocke la largeur de l'arbre déjà exploré, initialisé uniformément à 0*/*
abstract domain $e \in E$
$e = \alpha(D)$ */*initialisation aux domaines concrets*/*
répéter
  répéter
    choisir un opérateur de coupe $\oplus_{E,i}$, op[j]$\leftarrow i$
    $e \leftarrow \oplus_{E,i}(e)_{\text{width[j]}}$ */*point backtrackable*/*
    $e \leftarrow E$-consistance(e)
    j++
  jusqu'à e est vide **ou** $\tau(e) \leq r$
  si e est vide **alors**
    width[j]++
  fin si
jusqu'à width[j] $>$ | $\oplus_{\text{op}[j],E}$ | **ou** $\tau(e) \leq r$
si $\tau(e) \leq r$ **alors**
  retourner $\gamma(e)$ */*concrétisation de e*/*
sinon
  retourner échec
fin si

FIGURE 1 – Résolution avec des domaines abstraits. L'ensemble des opérateur de coupe est $\{\oplus_{E,i}, 1 \leq i \leq k\}$. L'exécution s'arrête quand la précision r est atteinte. Chaque fois qu'un opérateur de coupe est choisi, la partition obtenue par l'application de cet opérateur est marquée comme backtrackable.

Exemple Soit n fixé, l'ensemble $\mathbb{I}^n$ avec l'opérateur de coupe $\oplus_{\mathbb{I}^n,i}$ pour $i \in \{1...n\}$ est un domaine abstrait vérifiant (H1) et (H2). Afin de modéliser le fait que la résolution termine quand une certaine précision r est atteinte, on peut prendre $\tau_3(e) = \max(\overline{d_i} - \underline{d_i}$, pour $D_i = [\underline{d_i}, \overline{d_i}], i \in \{1...n\})$ et s'arrêter quand $\tau_3 \leq r$. La résolution sur $\mathbb{I}^n$ correspond à celle utilisée généralement dans les solveurs continus avec la Hull-consistance.

Ces deux exemples montrent comment retrouver les domaines abstraits de bases, qui sont cartésiens.

Il est possible de construire de nouveaux domaines abstraits. Nous détaillerons dans la prochaine section les octogones, la nouveauté est que ce domaine n'est pas cartésien (relationnel).

3 Octogones

Les octogones ont été introduits par Antoine Miné [9] et sont implantés dans Apron [1]. Nous présentons ici une façon de les adapter à la PPC en définissant une consistance, un opérateur de coupe et une précision octogonale.

Définition Soit $\mathcal{V} = (V_1 \dots V_n)$ un ensemble de variables, on appelle *contrainte potentielle* les contraintes de la forme $V_i - V_j \leq c$ avec c une constante.
On appelle *contrainte octogonale* les contraintes de la forme $\pm V_i \pm V_j \leq c$ avec c une constante.

Remarque Les contraintes potentielles et les contraintes d'intervalle ($V_i \geq a$, $V_i \leq b$) sont des cas particuliers des contraintes octogonales.

Définition Un octogone *Oct* est un ensemble de points satisfaisant une conjonction de contraintes octogonales.

Différentes représentations peuvent être utilisées pour les octogones, nous en présenterons trois, la première dérivant de la définition donnée par Miné. Nous proposons une deuxième représentation qui permet d'utiliser naturellement les techniques de contraintes sur les intervalles. Quant à la troisième, elle est utilisée et décrite par Miné. Ces trois représentations sont illustrées figure 2.

Octogone comme ensemble de contraintes octogonales Plutôt que de stocker tous les points satisfaisants une conjonction de contraintes, un octogone peut être représenté par l'ensemble des contraintes octogonales qu'il satisfait. Notons qu'un octogone a au plus $2n^2$ côtés en dimension n et donc qu'il y a au plus $2n^2$ contraintes octogonales non redondantes le décrivant en dimension n. Il est facile de déterminer si une contrainte octogonale est redondante par rapport à une autre, en effet si il existe deux contraintes avec la même partie gauche on conservera celle avec la plus petite partie droite (par exemple, $V_i - V_j \leq 10$ est redondante par rapport à $V_i - V_j \leq 2$). Un exemple de cette représentation est donné figure 2(a).

Octogone comme intersection de plusieurs boites La forme particulière des contraintes octogonales permet de représenter les octogones comme l'intersection de plusieurs boites, celle de la base d'origine de vecteurs $u_1 ... u_n$, qu'on appellera canonique notée B_{Can}, et celle de bases tournées. À chaque couple de vecteurs de base u_i, u_j avec $i < j$, correspond une base tournée notée $B^{i,j}_{Rot}$.

Définition La base $B^{i,j}_{Rot}$ est obtenue par une rotation d'angle $\alpha = \pi/4$ des deux vecteurs de base u_i, u_j.

Remarque Cette rotation est effectuée en remplaçant symboliquement dans chaque contrainte C_k la

1. http ://apron.cri.ensmp.fr/

variable V_i par $\cos(\alpha)V_i^{i,j} - \sin(\alpha)V_j^{i,j}$ et V_j par $\sin(\alpha)V_i^{i,j} + \cos(\alpha)V_j^{i,j}$, avec $V_i^{i,j}$ et $V_j^{i,j}$ les nouvelles coordonnées de $B_{Rot}^{i,j}$ correspondant aux variables V_i, V_j. On notera $C_k^{i,j}$ cette nouvelle contrainte.

Cette représentation revient à transformer un CSP initial en un autre CSP contenant :

- les variables initiales $(V_1...V_n)$;
- les variables tournées $(V_1^{1,2}, V_2^{1,2}, V_1^{1,3}...V_n^{n-1,n})$, où $V_i^{i,j}$ est la i-ème variable de la base $B_{Rot}^{i,j}$;
- les domaines initiaux $(D_1...D_n)$;
- les domaines tournés $(D_1^{1,2}, D_2^{1,2}, D_1^{1,3}...D_n^{n-1,n})$;
- les contraintes initiales $(C_1...C_p)$;
- les contraintes tournées $(C_1^{1,2}...C_1^{n-1,n}...C_p^{n-1,n})$;

Notons que pour $i = j$, $V_l^{i,j} = V_l$, $D_l^{i,j} = D_l$ et $C_k^{i,j} = C_k$ avec $i, j, l \in \{1...n\}$ et $k \in \{1...p\}$. De même, pour $l \neq i$ et $l \neq j$, $V_l^{i,j} = V_l$ et $D_l^{i,j} = D_l$.

Par la suite on notera $V_1...V_n^{n-1,n}$ l'ensemble des variables de toutes les bases prenant leurs valeurs dans les domaines $D_1...D_n^{n-1,n}$ et $C_1...C_p^{n-1,n}$ l'ensemble de toutes les contraintes.

Un exemple en dimension 2 est donné figure 2(b).

Octogone avec une DBM Un octogone peut être stocké à l'aide d'un matrice à différences bornées (DBM) comme proposé par Miné.

Définition Une DBM est une matrice carrée $n \times n$. L'élément à la ligne i et colonne j, avec $i, j \in \{1 \dots n\}$, est égal à c s'il existe une contrainte $V_j - V_i \leq c$, et $+\infty$ sinon.

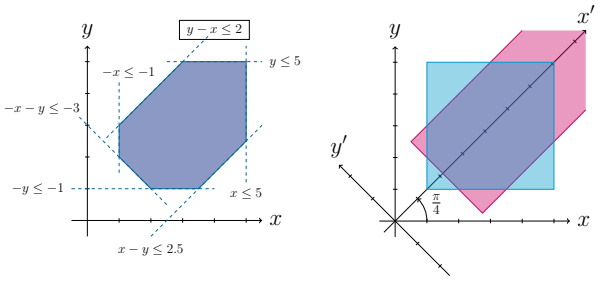
Afin de représenter un octogone avec une DBM, il est nécessaire de réécrire les contraintes octogonales en contraintes potentielles.

Soit $\mathcal{V}' = (V'_1 \dots V'_{2n})$ un ensemble contenant deux fois plus de variables. Chaque variable V_i a une *forme positive* $V'_{2i-1} = V_i$, et une *forme négative* $V'_{2i} = -V_i$. Ce nouvel ensemble permet de transformer les contraintes octogonales dans $\mathcal{V}$ en contraintes potentielles dans $\mathcal{V}'$.

Exemple La contrainte $V_i + V_j \leq c$ se réécrit $V'_{2i-1} - V'_{2j} \leq c$ et $V'_{2j-1} - V'_{2i} \leq c$.

Grâce à cette réécriture, un octogone peut être représenté par un DBM de dimension $2n$. Par exemple, figure 2(c), l'élément à la ligne x et colonne y correspond à la contrainte $y - x \leq 2$.

Nous avons vu, section 2, que pour utiliser un domaine abstrait en PPC, il est nécessaire de définir une consistance, une séquence d'opérateurs de coupe, des connexions de Gallois et une fonction de précision.



(a) (b)

$$\begin{array}{c} \\ x \\ -x \\ y \\ -y \end{array} \begin{array}{c} \begin{array}{cccc} x & -x & y & -y \end{array} \\ \begin{pmatrix} 0 & -2 & \boxed{2} & -3 \\ 10 & 0 & +\infty & 2.5 \\ 2.5 & -3 & 0 & -2 \\ +\infty & \boxed{2} & 10 & 0 \end{pmatrix} \end{array}$$

(c)

FIGURE 2 – Différentes représentations d'un octogone : représentation géométrique avec les contraintes octogonales 2(a), comme l'intersection de deux boites 2(b) et la DBM 2(c).

3.1 Consistance

Notre but étant d'utiliser une représentation différente pour les domaines, il est nécessaire de redéfinir un certains nombres d'outils essentiels à la résolution. Dans un premier temps nous nous intéresserons à la consistance.

Un ensemble est dit consistant pour un ensemble de contraintes s'il est le plus petit élément contenant l'ensemble des solutions $\mathcal{S}$. Calculer un octogone consistant revient donc à calculer le plus petit octogone contenant les solutions.

Définition En utilisant le CSP transformé, la consistance octogonale peut se définir comme la hull-consistance sur l'ensemble des domaines $D_1...D_n^{n-1,n}$.

En pratique cette définition de la consistance est trop coûteuse en terme de temps de calcul, dû à la multiplication du nombre de variables et de contraintes à propager. En effet, en plus des contraintes de chacune des bases, il faut ajouter les contraintes de *channeling* permettant de propager l'information entre les différentes bases. On définira donc la consistance octogonale sur l'intersection de boites.

Définition Soit C une contrainte, et $i, j \in \{1...n\}$. Notons $\mathcal{B}^{i,j}$ la Hull-consistant boite pour la contrainte tournée $C^{i,j}$, et $\mathcal{B}$ la Hull-consistant boite pour C. L'octogone consistant pour C est l'intersection de toutes les $\mathcal{B}^{i,j}$ et de $\mathcal{B}$.

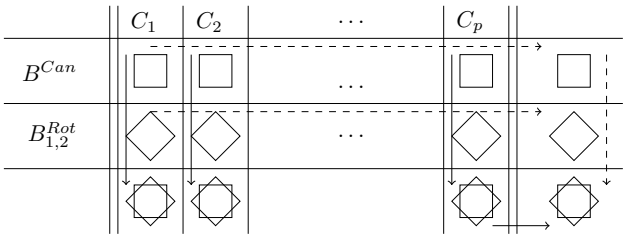


FIGURE 3 – Schéma des possibles heuristiques pour la consistance octogonale. En pointillés la première version, la propagation des deux CSPs est faite en parallèle. En trait plein la seconde version, la consistance octogonale est calculée pour chaque contrainte.

Comme montré sur la figure 3, on peut imaginer deux stratégies. La première, en pointillés, applique la consistance en parallèle dans chaque base, une seule passe sur les contraintes est effectuée, puis récupère l'octogone par intersection des boites obtenues. Étant donné que cette intersection peut réduire les boites, ces deux étapes sont répétées jusqu'à l'obtention du point fixe. N'effectuer qu'une seule passe sur les contraintes d'une base permet d'éviter une convergence lente de la consistance de cette base.

Dans la seconde version, en trait plein sur la figure 3, on calcule la consistance pour une contrainte donnée dans chacune des bases. L'octogone consistant pour cette contrainte est calculé par l'intersection des boites obtenues et est le point de départ pour la prochaine contrainte à propager. Ceci est répété jusqu'à l'obtention du point fixe. Cette seconde version correspond à la version intuitive de la consistance octogonale.

Remarque Quelque soit la version choisie, l'octogone consistant est le même à la fin. Cette représentation des octogones est close par intersection, la preuve est donc directe grâce à la Proposition 2.1.

Il reste un problème, comment réaliser l'intersection de boites ne se trouvant pas dans la même base. Pour palier à ce problème nous utilisons une DBM pour représenter l'octogone, ainsi l'intersection se fait en exécutant une version modifiée de l'algorithme de plus court chemin de Floyd-Warshall sur cette matrice. Chercher le plus court chemin entre V_i et V_j revient à chercher la plus petite constante c pour la contrainte $V_i - V_j \leq c$. En plus des chemins passant par un autre sommet V_k, cette version modifiée prend en compte des chemins supplémentaires dû au doublement des variables dans la DBM. Chaque variable apparait deux fois, sous forme positive et sous forme négative. On sait que $V_i - V_j \leq c$ est équivalent à $2V_i - 2V_j \leq 2c$, or si l'on représente la DBM sous forme de graphe, il n'existe pas de chemins pour la seconde inégalité. Cette absence de

```
Octogone oct
boolean modif /*vrai si domaine est réduit, faux sinon*/
float M[2n][2n] /*DBM associée à oct*/
pour chaque base b de oct faire
  b ← Hull-consistance(b)
  si il y a eu une réduction alors
    Modifier les cases correspondantes de M
    modif ← true
  fin si
fin pour
tant que modif faire
  modif ← false
  M ← Floyd-Warshall-Modif(M)
  pour chaque cellule qui a été modifiée par Floyd-
  Warshall-Modif faire
    Changer le domaine de la variable correspondante
  fin pour
  pour chaque base b dont au moins un des domaines
  a été réduit par Floyd-Warshall-Modif faire
    b ← Hull-consistance(b)
    si il y a eu une réduction alors
      Modifier les cases correspondantes de M
      modif ← true
    fin si
  fin pour
fin tant que
retourner oct
```

FIGURE 4 – Pseudo code de la première version de la consistance octogonale.

chemins rend la version originale de Floyd-Warshall incomplète, d'où la nécessité d'une version modifiée. Comme prouvé dans [9], cet algorithme réduit les domaines des différentes boites sans modifier l'octogone initial. Son exécution sur la DBM permet de simuler la propagation des contraintes octogonales.

La consistance ainsi définie suppose de maintenir les deux représentations différentes des octogones mais permet en pratique un gain de temps.

La figure 4 présente le pseudo-code de la première version de la consistance octogonale. Cet algorithme termine, est correct et est confluent, quel que soit l'ordre dans lequel les consistances sont appliquées l'octogone consistant est le même.

Nous avons maintenant les outils nécessaires à l'approximation extérieure d'un ensemble de solutions à l'aide d'un octogone. Nous n'avons cependant pas encore les éléments essentiels à la résolution d'un CSP à l'aide d'octogones. En particulier, il reste à définir un opérateur de coupe et une précision octogonale.

3.2 Résolution

En continu, les techniques de résolution choisissent généralement un domaine, le coupent en deux (ou plus)

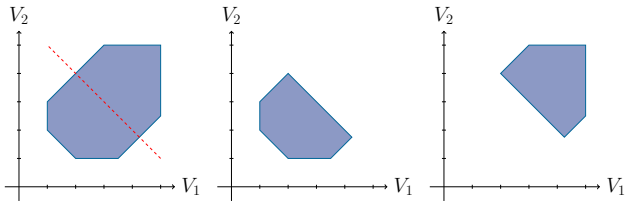


FIGURE 5 – Exemple d'une coupe : l'octogone à gauche est coupé dans la base $B^{1,2}$.

et appliquent une consistance sur les deux boites obtenues. Ces étapes sont répétées tant qu'une certaine précision n'est pas atteinte. Dans le cas de la résolution avec des octogones, il nous faut donc définir les trois points clés de la résolution qui sont l'opérateur de choix du domaine à couper, l'opérateur de coupe et la précision.

Opérateur de choix La représentation d'un octogone comme l'intersection de boites permet ici de définir directement l'opérateur de choix, noté $\mathbb{O}_{Oct}(D_1...D_n, D_1^{1,2}...D_n^{n-1,n})$. En effet, avec cette représentation, les opérateurs de choix usuels sont utilisables. En pratique on choisira le plus grand domaine comme prochain domaine à couper. Deux stratégies sont possibles, la première, est de restreindre le choix aux domaines des variables de la base canonique. La seconde, est d'étendre le choix aux domaines des variables de toutes les bases.

Définition L'opérateur de choix peut être défini comme suit : $\mathbb{O}_{Oct}(D_1...D_n^{n-1,n}) = D_i^{i,j}$ avec $i,j \in \{1...n\}$ et $\overline{d_i^{i,j}} - \underline{d_i^{i,j}} = max(\overline{d_k^{k,l}} - \underline{d_k^{k,l}}$, pour $D_k^{k,l} = [\underline{d_k^{k,l}}, \overline{d_k^{k,l}}], k,l \in \{1...n\})$.

Dans le cas de la première stratégie, on pose $i = j$.

Opérateur de coupe Une fois le domaine à couper choisi, il faut déterminer comment le couper. Toujours en utilisant la représentation de l'intersection des boites, l'opérateur de coupe octogonal peut être défini comme suit :

Définition (et proposition) $\oplus_{Oct}(D_1...D_n^{n-1,n}) = \{D_1...D_i^{i,j\vdash}...D_n^{n-1,n}, D_1...D_i^{i,j\dashv}...D_n^{n-1,n}\}$, avec $D_i^{i,j\vdash}$ et $D_i^{i,j\dashv}$ deux parties non vides de D_i, $D_i^{i,j\vdash} \cup D_i^{i,j\dashv} = D_i$ et $D_i = \mathbb{O}_{Oct}(D_1...D_n^{n-1,n})$.

Ainsi défini, l'opérateur de coupe ne prend pas en compte les corrélations entre les variables des bases. Il faut donc ajouter une contrainte dans les autres bases afin de communiquer cette modification aux autres bases.

```
Octogone oct /*octogone initial*/
queue splittingList ← oct /*liste d'octogones à couper*/
queue acceptedOct ← ∅ /*liste d'octogones acceptés*/
tant que splittingList ≠ ∅ faire
  Octogone octAux ← splittingList.top()
  splittingList.pop()
  octAux ← Oct-consistance(octAux)
  si τ(octAux) < r ou octAux ne contient que des solutions alors
    Ajouter octAux à acceptedOct
  sinon
    Octogone octGauche ← gauche(⊕_Oct(octAux))
    Octogone octDroit ← droit(⊕_Oct(octAux))
    splittingList.add(octGauche)
    splittingList.add(octDroit)
  fin si
fin tant que
retourner acceptedOct
```

FIGURE 6 – Résolution avec des octogones.

Exemple Soit $D_i = \mathbb{O}_{Oct}(D_1...D_n^{n-1,n}), i \in \{1...n\}$ le domaine choisi par l'opérateur de coupe. Il faut ajouter dans toutes les bases obtenues par rotation des vecteurs de bases u_i, u_j la contrainte :

$$\cos(\alpha)V_i^{i,j} - \sin(\alpha)V_j^{i,j} \diamond c \text{ si } i < j$$
$$\sin(\alpha)V_i^{i,j} + \cos(\alpha)V_j^{i,j} \diamond c \text{ si } i > j$$

où $j \in \{1...n\}, \diamond \in \{\leq, \geq\}$ (suivant le côté) et c est la valeur sur laquelle le domaine a été coupé.

Un exemple de coupe est présenté figure 5.

De même que les opérateurs de choix et de coupe, il faut définir ce que signifie la précision octogonale.

Précision Dans la plupart des solveurs continus, la précision est définie comme la taille du plus grand domaine. Pour les octogones, cette définition ne peut être utilisée car elle repose uniquement sur la taille des domaines et ne prend pas en compte la corrélation entre les variables.

Définition (et proposition) Soit $\mathcal{O}$ un octogone, on définit la précision octogonale $\tau(\mathcal{O}) = \min_{1\leq i,j\leq n}(\max_{1\leq k\leq n}(\overline{d_k^{i,j}} - \underline{d_k^{i,j}}))$.

Pour une seule boite, la définition de τ est identique à la précision usuelle. Pour un octogone, nous prenons le minimum des précisions des boites de toutes les bases. Même si techniquement la définition est différente, le sens de la précision octogonale est identique à celui de la précision usuelle : dans un octogone de précision r, tout point est à distance au plus r de l'ensemble des solutions $\mathcal{S}$.

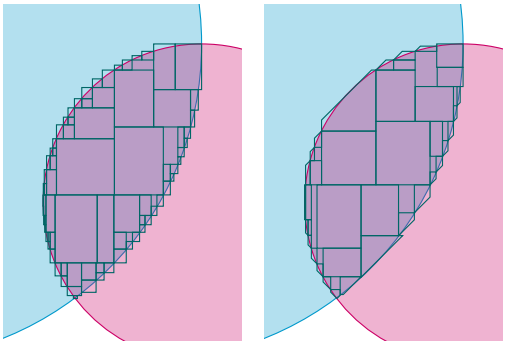


FIGURE 7 – Les résultats obtenus avec une limite de temps (7ms). à gauche la résolution dans $\mathbb{I}^n$ de IBEX. à droite, la résolution octogonale avant la concrétisation.

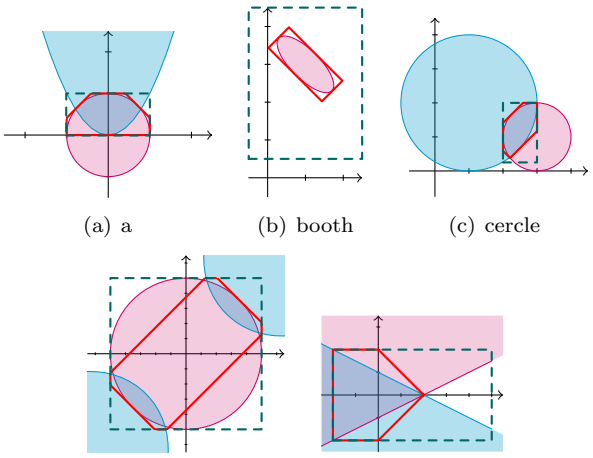


FIGURE 8 – Exemples de problèmes en dimension 2, avec la consistance par intervalles en pointillés vert et la consistance octogonale en rouge.

Résolution La figure 6 décrit l'algorithme de résolution octogonale. Cet algorithme termine et est correct. Il retourne la liste des octogones solutions, c'est-à-dire ceux ne contenant que des solutions ou dont la précision est inférieure à un paramètre r. Notons que dans l'hypothèse où l'on souhaite renvoyer une représentation non relationnelle, la concrétisation peut être faite en utilisant la définition donnée par Miné [9].

4 Résultats

Un prototype de résolution de problèmes continus à l'aide d'octogones comme représentation a été implanté dans IBEX, un solveur continu [4]. Dans cette implantation la consistance octogonale utilise la procédure *HC4-Revise* [3] pour contracter les contraintes de chacune des bases. Les deux versions de consistance octogonale décrites précédemment ont été implantées, et en pratique leur temps d'exécution est équivalent. De même les deux stratégies pour l'opérateur de choix ont été implantées. Le paramètre de précision est fixé à 0.001 ($r = 0.001$). Après la création d'un octogone, les contraintes sont simplifiées afin de réduire les multiples occurrences des variables. Pour cette étape la fonction *Simplify* de Mathematica est utilisée.

La figure 7 compare les résultats obtenus, sur un problème simple d'intersection de deux disques, par d'un côté la résolution usuelle avec les intervalles et de l'autre celle avec les octogones. Pour cette expérimentation, la précision n'est pas fixée et nous utilisons une limite de temps, de 7ms, pour arrêter l'algorithme de résolution.

Le tableau 1 compare le temps CPU, le nombre de nœuds solutions et le nombre de points de choix obtenus par la résolution par intervalles avec *HC4* à ceux obtenus par la résolution par octogones. Deux versions de la résolution octogonale sont présentées, dans la première, Oct-v1, l'opérateur de choix choisit parmi tous les domaines (de la base canonique et des bases tournées), alors que dans la seconde, Oct-v2, seuls les domaines de la base canonique peuvent être choisis.

Ce tableau montre les bons résultats obtenus par la résolution octogonale, en termes de pouvoir de filtrage (petit nombre de points de choix) et de contraction (petit nombre d'octogones solutions). De plus, on constate que la phase de simplification des contraintes améliore grandement les résultats, le temps CPU est réduit grâce à un meilleur filtrage et une meilleure contraction. Notons que les temps n'incluent pas la simplification de Mathematica. En pratique ce temps est négligeable.

La figure 8 compare sur les problèmes testés, la consistance obtenue avec l'algorithme HC4 sur les intervalles et la consistance octogonale. Pour tous ces exemples, la consistance octogonale est plus précise que la consistance usuelle en continu. Notons que HC4 calcule la hull-consistance sur l'ensemble des contraintes primitives sans les décomposer. Cette forme compacte peut générer une surapproximation notamment lors d'occurrences multiples des variables.

Sur le premier exemple, 8(a), la consistance octogonale ne réduit pas autant qu'espéré, cela est dû aux nouvelles multiples occurrences des variables qui apparaissent dans la contrainte parabolique dans la base tournée. Au contraire, sur le deuxième exemple 8(b), il existe plusieurs occurrences des variables dans la base canonique qui disparaissent dans la base tournée.

			Avec Simplifications				Sans Simplifications			
	$\mathbb{I}^n$		Oct-v1		Oct-v2		Oct-v1		Oct-v2	
a	0.31	*4905* 9809	0.57	*1444* 3175	0.59	*1742* 3483	1.39	*2347* 5171	1.17	*2462* 4923
booth	0.42	*8773* 19421	0.28	*956* 2005	0.37	*1116* 2431	3.18	*4972* 11863	2.64	*5163* 10865
cercle	0.37	*5337* 10673	0.32	*897* 1793	0.32	*880* 1759	1.34	*2330* 4939	1.16	*2399* 4797
hs8	1.2	*18596* 37191	0.7	*2225* 4449	0.82	*2215* 4429	6.36	*10299* 22349	5.2	*10871* 21741
triangle	0.77	*11897* 23793	0.63	*1967* 3933	0.74	*1903* 3805	3.48	*6253* 12529	3.1	*6537* 13073

TABLE 1 – Résultats expérimentaux. Chaque colonne donne le temps CPU en secondes (à gauche), et nombre de point de choix (en bas à droite) et le nombre de nœuds solutions (en haut à droite).

5 Conclusion

Cet article a présenté une nouvelle représentation des domaines s'inspirant des domaines abstraits de l'Interprétation Abstraite. Il définit les outils nécessaires pour résoudre avec des domaines abstraits qui généralisent les domaines habituels de la PPC. De plus, il décrit un exemple détaillé, celui des octogones.

Lors de la création de l'octogone, toutes les bases tournées sont générées. On obtient de meilleurs performances sur de petits exemples, mais cette génération systématique risque de ne pas passer à l'échelle. Dans des travaux futurs, une étude syntaxique des contraintes permettrait de ne générer que certaines bases. Une heuristique d'octogonalisation permettrait la création d'un octogone plus adapté au problème à résoudre en choisissant les contraintes à octogonaliser et un angle de rotation plus adéquat.

Ces travaux sont un premier pas vers l'utilisation des domaines abstraits en Programmation par Contraintes. De plus, les définitions génériques des différents points clés de la résolution laissent entrevoir un cadre uniforme pour la résolution de problèmes mixtes discret/continu.

Références

[1] Krzysztof R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221, 1999.

[2] Frédéric Benhamou. Heterogeneous constraint solvings. In *Proceedings of the 5th International Conference on Algebraic and Logic Programming*, pages 62–76, 1996.

[3] Frédéric Benhamou, Frédéric Goualard, Laurent Granvilliers, and Jean-François Puget. Revisiting hull and box consistency. In *Proceedings of the 16th International Conference on Logic Programming*, pages 230–244, 1999.

[4] Gilles Chabert and Luc Jaulin. Contractor programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[5] Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium of Principles of Programming Languages*, pages 269–282, 1979.

[6] Patrick Cousot and Radia Cousot. Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming*, pages 106–130, 1976.

[7] David Goldberg. What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys*, 23(1) :5–48, 1991.

[8] Antoine Miné. *Domaines numériques abstraits faiblement relationnels.* PhD thesis, École Normale Supérieure, December 2004.

[9] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1) :31–100, 2006.