Département d'Informatique
Université d'Orléans

Rapport de Stage
D.E.A. d'Informatique

Présenté par:

GOUMAIRI Abdelkhalek

soutenu le 8 septembre 1998

Thème

Simplification de contraintes sur les domaines finis

Responsable du stage
Monsieur: Alexandre Tessier

Remerciements

Je remercie Monsieur Alexandre Tessier qui a accepté de diriger ce stage. Ses conseils, ses encouragements et sa gentillesse m'ont été très précieux durant ce travail.

Je dois aussi exprimer toute ma gratitude à Monsieur Gérard Ferrand, Professeur à l'université d'Orléans. Je tiens à le remercier pour sa grande rigueur scientifique et ses conseils éclairés qui m'ont fourni une aide décisive. Sa grande culture scientifique a souvent été l'occasion de discussions enrichissantes.

Enfin, je dédie ce stage à tous les membres de ma famille, et plus particulièrement à ma mère, à qui l'achèvement positif de longues années d'études lui tient le plus à coeur.

Table des matières

Chapitre 1

Introduction

Ce travail entre dans le cadre du projet ESPRIT DiSCiPl (Debugging Systems for Contraints Programming).

Le stage est fait dans l'équipe LOCO (commune à l'INRIA et l'Université d'Orléans) qui est un des partenaires du projet ESPRIT DiSCiPl.

Dans plusieurs parties du projet, le problème de visualisation d'un ensemble de contraintes se pose. Ce problème n'est pas nouveau, en effet, il se pose déjà dans la réalisation de système CLP pour l'affichage des réponses à un but: c'est le problème de présentation. Pour cela, dans le cadre du projet, plusieurs solutions ont été proposées, par exemple: outils graphiques, approximation des domaines, ... etc. Ici, on a choisi la transformation en un ensemble de contraintes équivalent plus lisible.

L'une des tâches de l'équipe LOCO dans le projet DiSCiPl est le diagnostic déclaratif d'erreur (task T.WP2.1, déclarative debugging).

Lors d'une session de diagnostic déclaratif d'erreur l'outil peut-être amené à poser des questions à l'utilisateur, par exemple de de la forme:

$$\exists\ \tilde{y}\ C \longrightarrow a? \qquad \text{où}$$

- C est une conjonction de contraintes.
- a est un atome du programme.
- et $\tilde{y}$ sont les variables libres de C qui ne sont pas libres dans a.

L'objectif est de simplifier (dans le sens rendre le plus simple à comprendre, à lire), la formule $\exists\ \tilde{y}\ C$. Pour cela , on peut éliminer des variables existentielles, éliminer des contraintes redondantes et effectuer des transformations symboliques (plus précisement remplacement d'une formule par une autre simple et équivalente, en utilisant par exemple des techniques de démonsration automatique). Dans ce stage, on ne s'interesse qu'aux deux premiers

points.

Exemple:

On considère la formule $\exists\ y\ C$ où C est l'ensemble de contraintes suivants:

$$\begin{cases} 0 \leq x \\ x \leq y \\ y \leq 5 \\ x \leq 7 \\ x = (3t - 4 + w)(3t + 8 - w) \end{cases}$$

On élimine la variable auxiliaire entre la deuxième et la troixième contrainte, on obtient:

$$\begin{cases} 0 \leq x \\ x \leq 5 \\ x \leq 7 \\ x = (3t - 4 + w)(3t + 8 - w) \end{cases}$$

La contrainte $x \leq 7$ étant redondante, on l'élimine et on obtient:

$$\begin{cases} 0 \leq x \\ x \leq 5 \\ x = (3t - 4 + w)(3t + 8 - w) \end{cases}$$

On effectue maintenant une transformation symbolique et obtient, par exemple, la formule finale simplifiée:

$$\begin{cases} 0 \leq x \\ x \leq 5 \\ x = (3t + 2)^2 - (6 - w)^2 \end{cases}$$

La plate-forme du projet LOCO est Calypso (nouvelle version de CLP(FD)). Calypso est un système de programmation logique avec contraintes sur les do-

maines finis. Les deux plates-formes industrielles du projet (CHIP V5 et Prolog IV) incluent également des contraintes sur les domaines finis.

Actuellement l'étude de la simplification de contraintes sur les domaines finis est quasi-inexistante contrairement aux domaines continus. C'est pour ces raisons que ce stage s'intéresse à la simplification de contraintes sur les domaines finis.

La particularité des domaines finis est que toute variable x est entière et à son domaine borné : $\alpha \leq x \leq \beta$, où α et β sont deux entiers naturels.

Dans un premier temps, on ne s'interesse qu'à l'étude théorique de la simplification de contraintes sur les domaines finis. L'implémentation de l'algorithme proposé dans ce stage peut être le sujet d'un autre travail à compléter.

Le chapitre 2 a pour but de rappeler l'algorithme de Fourier, des résultats concernant cet algorithme, des définitions de base et un rappel sur la programmation en nombre entier.

Le chapitre 3 a pour but de préciser de manière concise les différentes sortes de redondances. Une transformation sera introduite pour simplifier des contraintes redondantes en d'autres qui sont simples à détecter et ensuite à éliminer.

Le chapitre 4 présente le module sortie de CLP(R). Nous détaillerons l'algorithme de Fourier ainsi que l'extension de cet algorithme dûe à Cernicov.

Le chapitre 5 est une extension du chapitre 4, et présente la sortie simplifiée d'un ensemble de contraintes sur les domaines finis.

Chapitre 2

Préliminaires

Ce chapitre a pour but de rappeller des notions de bases qui nous seront utiles dans la suite.

2.1 programmtion logique par contraintes

La programmtion logique par contrainte a été un des développements majeurs dans la programmaton déclarative ces dernières années [10]. L'idée est de combiner la programmation logique avec la résolution de contraintes sur des domaines de calcul tels que l'arithmétique linéaire, l'algèbre de Boole et les domaines finis. Les différents systèmes de programmation logique par contrainte ont été développés dans plusieurs langage de programmation, par exemple Prolog II, Prolog III [4], CLP(R) [11], CHIP [17], prolog IV et CLP(FD) [9]. La programmation logique par contrainte a eu un grand succés dans beaucoup d'applications pratiques, en particulier dans beaucoup de problèmes d'intelligence artificielle et de recherche opérationnelle. La combinaison de la programmation logique et la résolution de contraintes était une réussite pour les deux. Du point de vue de la programmation logique, les techniques de résolution de contraintes augmentent l'efficacité des programmes logiques et la puissance expressive. Les algorithmes efficaces de mathématiques, intelligence artificielle ou recherche opérationnelle peuvent être directement introduits dans le langage de programmation logique. Du point de vue de la résolution de contraintes, un langage de programmation déclarative de haut niveau est disponible en plus du résolveur de contraintes. Cela signifie que le problème à résoudre peut être représenté d'une manière naturelle et déclarative. Le langage logique n'est pas seulement utilisé pour générer les formules de contraintes. Il nous permet aussi de résoudre des problèmes qui ne figurent pas dans le cadre de la résolution de contraintes.

L'idée générale de la résolution de contraintes est de calculer la forme résolue qui représente toutes les solutions d'un ensemble de contraintes données. Une des applications pratiques est la recherche de la solution optimale d'une fonction sous des contraintes. Bien que l'optimisation ne rentre pas dans le cadre de la programmation logique par contrainte originale, elle est présente dans beaucoup de systèmes de programmation logique par contrainte existants.

Dans sa forme classique [11], un programme logique avec contrainte sur une structure S consiste en un ensemble fini de règles de la forme

$$p_0(\vec{t_0}) : -c_1(\vec{u_1}), \ldots, c_m(\vec{u_m}), p_1(\vec{t_1}), \ldots, p_n(\vec{t_n})$$

où les p_i sont des symboles de prédicats différents des symboles de relations dans S, c_j sont des symboles dénotant des relations dans S, et $\vec{t_i}$, $\vec{u_j}$ sont des tuples de termes de S avec des symboles de fonctions. La sémantique déclarative de telles règles est que la tête $p_0(\vec{t_0})$ est logiquement impliquée par la conjonction $c_1(\vec{u_1}) \wedge \ldots \wedge c_m(\vec{u_m})$
$\wedge\, p_1(\vec{t_1}) \wedge \ldots \wedge p_n(\vec{t_n})$ de toutes les contraintes $c_j(\vec{u_j})$ et tous les atomes $p_i(\vec{t_i})$ du corps. Un but a la même forme qu'un corps de régle.

La sémantique opérationnelle est basée sur deux composantes: Un résolveur de contraintes pour les S-relations et une adaptation de la méthode de résolution du programme logique classique. Par exemple, du but ?- $c(t_1), p(t_2)$ et la régle $p(t_3) : -d(t_4), q(t_5)$ on peut dériver le nouveau but ?- $R(c(t_1), t_2 \doteq t_3, d(t_4)), q(t_5)$ pourvu que la contrainte $c(t_1), t_2 \doteq t_3, d(t_4)$ soit solvable dans S et $R(c(t_1), t_2 \doteq t_3, d(t_4))$ est sa forme résolue. La séquence de dérivation consiste en une suite de buts avec contraintes satisfiables. C'est un succés lorsque le dernier but contient seulement des contraintes. Elles sont appelées des contraintes réponses et constituent la sortie du programme. Une séquence de dérivation est finiment échoué si le dernier but n'est pas une contrainte réponse et ne peut être développé.

On dénote par, CLP(S) un langage CLP sur une structure S [11]. La structure S peut contenir des domaines très différents comme le corps des réels, l'anneau des Booléens, les domaines finis, mais elle doit toujours inclure l'ensemble des termes de Herbrand et l'égalité entre deux termes. Au passage on peut noter que Prolog est en fait un langage CLP(S) simple, dont le domaine S est le domaine de Herbrand, et dont l'unique contrainte est l'égalité de deux termes.

Parmi les différents domaines de calcul étudiés, celui des domaines finis semble être le plus prometteur, car il est très utile dans de nombreuses applications industrielles comme les problèmes combinatoires, l'ordonnancement,

les optimisations de stocks, l'aide à la décision, les problèmes booléens. Les domaines finis ont été introduits en CLP par Pascal van Hentenryck dans le langage CHIP vers la fin des années 80. Un domaine fini est tout simplement un ensemble de valeurs, numériques ou symboliques[1], de cardinalite finie: exemple $\{1, 5, 7, 8\}$ ou $\{blanc, noir, rouge\}$. Les contraintes proposées sont des contraintes arithmétiques linéaires ou non linéaires. Habituellement, les contraintes sur les domaines finis sont résolues par des techniques de consistance locale combinées avec énumération.

De nombreux problèmes réels peuvent être exprimés comme la recherche d'une solution sur un domaine fini. En générale dans des problèmes de recherche opérationnelle et d'intelligence artificielle.

Ces problèmes ont en commun un noyau central pouvant s'exprimer comme la recherche d'une solution sur un domaine fini défini par un ensemble de contraintes.

2.2 Algorithme de Fourier

On veut connaître la solubilité de $P = \{Ax \leq b\}$, où A est une matrice $m \times n$, x un vecteur de n variables réelles et b un vecteur de constantes. Pour cela, on va chercher à éliminer les différentes variables x_i de P.

Déroulement de l'algorithme:

Pour éliminer x_i de P, il faut former toutes les paires de contraintes de P telles que x_i apparaît dans ces contraintes avec des coefficients de signe opposé. Si de telles paires n'existent pas, c'est à dire si tous les coefficients de x_i ont le même signe, toutes les contraintes contenant x_i peuvent être enlevées de P. Sinon, on élimine x_i de P au moyen d'une combination linéaire à coefficient positif sur chacune des paires. On obtient ainsi un nouveau système P', formé de contraintes de P ne contenant pas x_i et éventuellement des contraintes obtenues par élimination de x_i. Cette opération est appelée étape de Fourier, et est notée $P \rightarrow_F P'$.

Si toutes les variables x_i peuvent être éliminées de cette manière sans générer de contradiction, c'est à dire une contrainte de la forme $c < 0$ où c est un scalaire positif ou nul, alors P est soluble. Sinon P est insoluble.

Théorème 1 *Soit x une variable et soit P un système d'inégalités linéaires. Si on élimine x par $P \rightarrow_F P'$ alors $\exists x P \Leftrightarrow P'$.*

[1] On ne s'intérésse pas à ce type de valeurs

Démonstration De manière évidente $P \Rightarrow P'$, donc $\forall x\ P \Rightarrow P'$, et comme x n'apparaît pas dans P', $(\exists x\ P) \Rightarrow P'$. On suppose maintenant que s est une solution de P'. On peut partitionner les contraintes de P contenant x sous la forme $l_i \leq x$ et $x \leq r_j$. On a alors $l_i(s) \leq r_j(s)$ pour tout i et tout j et en particulier $max_i l_i(s) \leq min_j r_j(s)$. Une solution de P est alors s augmentée d'une coordonnée correspondant à x et ayant par exemple pour valeur $max_i l_i(s)$. Ainsi $P' \Rightarrow \exists x P$. □

Ainsi une étape de Fourier calcule la projection du polyèdre représenté par les inégalités de P. Une conséquence du théorème précèdent est que l'algorithme de Fourier permet de déterminer la satisfiabilité d'une conjonction d'inégalités linéaires.

On a ainsi:

Théorème 2 *Un ensemble d'inégalités linéaires est incohérent si et seulement si il existe une combinaison linéaire à coefficients positifs de contraintes de P, éliminant toutes les variables de P, et de la forme* $0 \leq -|c|$.

Exemple 1 *Le système* $\begin{cases} 3x \leq 4 \\ -2x \leq -5 \end{cases}$ *est insatisfiable, en effet:*

$$\begin{array}{r} 2 \times (3x \leq 4) \\ \underline{3 \times (-2x \leq -5)} \\ 0 \leq -7 \end{array}$$

Remarques

Une analyse simple de l'algorithme de Fourier montre facilement que le nombre
d'inégalités peut augmenter au cours de son déroulement. C'est pourquoi la complexité au pire de cet algorithme est désastreuse. Le nombre de contraintes générées lors d'une étape d'élimination peut être très important: si les m contraintes de P sont équitablement réparties en $m/2$ contraintes où x_i apparaît avec un coefficient positif et $m/2$ contraintes à coefficient négatif, le nombre de paires, et donc de contraintes générées, est égal à $m^2/4$. Dans le cas le pire, où cela serait à chacune des n étapes d'éliminations, on obtiendrait $4(m/4)^{2^n}$ contraintes finales. La complexité de cet algorithme, à la fois en temps, et en occupation mémoire est donc doublement exponentielle. Pour se convaincre, le lecteur peut observer ce qui se passe si l'on applique l'algorithme de Fourier sur un cocube $\{\pm x_1 \pm x_2 \pm \ldots \pm x_n \leq 1\}$.

Il faut cependant noter qu'une forte proportion des contraintes générées par élimination de variables sont redondantes. Une contrainte est dite redondante par rapport à un ensemble de contraintes P si elle est impliquée par P. Pour améliorer les performances de l'algorithme de Fourier, qui dépendent essentiellement du nombre de contraintes générées à chaque étape, on peut envisager de supprimer les contraintes redondantes à chaque étape. Quelques résultats concernant l'élimination de la redondance sont présentés plus loin. Mais auparavant est présenté un corollaire important du théorème de Fourier.

2.3 Egalités implicites

On peut avoir besoin de connaître des informations plus précises que la simple satisfiabilité d'un ensemble de contraintes linéaires. C'est le cas en particulier de la détérmination des égalités implicites.

Définition 1 *Une contrainte $a_i x \leq \beta_i$ dans P est une* **égalité implicite** *si tout élément x de P satisfait $a_i x = \beta_i$.*

La détérmination des égalités implicites est un problème classique et peut être donnée par des techniques de programmation linéaire. Plus précisement la contrainte $a_i x \leq \beta_i$ dans P est une égalité implicite si et seulement si β_i est retournée comme solution optimale par le programme linéaire $min\{a_i x \mid Ax \leq b\}$. Le programme linéaire utilisé pour identifier les égalités implicites peut être executé en parallèle. Il existe aussi des algorithmes pour identifier toutes les égalités implicites utilisant un programme linéaire simple.

Dans [12], Lassez et Maher ont découvert une propriété cachée de l'algorithme de Fourier: il peut être aussi utilisée pour déterminer les égalités implicites. Ceci est démontré dans le théorème suivant:

Théorème 3 *Une inégalité $a_i x \leq \beta_i$ dans un ensemble P est une égalité implicite ssi l'application de l'algorithme de Fourier génère une inégalité de la forme $0 \leq 0$ comme une combinaison de contraintes de P contenant $a_i x \leq \beta_i$.*

Exemple 2

$$\begin{aligned} x + y &\leq 4 \\ -x \quad &\leq -2 \\ \underline{\quad -y &\leq -2} \\ 0 &\leq 0 \end{aligned}$$

En appliquant le théorème ci-dessus on obtient le système équivalent:

$$\begin{aligned} x + y &= 4 \\ x &= 2 \\ y &= 2 \end{aligned}$$

La détection des égalités implicites permet de simplifier le système en cours. De l'importance théorique de l'algorithme de Fourier on verra plus loin que ce nouveau théorème peut être utilisé pour des résultats concernant l'enveloppe affine et les égalités implicites.

2.4 Rappels

2.4.1 Définitions

Le but de cette section est de rappeler des définitions qui nous seront utiles pour la suite, ainsi que des notions sur la programmation en nombres entiers.

Pour plus de détail, le lecteur peut se referer à [13] et [16].

On désignera par:

A une matrice $m \times n$**.**

a **un vecteur non nul de** R^n**.**

b **un vecteur de** R^m**.**

x **un vecteur de** R^n**.**

β **un scalaire.**

Définition 2 *L'ensemble des points* $H = \{x \mid ax = \beta\}$ *est appelé* **Hyperplan**.

L'hyperplan divise R^n **en deux régions. Chaque région est de la forme** $\{x \mid ax \leq \beta\}$**.**

Définition 3 *L'ensemble des points* $\{x \mid ax \leq \beta\}$ *est appelé* **demi-espace**.

Définition 4 *L'ensemble des points* $P = \{x \mid Ax \leq b\}$ *est appelé* **Polyèdre**.

L'ensemble polyèdre est l'intersection d'un nombre fini de demi-espace.

Exemple 3 *Soit* $P = \{x + y \leq 5, -x - y \leq -5, -3x + 2y \leq 20, -x + 2y \leq 16\}$

Le polyèdre P *étant la demi droite d'origine le point* $(-2, 7)$ *et qui est représentée en gras sur la figure* 2.1.

Définition 5 L'enveloppe affine *de P noté E est le plus petit sous espace affine de R^n contenant P.*

Remarque

L'ensemble des égalités implicites définies l'enveloppe affine.

Définition 6 *La dimension de P est la dimension de son enveloppe affine.*

Exemple 4 *On reprend l'exemple précèdent.*

Le plus petit sous espace de R^2 contenant le polyèdre P est la droite d'équation $x + y = 5$. Par conséquent l'enveloppe affine de P est la droite d'équation $x + y = 5$. (on peut aussi remarquer qu'en faisant la somme des deux premières contraintes, on obtient $0 \leq 0$ et donc l'enveloppe affine de P est défini par $x + y \leq 5$, $-x - y \leq -5$ qui se réduit à $x + y = 5$).

Définition 7 *Soit H un hyperplan tel que $H \cap P$ se compose seulement de points frontières*[2] *de P alors: H est appelé* **Hyperplan d'appui** *de P.*

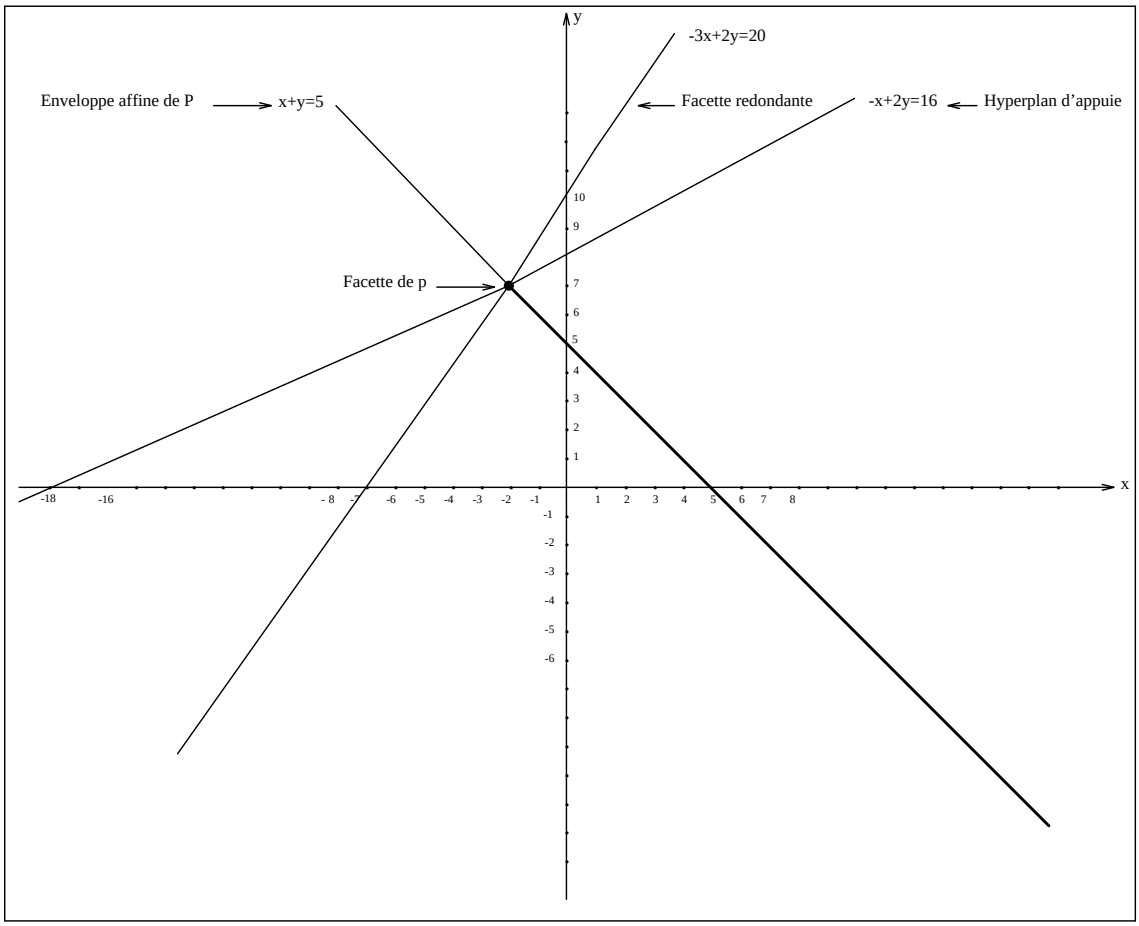

Figure 2.1 : Exemple de polyèdre

Exemple 5 *On reprend l'exemple précèdent.*

L'hyperplan H d'équation $-x + 2y = 16$ intersecte le polyèdre P au point $(-2, 7)$ qui représente une frontière de P. Par conséquent H: $-x+2y = 16$ est un hyperplan d'appuie.

Définition 8 *si H est un hyperplan d'appuie et $dim(H \cap P) = dim(E) - 1$ alors $H \cap P$ est appelé* **facette** *de P.*

[2]Frontière au sens topologique.

Exemple 6 *On reprend l'exemple précèdent.*

L'hyperplan d'appuie H: $-x + 2y = 16$ *intersecte le polyèdre P au point* $(-2, 7)$ *qui est de dimension 0. Or la dimension de l'enveloppe affine de P est 1. Donc la relation* $dim(H \cap P) = dim(E) - 1$ *est bien vérifiée. Par conséquent le point* $(-2, 7)$ *représente une facette de P, c'est l'unique facette de P.*

Définition 9 *On appelle* **Hyperplan Correspondant** *à la contrainte* $a_i x \leq \beta_i$*, l'ensemble des points définit par la contrainte* $a_i x = \beta_i$*.*

Définition 10 *La contrainte* $a_i x \leq \beta_i$ *dans P est dite* **redondante** *si P ne change pas quand* $a_i x \leq \beta_i$ *est supprimé, c'est-à-dire,* $\{x \mid Ax \leq b\} = \{x \mid a_j x \leq \beta_j, j \neq i\}$*.*

La détérmination des inégalités redondantes est aussi un problème classique et peut être résolu par le programme linéaire $max\{a_i x | a_j x \leq \beta_j, j \neq i\}$ *qui indique que* $a_i x \leq \beta i$ *est redondante si et seulement si la valeur optimale retournée est inférieure ou égale à* β_i*.*

Exemple 7 *On reprend l'exemple précèdent.*

La contrainte $-x + 2y \leq 16$ *est redondante*[3]*, en effet:*

$$\begin{aligned} P &= \{x + y \leq 5, -x - y \leq -5, -3x + 2y \leq 20, -x + 2y \leq 16\} \\ &= \{x + y \leq 5, -x - y \leq -5, -3x + 2y \leq 20\} \end{aligned}$$

Définition 11 L'enveloppe convexe *d'un ensemble P de* R^n*, notée conv P est le plus petit ensemble convexe contenant P.*

2.4.2 Programmation en nombres entiers

Ce rappel concerne la résolution des programmes en *nombres entiers*, **c'est-à-dire des problèmes d'optimisation dans lesquels les variables sont astreintes à ne prendre que des valeurs entières (ou certaines valeurs entières). Il s'agit là d'un des domaines les plus riches et les plus actifs de la programmation mathématique, et le volume de publications et de recherches qui lui ont été consacrées depuis les premiers travaux de** *Gomory* **(vers 1958) atteste la difficulté du sujet et l'importance des applications.**

[3]Le type de la redondance sera précisé plus loins.

Considérons le programme linéaire:

$$\textbf{(PL)} \quad \begin{cases} Minimiser\ z = c.x. \\ Sous\ les\ contraintes : \\ A.x = b \\ x \geq 0 \end{cases}$$

Tous les coefficients $c_j\ (j = 1, \ldots, n)$, $a_{ij}\ (i = 1, \ldots, m;\ j = 1, \ldots, n)$ **et** $b_i (i = 1, \ldots, m)$ **sont supposés** *entiers*.

D'autre part, pour simplifier la présentation, nous ferons l'hypothèse que le polyèdre:

$P = \{x \mid x \in R^n, Ax = b, x \geq 0\}$
est *borné* **et non vide.**

La première idée qui vient à l'esprit, lorsqu'on se trouve confronté à un problème en nombres entiers, est d'utiliser une *méthode d'arrondi*, **par exemple en remplaçant, dans la solution optimale continue, chaque composante fractionnaire par l'entier le plus proche.**

Malheureusement, dans beaucoup d'exemple, on a montré clairement l'insuffisance de telles méthodes ce qui a permis de mieux saisir la difficulté inhérente aux problèmes de programmation en nombres entiers.

Les deux principales familles de méthodes actuellement connues pour résoudre les programmes linéaires en nombres entiers sont: les méthodes de *recherche arborescente* **(ou d'énumération partielle) et les méthodes de** *coupes* **(ou de troncatures). Pour des raisons de coût-efficacité, on préfère utiliser la deuxième méthode.**

• Principe des méthodes de coupes

L'idée de base sur laquelle se fondent ces méthodes est la suivante.
On commence par résoudre le programme linéaire en continu (PL). Si la solution optimale est un point extrême à coordonnées entières c'est terminé.

Dans le cas contraire, il est facile de voir que l'on peut toujours tronquer le domaine des solutions (en rajoutant une contrainte supplémentaire au problème) de façon à éliminer ce point extrême sans exclure aucune solution entière. Une telle contrainte est appelée une *coupe* **(on dit encore: une troncature).**

Exemple 8 *On considére:*

$$P = \{x \in R^2_+ \mid Ax \geq b\}$$

$$A = \begin{pmatrix} 3 & -4 \\ 0 & -2 \\ -6 & -4 \\ -4 & 3 \end{pmatrix}, \qquad b = \begin{pmatrix} -7 \\ -5 \\ -25 \\ -11 \end{pmatrix}$$

La droite représentée sur la figure 2.2 représente une coupe.

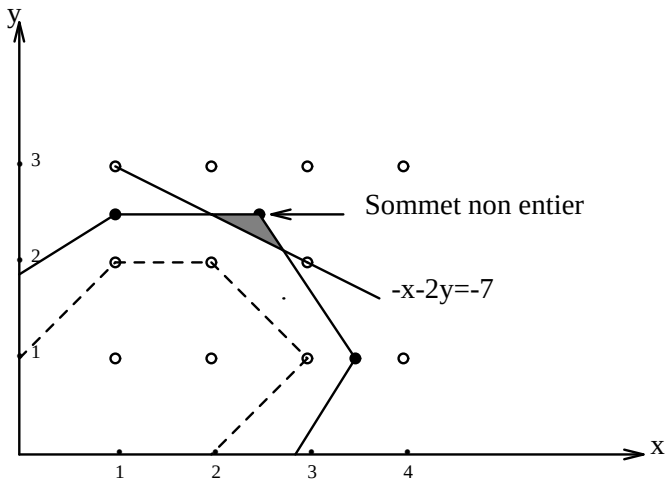

Figure 2.2 : Exemple de coupe

Après avoir rajouté une coupe (ou éventuellement plusieurs) le programme linéaire augmenté des contraintes correspondantes est à nouveau résolu (en continu) par la méthode du simplexe [19].

Si la solution optimale de ce nouveau problème est entière, c'est términé: on a obtenu une solution optimale du problème en nombres entiers. Sinon, le raisonnement précédent peut être répété: on recherchera une nouvelle coupe (éventuellement plusieurs) que l'on rajoutera à l'ensemble des contraintes; puis le programme linéaire ainsi augmenté sera optimisé à nouveau, etc.

Si les coupes sont correctement choisies à chaque étape, le polyèdre initial P sera ainsi progressivement réduit jusqu'à coïncider avec l'enveloppe convexe des solutions entières, au moins au voisinage de la solution optimale. La solution continue du problème augmenté deviendra alors entière et le problème sera résolu.

Il est clair, cependant, que le choix des coupes est détérminant pour la convergence de la méthode. Si par hasard, on choisi comme coupe les con-

traintes qui définissent l'enveloppe convexe des solutions entières du problème, on obtient directement l'optimum entier comme solution continue du programme linéaire augmenté avec ces coupes. Malheureusement et c'est là la difficulté essentielle, on ne connaît pas de méthode systématique pour engendrer toutes les équations ou inéquations définissant l'enveloppe convexe des points entiers contenus dans un polyèdre convexe donné. Elles peuvent d'ailleurs être en nombres énorme. Par conséquent engendrer toutes les faces de l'enveloppe convexe des points entiers serait coûteux et superflu: pour la plus part ces contraintes ne seraient pas actives et ne contribueraient en rien à définir l'optimum entier.

C'est pourquoi, un des résultats les plus importants en programmation en nombres entiers a été la mise en évidence (*Gomory* 1958) de coupes d'un type particulier permettant, moyennant certaines précautions, d'obtenir la convergence de la méthode.

•Coupes de Gomory

Considérons le programme linéaire continu:

$$\textbf{(PL)} \quad \begin{cases} Minimiser\ z = c.x. \\ Sous\ les\ contraintes: \\ A.x = b \\ x \geq 0 \end{cases}$$

avec B matrice de base optimale (sous-matrice carrée régulière extraite de A).

Puisque les coefficients de A sont entiers, $D = |det(B)|$ est entier. D'autre part, en prémultipliant le système$A.x = b$ par B^{-1}, toute variable de base x_i ($i \in I$=ensemble des indices des variables de base) peut s'exprimer en fonction des variables hors base $x_j (j \in J =$ ensemble des indices des variables hors base) par:

$$\textbf{(1)} \qquad x_i = \frac{\beta_i}{D} - \sum_{j \in J} \frac{\alpha_{ij}}{D}.x_j$$

où les coefficients β_i et α_{ij} sont entiers.

On rappelle que la solution de base correspondant à la base B est:

$$x_j = 0 \ (\forall j \in J) \text{ et } x_i = \frac{\beta_i}{D} \ (\forall i \in I).$$

Puisque l'on se place dans le cas où la solution optimale n'est pas entière, une des variables de base, x_i par exemple, est fractionnaire.

Exprimons le fait que l'on recherche une solution dans laquelle la variable x_i est entière: Cette condition fournit avec l'équation (1), l'équation de congruence:

(2) $\sum_{j\in J} \alpha_{ij}x_j \equiv \beta_i \pmod{D}$.

On remarque que l'on obtient une équation de congruence équivalente en multipliant les deux membres de (2) par un même nombre entier λ, premier avec D:

(3) $\sum_{j\in J}(\lambda\alpha_{ij})x_j \equiv (\lambda\beta_i) \pmod{D}$.

Pour un entier relatif quelconque y, notons $|y|_D$ le représentant de y dans [0,D-1] modulo D.

En posant alors: $f_j = |\lambda\alpha_{ij}|_D$ (pour $j \in J$) et : $f_0 = |\lambda\beta_i|_D$, on déduit de (3) qu'il existe un entier s tel que

$$\sum_{j\in J} f_j x_j = f_0 + sD.$$

Si s est négatif, $f_0 + sD$ est nécessairement négatif, ce qui contredit le fait que $f_j \geq 0$ et $x_j \geq 0$. Donc s est un entier positif ou nul, et l'inéquation:

(4) $\sum_{j\in J} f_j x_j \geq f_0$.

est nécessairement vérifiée par toute solution dans laquelle la variable x_i est entière.

Par ailleurs, on observe que cette inéquation n'est pas vérifiée par la solution de base courante puisque celle ci est définie par:

$$x_j = 0 \qquad \forall j \in J.$$

L'inéquation (4) constitue donc bien une *coupe*.

•Méthode de congruences décroissantes

Plutôt que de réoptimiser complétement le problème chaque fois que l'on rajoute une coupe, une autre idée consiste à effectuer une seule itération de l'algorithme dual du simplexe.

Supposons que l'on rajoute une coupe de la forme (4):

$$\sum_{j\in J} \frac{f_j}{D}.x_j \geq \frac{f_0}{D}.$$

dérivée de l'équation:

(1) $$x_i = \frac{\beta_i}{D} - \sum_{j\in J} \frac{\alpha_{ij}}{D}.x_j$$

La première itération de l'algorithme dual du simplexe conduit à pivoter sur un élément $f_j 0/D \neq 0$ ($j_0 \in J$ est l'indice de la variable qui rentre en base).

Le détérminant D' de la nouvelle matrice de base est alors:

$$D' = D \times \frac{f_{j0}}{D} = f_{j0} \leq D - 1$$

Si après avoir effectué ce pivot, la solution obtenue n'est toujours pas entière, en écrivant la condition d'intégrité d'une variable de base on obtiendra alors une équation de congruence du même type que (2) mais dans un groupe d'ordre D' strictement inférieur à D.

On voit donc qu'en répétant l'opération qui consiste à :

- **ajouter une coupe traduisant l'intégrité d'une variable de base,**
- **effectuer une étape de l'algorithme dual,**

on arrive nécessairement, en un nombre fini d'étapes, à une solution *entière, duale réalisable.*

C'est la méthode dite des congruences décroissantes; elle permet donc à partir de toute solution non entière duale réalisable, d'obtenir en un nombre fini d'étapes une solution entière duale réalisable. De plus, la méthode est très efficace car on constate expérimentalement que le nombre d'étapes nécessaire

est en moyenne de l'ordre de $log_2 D$ lorsqu'on choisit dans (3) le coefficient λ donnant à f_0 sa valeur maximale.

L'algorithme de coupes par la méthode des congruences décroissantes est le suivant:

Algorithme de coupes par la méthode des congruences décroissantes

Etape 1 Résolution du problème (PL) en continu par l'algorithme du simplexe. Si la solution obtenue est entière, FIN. Sinon:
Etape 2 (Réduction du problème).
Rechercher (par un procédé heuristique) une bonne solution entière. Si l'on n'en trouve pas, aller à l'étape 3. Sinon soit ẑ la valeur de cette solution. La fonction objectif s'écrit sous forme canonique relativement à la base optimale:
$z = z_0 + \sum_{j \in J} c_j x_j$
(où z_0 est la valeur de l'optimum continu et J l'ensemble des indices des variables hors base). On a donc $\forall j \in J : c_j \geq 0$.
Alors toute variable x_j telle que:
$c_j > \hat{z} - z_0$
doit être nulle dans une solution optimale et peut, par conséquent, être éliminée du problème.
Etape 3 Soit $J' = \{j \in J \mid c_j = 0\}$.
Si la valeur z_0 est entière et si $J' \neq \emptyset$, aller à l'étape 5. Sinon rajouter des coupes par la méthode des congruences décroissantes jusqu'a ce que le problème augmenté ait une solution entière (duale réalisable).
Si cette solution est aussi primale réalisable(c'est-à-dire si $x_j \geq 0$ pour toutes les variables j) alors c'est une solution optimale du problème. FIN.
Sinon:
Etape 4 Résolution du problème augmenté en *continu* par l'algotithme dual du simplexe.
Si l'optimum obtenu est entier, c'est une solution optimale du problème
FIN.
Sinon, aller à l'étape 3 (après avoir éventuellement éliminé un certain nombre de contraintes de coupes non actives).
Etape 5 Rechercher par énumération (implicite) une solution entière de $Ax = b$ avec $x_j = 0$ pour $j \in J - J'$.
S'il en existe une, c'est une solution optimale du problème. FIN.
Sinon on rajoute la coupe:
$\sum_{j \in J - J'} x_j \geq 1$.
Effectuer une première itération de l'algorithme dual en pivotant sur cette contrainte et retourner à l'étape 3.

Chapitre 3

Simplification des Contraintes Arithmétiques linéaires

Ce chapitre est une synthèse de [13].

Le problème de la redondance des contraintes arithmétiques linéaires est un des problèmes de la recherche opérationnelle et de l'intéligence artificielle. Sa particularité importante est dans l'implémentation des langages de programmation logique. Dans ce chapitre les contraintes redondantes sont classées selon leur propriéte géometrique, le coût de détection et le potentiel pour l'élimination en parallèle. On introduit une transformation pour simplifier des classes de contraintes redondantes en d'autres classes de contraintes redondantes qui sont simples à détécter et ensuite à éliminer. On utilise la méthode de Fourier pour l'élimination de variables comme un test pour l'importance de cette approche.

3.1 Introduction

Ainsi qu'on l'a mentionné dans le chapitre précèdent, l'algorithme de Fourier se caractérise par la taille potentiellement très importante de son résultat. Ceci se traduit par un grand nombre de contraintes redondantes. Le problème est illustré par la table suivante empruntée à [13]:

Nombre de variable éliminé	Nombre de contrainte générée	Nombre de contraintes équivalentes
0	32	18
1	226	40
2	12.744	50
3	39.730.028	19
4	390.417.582.083	2

La colonne du milieu nous montre la taille de la sortie quand la méthode de Fourier est utilisée pour éliminer entre 1 et 4 variables d'un ensemble initial de 32 contraintes. Alors que la colonne de droite donne la taille minimum d'une sortie équivalente.

3.2 Classes de redondances

Dans ce paragraphe on définit des classes de redondances. Ces classes couvrent tous les cas de redondances d'une contrainte dans un ensemble.

3.2.1 Tautologie

Définition 12 *Une* **tautologie** *est une contrainte qui a la forme* $0 \leq \beta$*, où* β *est un scalaire non négatif.*

Exemple 9 $0 \leq 1$ *est une tautologie*

Les tautologies sont les cas les plus triviaux de redondances. L'indépendance entre les contraintes redondantes, permet de les détecter et les éliminer en parallèle.

3.2.2 Syntaxe redondance

Définition 13 *La contrainte* $a_i x \leq \beta_i$ *est* **syntaxiquement redondante** *dans le système* $Ax \leq b$ *si pour un* $j \neq i$ *les contraintes* $a_i x \leq \beta_i$ *et* $a_j x \leq \beta_j$ *définissent le même demi-espace. Algébriquement* $a_i x \leq \beta_i$ *est une multiple scalaire positive de* $a_j x \leq \beta_j$*.*

Exemple 10 *Soit* $P = \{x + y \leq 1, 2x + 2y \leq 2\}$*.*

Les deux contraintes dans P sont syntaxiquement redondantes .

La redondance syntaxique est déterminée facilement par le test: Une contrainte est une multiple scalaire positive d'une autre.
Elle peut être éliminée par un degré de parallélisme. Parmi un ensemble de contraintes redondantes (syntaxiques entre elles), une doit être arbitrairement retenue.

3.2.3 Quasi-syntaxe redondance

Définition 14 *La contrainte $a_i x \leq \beta_i$ est* **Quasi-syntaxiquement redondante** *dans le système $Ax \leq b$ s'il existe une contrainte $a_j x \leq \beta_j$ où $j \neq i$ telle que:*
$a_j x = \beta_j$ est parallèle à $a_i x = \beta_i$. Algébriquement il existe un scalaire positif γ tel que: $a_j x = \gamma a_i x$ et $\beta_j < \gamma \beta_i$.

Remarque:

Cette définition peut être aussi formulée comme suit:

Une contrainte $c \in C$ est dite quasi syntaxiquement redondante si pour un $c' \in C$ et un ε, $c = c' + (0 \leq \varepsilon)$.

Exemple 11 *Soit $P = \{x + y \leq 1, x + y \leq 2, 2x + 2y \leq 6\}$. (voir figure 3.1)*
Les deux droites $x + y = 1$ et $2x + 2y = 6$ sont parallèles. Par conséquent la contrainte $2x + 2y \leq 6$ est Quasi-syntaxiquement redondante. De même la contrainte $x + y \leq 2$ est Quasi-syntaxiquement redondante.

En pratique, la redondance Quasi-syntaxique est détérminée avec le même critère que la redondance syntaxique. D'un autre coté, toutes les contraintes redondantes Quasi-syntaxique peuvent être éliminées simultanément.

3.2.4 Hull redondance

Définition 15 *La contrainte $a_i x \leq \beta_i$ est* **Hull redondante** *dans le système $Ax \leq b$ si son hyperplan correspondant est parallèle à l'envellope affine de l'ensemble polyèdre.*

Exemple 12 *Soit $P = \{-x + y - z \leq 0, x - y + 2z \leq 0, -x + y - 3z \leq 0, x - y \leq 1\}$. (voir figure 3.2)*

- *L'enveloppe affine de P est $E = \{x = y, z = 0\}$, en effet:*

On note respectivement par c_1, c_2, c_3, c_4, les 4 contraintes de P. La combinaison $c_1 + 2c_2 + c_3$ donne $0 \leq 0$. Donc c_1, c_2, c_3 sont des égalités implicites, par conséquent E est donné par le système $P = \{-x + y - z = 0, x - y + 2z = 0, -x + y - 3z = 0\}$, qui admet pour solution $x = y, z = 0$.

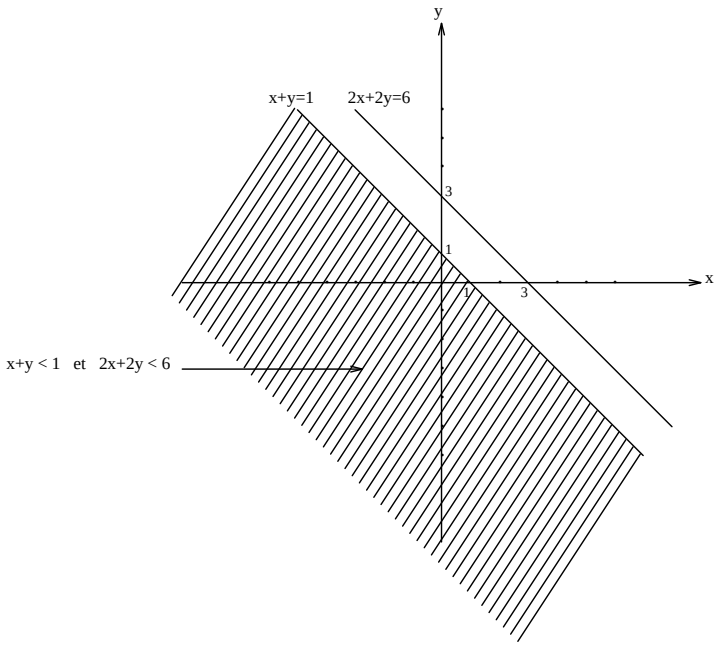

Figure 3.1 : Redondance quasi-syntaxique

- *Le plan $x - y = 1$ est parallèle à E.*

Par conséquent la dernière contrainte est hull redondante dans P.

Géométriquement, la hull redondance ne contribue pas à la définition des facettes. La détection des contraintes de cette classe nécessite la connaîssance de l'enveloppe affine de P. L'indépendance entre les contraintes redondantes, permet de les détecter et les éliminer en parallèle.

3.2.5 Facette redondance

Définition 16 *La contrainte $a_i x \leq \beta_i$ est* **facette redondante** *dans le système $Ax \leq b$ s'il existe une autre contrainte $a_j x \leq \beta_j$ où $j \neq i$ tel que les hyperplans $a_i x = \beta_i$ et $a_j x = \beta_j$ détérminent la même facette de l'ensemble polyèdre P.*

Exemple 13 *Soit $P = \{x + y \leq 5, -x - y \leq -5, -3x + 2y \leq 20, -x + 2y \leq 16\}$.*
(voir figure 3.3)

- *La seule facette de P est le point $\{x = -2, y = 7\}$, en effet:*

L'hyperplan d'appuie H: $-x + 2y = 16$ intersecte le polyèdre P au point $(-2, 7)$ qui est de dimension 0. Or la dimension de l'enveloppe affine $E = \{x + y = 5\}$ est 1. Donc la relation $dim(H \cap P) = dim(E) - 1$ est bien vérifiée. Par conséquent le point $(-2, 7)$ représente une facette de P.

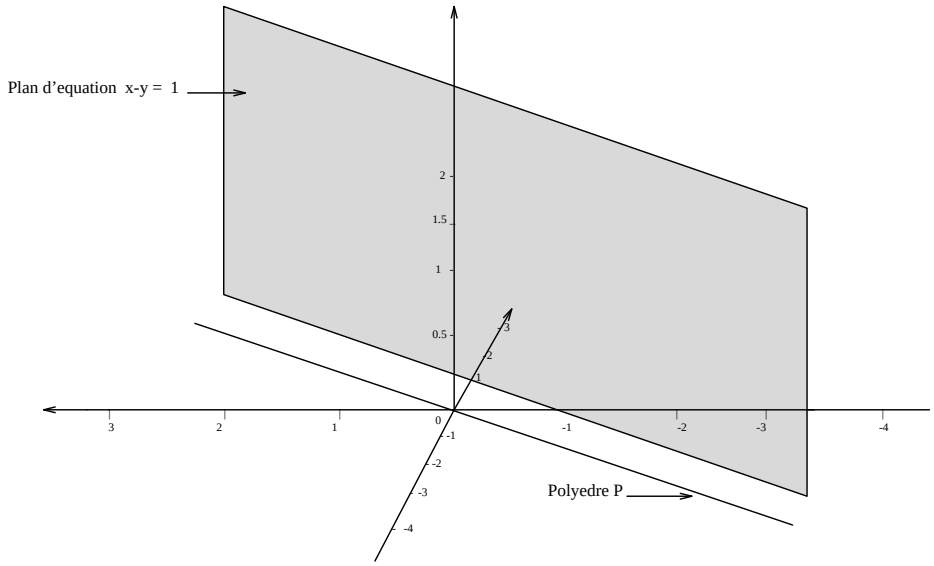

Figure 3.2 : Hull redondance

• *La droite d'équation* $-3x + 2y = 20$ *intersecte la droite d'équation* $-x + 2y = 16$ *au point* $(-2, 7)$ *qui est une facette de P.*

Par conséquent la contrainte $-3x + 2y \leq 20$ *est facette redondante dans P. De même la contrainte* $-x + 2y \leq 16$ *est facette redondante.*

La detérmination de la facette redondance nécessite des techniques de programmation linéaire. Géométriquement, on voit que de telles contraintes ne peuvent être éliminées en parallèle.

3.2.6 Quasi-facette redondance

Définition 17 *La contrainte* $a_i x \leq \beta_i$ *est* **quasi-facette redondante** *dans le système* $Ax \leq b$ *si l'intersection entre son hyperplan correspondant et l'enveloppe affine de l'ensemble polyèdre est parallèle à une facette de l'ensemble polyèdre.*

Exemple 14 *Soit* $P = \{x + y \leq 5, -x - y \leq -5, -x + 2y \leq 16, -x + y \leq 15\}$.
(voir figure 3.4*)*

• *La facette de P est le point* $(-2, 7)$ *(même raisonnement que l'exemple précèdent)*
• *L'intersection entre* $-x + y = 15$ *et l'enveloppe affine de P,* $E = \{x + y = 5\}$ *est le point* $(-5, 10)$ *qui est parallèle à la facette de P.*
Par conséquent la contrainte $-x + y \leq 15$ *est Quasi-facette redondante dans P.*

L'identification de cette classe de redondance, nécessite la connaîssance de l'enveloppe affine de P et de ses facettes. L'indépendance entre les contraintes

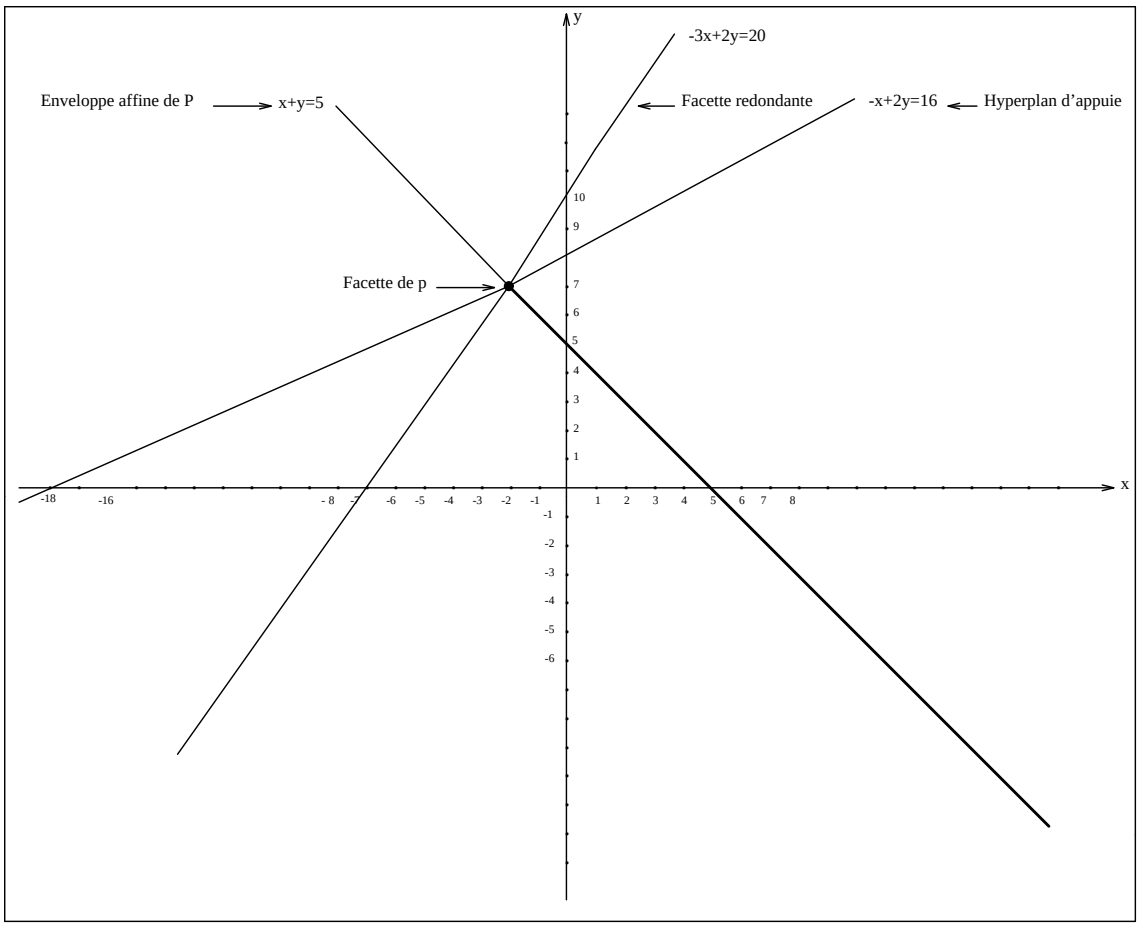

Figure 3.3 : Facette redondance

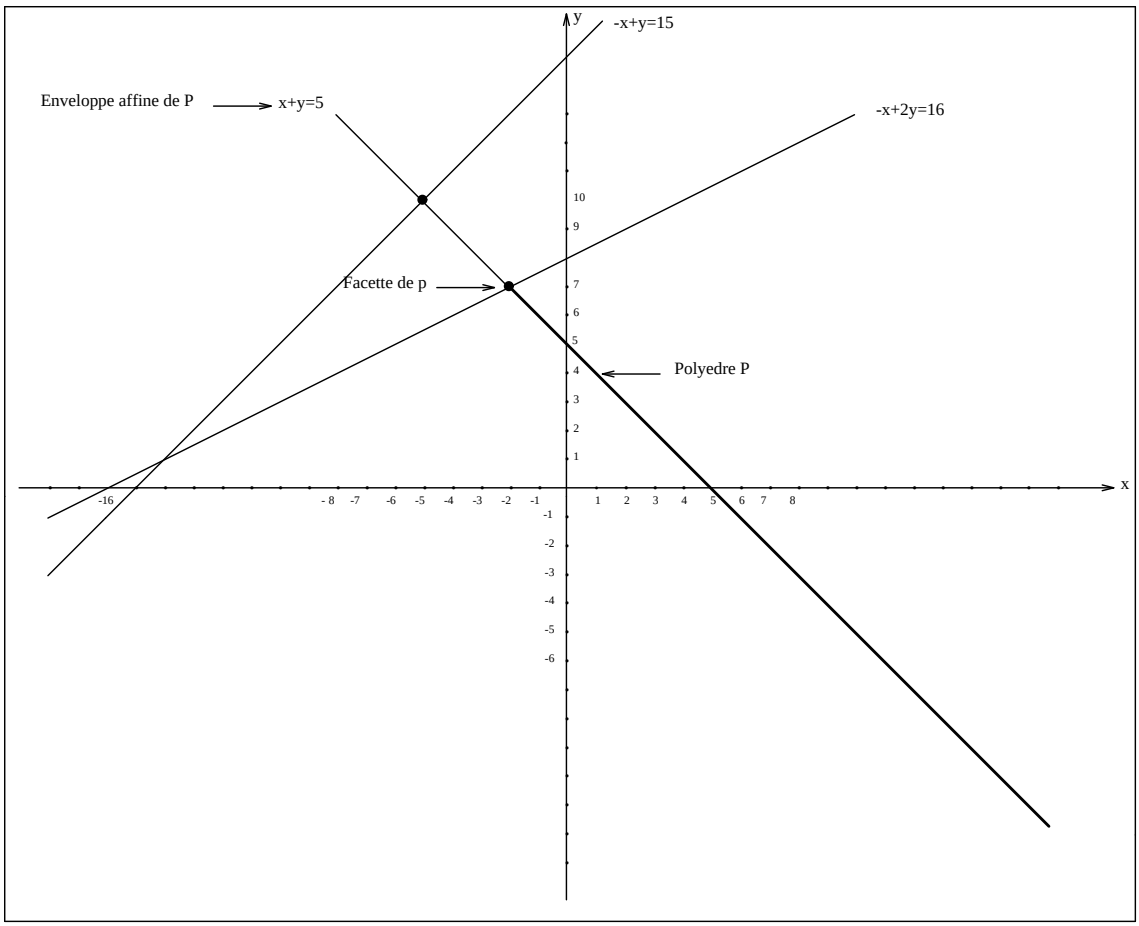

Figure 3.4 : Quasi-facette redondance

redondantes, permet de les détecter et les éliminer en parallèle.

3.2.7 Indépendante redondance

Définition 18 *la contrainte $a_i x \leq \beta_i$ est* **indépendante redondante** *dans le système $Ax \leq b$ si son hyperplan correspondant intersecte l'enveloppe affine de l'ensemble polyèdre mais l'intersection ne contient aucune facette de ce polyèdre.*

Exemple 15 *Soit $P = \{-x + y \leq 0, x + y \leq 4, -x + 2y \leq 2, y \leq 3\}$.*
(voir figure 3.5)

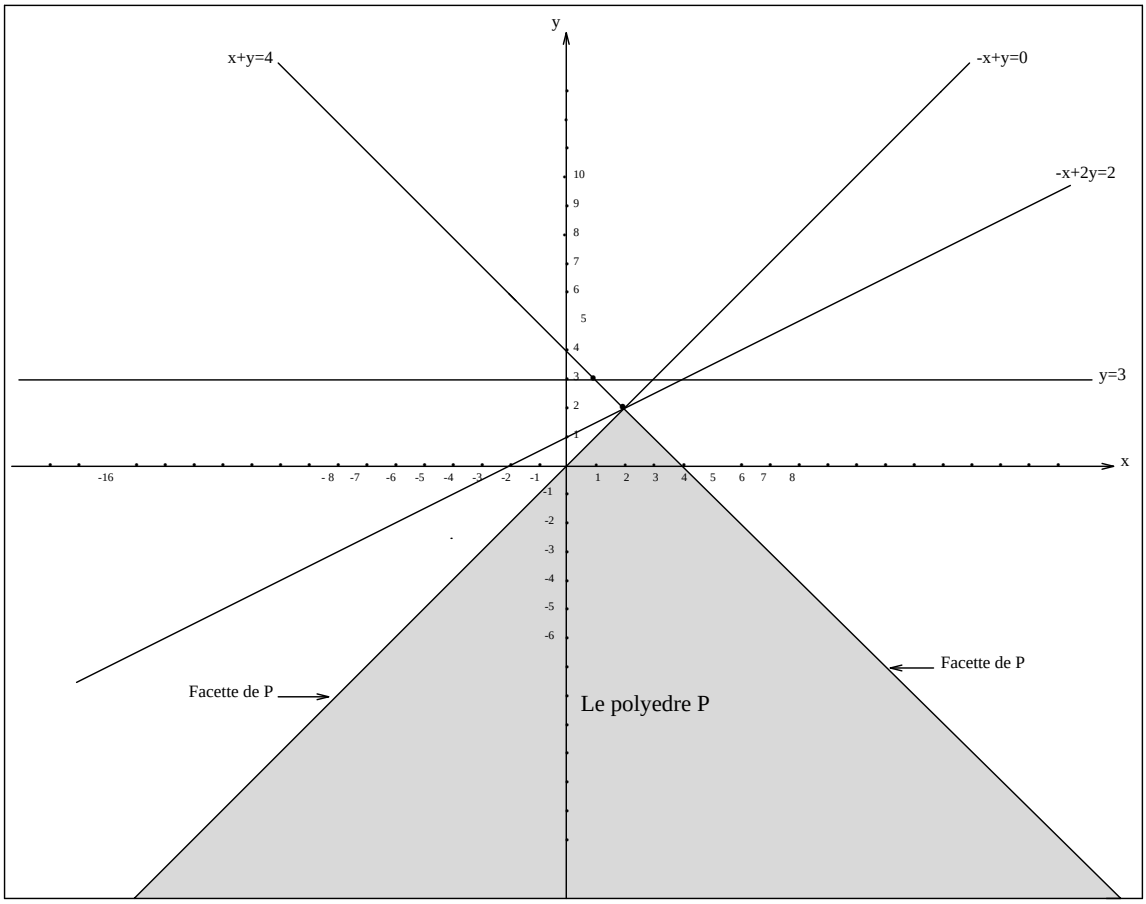

Figure 3.5 : Indépendante redondance

• *l'enveloppe affine de P étant le plan (car c'est le plus petit espace contenant le polyèdre P).*

• *L'intersection entre l'enveloppe affine de P et la droite d'équation $-x + 2y = 2$ est cette droite elle même.*

• *Les facettes de P sont les deux demi droite d'origine* $(2, 2)$ *représentées sur la figure ci-dessus.*

Donc l'hyperplan $-x + 2y = 2$ *intersecte l'enveloppe affine de P et cette intersection ne contient aucune facette de P. Par conséquent la contrainte* $-x+2y \leq 2$ *est redondante indépandante. De même la contrainte* $y \leq 3$ *est redondante indépandante.*

L'identification de cette classe de redondance, nécessite des techniques de programmation linéaire.

3.2.8 Implicite redondance

Considérons $P = \{x + y + z \leq 4, -x - y + z \leq -4, -x - z \leq 0, x - z \leq 0\}$**.**

Aucune des contraintes n'est redondante. Cependant chaque contrainte est une égalité implicite, en effet la combinaison $c_1 + c_2 + c_3 + c_4$ **des 4 contraintes de P, donne** $0 \leq 0$**. Par conséquent on a :** $P = \{x+y+z = 4, -x-y+z = -4, -x-z = 0, x - z = 0\}$ **qui est tout simplement réduit à l'ensemble** $\{x = 0, y = 4, z = 0\}$**. Donc, dans cet exemple on voit qu'il y a une certaine notion de redondance qui apparaît seulement lorsque les inégalités sont remplacées par des égalités, en effet les 3 contraintes** $x + y + z = 4, -x - y + z = -4, -x - z = 0$**, par exemple, donne le même ensemble** $\{x = 0, y = 4, z = 0\}$**. Autrement dit** $x - z = 0$ **est une contrainte redondante dans P.**

Définition 19 *On dit qu'une contrainte non redondante* $a_i x \leq \beta_i$ *est* **implicite redondante** *si et seulement si* $a_i x = \beta_i$ *est une contrainte redondante.*

Comment peut on caractériser les contraintes qui ont cette forme implicite de redondance ? Considérons le système $\{Ex = c, Ax \leq b\}$ **où** $Ex = c$ **est l'ensemble des égalités. Dans [13] Jean-Louis lassez et Ken McAloon ont prouvé le théorème suivant et son corollaire:**

Théorème 4 *Si on remplace toutes les égalités implicites* $a_i x \leq \beta_i$ *par* $a_i x = \beta_i$ *dans le système* $\{Ex = c, Ax \leq b\}$*, alors de telles égalités sont redondantes dans le nouveau système.*

Corollaire 1 *Une contrainte est redondante implicite si et seulement si c'est une égalité implicite.*

Remarque:

Une conséquence importante de ce corollaire est que pour éliminer les contraintes redondantes implicites, il est nécessaire de calculer l'enveloppe affine. C'est le but de la section suivante:

3.3 Simplification de la redondance via la transformation

Dans ce paragraphe, on définit la transformation appelée la Hull transformation qui transforme des classes de redondance en d'autres plus simples à détécter. Par exemple la hull redondance qui initialement demande des techniques de programmation linéaire pour être identifiée est transformée en tautologie, trivial à détécter et à éliminer. De même, la facette redondance est transformée en redondance syntaxique. Cette transforation réduit le coût d'identification et d'élimination des contraintes redondantes et assure que le reste des contraintes redondantes peuvent être éliminées en parallèle. Le coût de la transformation est lié au calcul des égalités implicites. Avant de définir la transformation on rappelle quelques définitions utiles pour la suite.

Définition 20 *L'ensemble $y = Ex$ d'équations linéaires est dit sous* **forme résolue** *si chaque égalité dans $y = Ex$ est dans la forme standard $y_i = a_i x + c_i$, où y_i est appelée variable liée et chaque élément x_j du vecteur x est appelée paramétre. Les vecteurs x et y n'ont pas d'élément commun.*

Définition 21 *Soit P l'ensemble polyèdre défini par $\{Ax \leq b, y = Ex\}$. Alors si $y = Ex$ défini l'enveloppe affine de P dans sa forme résolue, on appelle $Ax \leq b$,* **la représentation paramétrique de P**.

Comment obtenir la représentation paramétrique de P ?

On extrait d'abord toutes les égalités implicites de $Ax \leq b$, on calcule alors l'enveloppe affine utilisant ces égalités et on écrit la représentation de l'enveloppe affine dans sa forme résolue $y' = E'x'$. Finalement on élimine toutes les variables liées des contraintes restantes dans $Ax \leq b$. On obtient alors un nouveau système $A'x' \leq b'$ et x' une variable libre. Par conséquent, $A'x' \leq b'$ est la représentation paramétrique de P. Ce processus est appelé Hull transformation. On note que la représentation paramétrique de P n'est pas unique du fait qu'elle dépend des variables liées choisies.

Pour bien illustrer la hull transformation, voyons un exemple :

Soit P le polyèdre défini par les inégalités suivantes:

$$C_1 : \quad -x + y - z \leq 0 \qquad (implicite\ redondante)$$
$$C_2 : \quad x - y + 2z \leq 0 \qquad (implicite\ redondante)$$

$$
\begin{array}{lll}
C_3: & -x+y-3z \leq 0 & (implicite\ redondante) \\
C_4: & 2x+y-4z \leq 18 & (facette\ redondante) \\
C_5: & -x+2y+z \leq 6 & (facette\ redondante) \\
C_6: & -x+y-z \leq -4 & (quasi-facette\ redondante) \\
C_7: & x-y \leq 1 & (hull\ redondante) \\
C_8: & -x \leq -2 &
\end{array}
$$

En appliquant l'algorithme de Fourier, on voit bien que les trois inégalités c_1, c_2, c_3 sont des égalités implicites (la simple combinaison $c_1 + 2c_2 + c_3$ donne $0 \leq 0$). L'enveloppe affine de P est $E = \{c_1, c_2, c_3\} = \{y = x, z = 0\}$ et y et z sont des variables liées.

Eliminons y et z des contraintes restantes on obtient un nouvel ensemble d'inégalités qui est la représentation paramétrique de P :

$$
\begin{array}{lll}
C'_4: & 3x \leq 18 & (syntaxiquement-redondante) \\
C'_5: & x \leq 6 & (syntaxiquement-redondante) \\
C'_6: & -4x \leq -4 & (quasi-syntaxiquement\ redondante) \\
C'_7: & 0 \leq 1 & (tautologie) \\
C'_8: & -x \leq -2 &
\end{array}
$$

On peut alors constater que les contraintes facettes redondantes c_4 et c_5 sont transformées en contraintes redondantes syntaxiques c'_4 et c'_5, la contrainte quasi-facette redondante c_6 est transformée en contrainte quasi syntaxiquement redondante c'_6 et la contrainte hull redondante c_7 est transformée en tautologie c'_7. Après l'élimination de toutes les contraintes redondantes, on obtient l'ensemble final:

$$
\begin{array}{ll}
E_1: & y = x \\
E_2: & z = 0 \\
C_5: & x \leq 6 \\
C_8: & -x \leq -2
\end{array}
$$

Dans [13], Jean Louis Lassez et Ken McAloon ont montré le théorème suivant:

Théorème 5 *Soit T la hull transformation qui transforme les classes de redondances en classes de redondances, alors on a:*

(1) T : $hull\ redondance \rightarrow tautologie$

(2) T : $facette\ redondance \rightarrow syntaxique\ redondance$

(3) T : $quasi - facette\ redondance \rightarrow quasi - syntaxique\ redondance$

(4) T : $redondance\ implicite \rightarrow egalite\ redondante$

3.4 Algorithme

Dans ce paragraphe, on décrit un algorithme qui est basé sur la hull transformation pour la suppression des contraintes redondantes. Il utilise 3 procédures prédéfinies:

• La première est une boite noire pour la résolution de problèmes de programmation linéaire appelée min ou max qui accepte une fonction objective avec un ensemble de contraintes et qui retourne une valeur optimale si elle est fini ou retourne $\pm\infty$ si la valeur optimale est illimitée.

• La deuxième est appelé Gauss-Jordan qui prend un ensemble d'égalités comme paramétre et les retourne en forme de lignes échelonnées après l'élimination de Gauss-Jordan.

• La troixième appelée Eliminate Bound variables est auto explicative. (car c'est juste une substitution)

On suppose que les tautologies ont été suprimées après que toutes les variables liées aient été éliminées. Les redondances syntaxiques et quasi-syntaxiques sont éliminées dans la procédure Remove Syntactic Redundancy. Pour chaque contrainte $a_i x \leq \beta_i$ dans P, on associe une variable booléene appelée $redundant_i$ qui est utilisée pour indiquer si la contrainte est redondante.

L'algorithme est le suivant:

Procedure Parametric hull Representations $(\{Ax \leq b, y = Ex\})$;
$/*$ m **is the number of contraints in** $(\{Ax \leq b\})$ $*/$
For all $1 \leq i \leq m$ **do in parallel**
If (min $\{a_i x : Ax \leq b\} = \beta_i$)
Then $redundant_i = true$ $/*$ $a_i x \leq \beta_i$ **is an implicit equality** $*/$
Else $redundant_i = false$
End Sync;
$\{y' = E'x'\})$ **=Gauss Jordan** $(\{a_i x \leq \beta_i : redundant_i = true, y = Ex\})$;

$\{A'x \leq b'\}$= **Eliminate Bound variables** $(\{a_i x \leq \beta_i : redundant_i = false, y' = E'x'\})$;
Return $(\{A'x' \leq b', y' = E'x'\})$;
Procedure Remove syntaxic redandancy $(\{Ax \leq b\})$;
/* **m is the number of contraints in** $(\{Ax \leq b\}$ */
/*γ **is a positive scalar** */
For $i = 1$ **to** $m - 1$ **do**
If(not $redundant_i$**)**
Then Forall $i + 1 \leq j \leq m$ **do in parallel**
If(not $redundant_i$**) and** $(a_j x = \gamma a_i x)$ **and** $(\gamma > 0)$
Then If $(\beta_j < \gamma \beta_i)$
Then $redundant_i = true$ /* $\{a_i x \leq \beta_i$ *is* $redundant$* /
else $redundant_j = true$ /* $\{a_j x \leq \beta_j$ *is* $redundant$*/
End Sync;
End For;
Return $(\{a_i x \leq \beta_i : redundant_i = false\})$;
Procedure remove Independent redundancy $(\{Ax \leq b)$;
/* **m is the number of contraints in** $(\{Ax \leq b\}$ */
For all $1 \leq i \leq m$ **do in parallel**
If (max $\{a_i x : a_j x \leq \beta_j, j \neq i\} \leq \beta_i)$
Then $redundant_i = true$ /* $a_i x \leq \beta_i$ *is* $redundant$* /
Else $redundant_i = false$
End Sync;
Return $(\{a_i x \leq \beta_i : redundant_i = false\})$;
Procedure Remove Redundancy $(\{Ax \leq b, y = Ex\})$;
/* **Transform** $\{Ax \leq b\}$ **into a parametric representation** */
$(\{A_1 x' \leq b_1, y' = E'x'\})$ = **Parametric Hull Representations** $(\{Ax \leq b, y = Ex\})$;
/* **Remove syntactic and quasi syntactic redundancy** */
$\{A_2 x' \leq b_2\}$ = **Remove syntactic redundancy** $(\{A_1 x' \leq b_1\})$;
/* **Remove independant redundancy** */
$(\{A'x' \leq b'\})$ = **Remove Independent Redundancy** $(\{A_2 x' \leq b_2\})$;
Return $(\{A'x' \leq b', y' = E'x'\})$;

3.5 Application à l'élimination de variable

On applique l'algorithme de Fourier pour l'élimination des variables. Comme c'était déjà cité, l'algorithme de Fourier génère beaucoup de contraintes redondantes qui n'intéréssent pas l'utilisateur. Cette étude relève de l'implémentation de langage de programation logique par contrainte qui emploi les méthodes de Fourier.

Dans [13], Jean Louis lassez et Ken McAloon ont proposé l'algorithme suivant pour l'élimination de variables.

Le programme prend comme entrée $Ax \leq b$ qui définit le polyèdre P et retourne un nouvel ensemble de contraintes après l'élimination des variables spécifiées par la méthode de Fourier. Ce nouvel ensemble de contraintes définit le polyèdre P' qui est la projection de P. Chaque fois qu'une variable est éliminée, il y a une suppression de la redondance associée.

Transform the initial set of contraints into a parametric representation in parallel
Remove tautologies, syntaxic and quasi syntaxic redandancy
Remove independent redundancy in parallel
For each variable to be eliminated do
    Apply Fourier's method to eliminate the variable
    Remove tautologies, syntaxic and quasi syntaxic redandancy
    Remove independent redundancy in parallel
End for

Figure 3.6 : Elimination de contraintes redondantes

Une application de cet algorithme est résumé dans la table[1] suivante:
Le nombre de processeurs utilisés dans cet exemple est 624.

[1]Pour tout détail, le lecteur peut se reférer à [13].

Nombre de contraintes initiales	**8**	**16**	**32**
Nombre de variables éliminées	**2**	**3**	**4**
Nombre de contraintes potentiellement générées par l'algorithme de Fourier	**60**	**258.597**	**390.417.582**
Nombre de contraintes irredondantes	**2**	**2**	**2**
Supression séquentielle de redondance	**0,292s**	**17,085s**	**623,410s**
Supression parallèle de redondance	**0,047s**	**0,353s**	**4,567**

Figure 3.7 : Application de l'algorithme d'élimination de contraintes redondantes

Chapitre 4

Sorties dans CLP(R)

Ce chapitre est une synthèse de [15].

On considère les systèmes de contraintes de CLP(R) et on décrit les traits essentiels de son module sortie. On met le point sur le problème bien connu de projection dans les contraintes arithmétiques linéaires. On commence par l'algorithme classique de Fourier et on l'augmente par la procédure d'élimination de contraintes redondantes générées par cet algorithme. Le reste du chapitre traite des autres sortes de contraintes, équations sur les arbres et les équations non linéaires et montre comment elles peuvent être sorties avec les contraintes linéaires.

4.1 Introduction

Parmi les problèmes dans les systèmes de programmation logique par contrainte, celui de la simplification de contraintes semble très important. La sortie d'un programme de programmation logique par contrainte est la collection de toutes les contraintes accumulées le long des pas de succés. Cependant une telle collection est peu lisible car elle est très large et contient beaucoup de variables intérmidiaires qui n'intéressent pas l'utilisateur. Une simple formulation du problème est :

Etant donné un ensemble $\tilde{x}$ de variables cibles et une conjonction $C(\tilde{x}, \tilde{y})$ de contraintes, exprimer $\exists \tilde{y}\ C(\tilde{x}, \tilde{y})$ dans la forme la plus simple.
Exemples:

x et y désignent des variables cibles:

(a) les contraintes $x = f(z, z)$, $z = g(y, w)$ peuvent être simplifiées en $x = f(g(y, w), g(y, w))$.

(b) **les contraintes** $x = z+1$, $y = 2*z$ **peuvent être simplifiées en** $x = 0.5*y+1$.
(c) **les contraintes** $x < z$, $z \leq y$,$z \leq y+1$ **peuvent être simplifiées en** $x < y$.
(d) **la contraintes** $x = sinz * sinz + cosz * cosz + y$ **peuvent être simplifiées en** $x = 1 + y$.

On peut classer la simplification de ces contraintes en 3 catégories:

(I) L'élimination de variables auxilaires (comme dans (a),(b)**et** (c)**) ;**
(II) L'élimination de contraintes redondantes (comme dans(c)) ;
(III) Le remplacement des expressions par d'autres équivalentes et simples (comme d).

Le problème (I) de l'expression de $\exists \tilde{y}\ C(\tilde{x}, \tilde{y})$ **comme une formule contenant seulement** $\tilde{x}$ **est connu sous la forme d'un problème de projection, élimination de variables et élimination de quantification. L'élimination de toutes les variables n'est pas toujours possible dans CLP(R), par exemple dans** (a) **ci-dessus. Cependant on note qu'il est théoriquement possible d'éliminer toutes les variables auxiliaires d'un ensemble de contraintes arithmétiques pures. L'élimination de contraintes redondantes (II) est en général un problème très difficile, comme celui de la satisfiabilité de contraintes. La découverte des expressions équivalentes (III) est aussi en général un problème très difficile.**

L'approche traditionnelle pour la simplification de contraintes, est d'utiliser la notion de *forme canonique* **calculée par un algorithme efficace. Informellement, telle forme représente l'information contenant les contraintes originales dans une manière minimale par rapport aux variables cibles** $\tilde{x}$**. Pour les équations linéaires, il est bien connu que la forme canonique est appelée forme paramétrique où les équations sont représentées dans la forme** $\tilde{x} = t(\tilde{y})$ **où les** $\tilde{y}$ **sont distinctes des** $\tilde{x}$ **et** t **est un tuple d'expressions linéaires.**

4.2 Contraintes linéaires

Soit C un ensemble de contraintes linéaires.
Soit $\tilde{x} = x_1, x_2, \ldots, x_N$ **les variables cibles dans C.**
Soit $\tilde{y} = y_1, y_2, \ldots, y_M$ **les variables auxiliaires dans C.**
Dans ce paragraphe, on décrit un algorithme pour la sortie de $\exists\ \tilde{y}\ C(\tilde{x}, \tilde{y})$**, où C est linéaire, en fonction uniquement des variables cibles, traitant d'abord les équations et après les inéquations. L'algorithme peut produire une sortie ne contenant pas la variable cible particulière** x **qui apparaît dans C, par exemple lorsqu'on elimine** y **de** $\exists\ y\ x = y + 2$**, dans telle cas on rajoute la contrainte**

réel(x) qui signifie tout simplement que x est réel.

4.2.1 équations linéaires

Les équations sont maintenues sous une *forme paramétrique*, $\tilde{u} = t(\tilde{v})$ où les variables de $\tilde{u}$ sont appelées variables objets et sont distincts des variables de $\tilde{v}$, appelés, quant à elles, paramétres. Pour chaque variable objet z on note par $r.h.s(z)$ l'expression linéaire qui contient z.

L'algorithme a la forme d'élimination de Gauss, il est décrit dans la figure ci-dessous, il suppose qu'il y a une priorité entre les variables, $\pi(x_1) > \ldots > \pi(x_N) > \pi(y_1) > \ldots > \pi(y_M)$, exprimant l'importance relative de chaque variable dans la sortie[1]. L'algorithme assure que les variables de priorité supérieure sont représentées en fonction de variables de priorité inférieure.

Le point intéressant pour l'efficacité de l'algorithme est que le corps de la boucle est itéré N fois, et $N \ll M$ en général (le nombre de variables cibles est souvent très petit par rapport aux nombres de variables dans le système).

```
for (i = 1; i ≤ N; i = i + 1){
      if (x_i is a parameter) continue:
      let ε denote the equation x_i = r.h.s(x_i) at hand :
      if (r.h.s(x_i) contains a variable z of lower priority than x_i) {
            choose the z of lowest priority:
            rewrite the equation ε into the form z = t :
            if (z is a target variable) mark the equation ε as final:
            substitue t for z in the other linear équations and inegalities:
      } else mark the equation ε as final:
}
return all final équations
```

Figure 4.1 : Equations linéaires

4.2.2 Inégalités linéaires

Dans ce paragraphe, on traite le problème d'élimination des variables non cibles qui figurent dans les inégalités. On utilise la méthode basée sur l'algorithme de Fourier. Il est bien connu que l'application direct de cet algorithme est impraticable du fait qu'il génére beaucoup de contraintes redondantes (voir chapitre 2). L'essai pour éliminer toutes les redondances à chaque

[1]La priorite entre les y_i est arbitraire.

étape est aussi impraticable. Les adaptations de l'algorithme de Fourier dues à Cernicov augmentent considérablement la performance. On montre comment incorporer d'autres méthodes d'élimination de redondance avec celles de Cernicov pour obtenir un algorithme pratique pour l'élimination de variables des inégalités linéaires.

Méthode basée sur Fourier

Soit x_i une variable.
Soit c et c' deux contraintes.
Soit $c_j = \sum_{i=1}^{m} \alpha_{j,i} x_i \leq \beta_j$ une contrainte.
Soit C un ensemble de contraintes étiquetées.
On dit que c est subsumé par c' si étiquette(c') $\subseteq$ étiquette(c).

On divise C en trois sous ensemble:

- **$C^+_{x_i}$, contraintes dans lesquelles x_i a un coefficient positif.**
- **$C^-_{x_i}$, contraintes dans lesquelles x_i a un coefficient négatif.**
- **$C^0_{x_i}$, contraintes dans lesquelles le coefficient de x_i est nul.**

Soit $d = 1/\alpha_{k,i} * c_k - 1/\alpha_{l,i} * c_l$. Alors par construction x_i ne figure pas dans la contrainte d. Si S_k est l'étiquette de c_k et S_l l'étiquette de c_l, alors d a l'étiquette $S_k \cup S_l$. Soit D la collection de tous les d. Alors $D \cup C^0$ est le résultat d'un pas de l'élimination de Fourier. On écrit $fourier_i(C) = D \cup C^0$. Lorsque C^+ et C^- sont tous les deux non vides, alors D est non vide et le pas est appelé une élimination de variable active. Après l'élimination de x, le nombre de contraintes dans C augmente (possiblement diminue) par mesure $(x_i \ , \ c) = |C^+| \times |C^-| - |C^+| - |C^-|$.

Soit $\mathcal{A}$ un ensemble de contraintes étiquetées chacune par son étiquette. On définie $\mathcal{F}_0 = \mathcal{A}$ et $\mathcal{F}_{i+1} = fourier_{i+1}(\mathcal{F}_i)$. Alors $\{\mathcal{F}_i\}_{i=0,1,\ldots}$ est la séquence de l'ensemble des contraintes obtenus par la méthode de Fourier, éliminant dans l'ordre, $y_1, y_2, \ldots$. On obtient $\mathcal{F}_n \leftrightarrow \exists y_1, y_2, \ldots y_n \ \mathcal{A}$. Ainsi l'algorithme de Fourier calcule les projections.

Exemple 16 *On note par $\mathcal{A}$ l'ensemble de contraintes étiquetées suivants:*

$$\begin{array}{ll} \{1\} & w + x + y + z \leq 1 \\ \{2\} & w - x + y + z \leq 1 \\ \{3\} & -w + x + y + z \leq 1 \end{array}$$

$$\begin{array}{lrrl}
\{4\} & & -w-x+y+z & \leq 1 \\
\{5\} & v & -y & \leq 0 \\
\{6\} & -v & & \leq 0
\end{array}$$

pas 1 *On applique l'algorithme de Fourier pour éliminer v on obtient:*

$$\begin{array}{ll}
\{1\} & w+x+y+z \leq 1 \\
\{2\} & w-x+y+z \leq 1 \\
\{3\} & -w+x+y+z \leq 1 \\
\{4\} & -w-x+y+z \leq 1 \\
\{5,6\} & -y \leq 0
\end{array}$$

pas 2 *Ensuite on élimine w on obtient:*

$$\begin{array}{ll}
\{1,3\} & x+y+z \leq 1 \\
\{1,4\} & y+z \leq 1 \\
\{2,3\} & y+z \leq 1 \\
\{2,4\} & -x+y+z \leq 1 \\
\{5,6\} & -y \leq 0
\end{array}$$

pas 3 *On élimine maintenant x on obtient:*

$$\begin{array}{ll}
\{1,2,3,4\} & y+z \leq 1 \\
\{1,4\} & y+z \leq 1 \\
\{2,3\} & y+z \leq 1 \\
\{5,6\} & -y \leq 0
\end{array}$$

pas 4 *Dans le dernier pas de Fourier on élimine y on obtient:*

$$\begin{array}{ll}
\{1,2,3,4,5,6\} & z \leq 1 \\
\{1,4,5,6\} & z \leq 1 \\
\{2,3,5,6\} & z \leq 1
\end{array}$$

Donc dans le pas final de Fourier on obtient le système simplifié:

$$z \leq 1$$

Cependant l'algorithme de Fourier génère beaucoup de contraintes redondantes et a une complexité double exponentielle dans le pire des cas. Dans [6],Cernikov propose des modifications qui permettent la supression de quelques contraintes redondantes durant le pas de Fourier. La première méthode élimine toutes contraintes générées à la n^{ieme} étape qui a une étiquette de cardinalité supérieure ou égale à $n+2$. La seconde méthode retient à chaque étape l'ensemble S des contraintes tel que toute contrainte générée à cette étape est subsumée par une contrainte dans S. Ces méthodes sont correctes dans le sens suivant: Si $\{C_i\}_{i=0,1,\ldots}$ est la séquence générée, alors $C_i \leftrightarrow \exists y_1, \ldots y_i \; \mathcal{A}$, pour tout i.

Exemple 17 *On reprend l'exemple précèdent:*

$$\begin{array}{lrcl}
\{1\} & & w+x+y+z & \leq 1 \\
\{2\} & & w-x+y+z & \leq 1 \\
\{3\} & & -w+x+y+z & \leq 1 \\
\{4\} & & -w-x+y+z & \leq 1 \\
\{5\} & v & -y & \leq 0 \\
\{6\} & -v & & \leq 0
\end{array}$$

pas 1 *On applique l'algorithme de Fourier pour éliminer v on obtient:*

$$\begin{array}{lcl}
\{1\} & w+x+y+z & \leq 1 \\
\{2\} & w-x+y+z & \leq 1 \\
\{3\} & -w+x+y+z & \leq 1 \\
\{4\} & -w-x+y+z & \leq 1 \\
\{5,6\} & -y & \leq 0
\end{array}$$

Remarquons que le critère de Cernikov ne nous permet de suprimer aucune contrainte.
pas 2 *Ensuite on élimine w on obtient:*

$$\begin{array}{lcl}
\{1,3\} & x+y+z & \leq 1 \\
\{1,4\} & y+z & \leq 1 \\
\{2,3\} & y+z & \leq 1 \\
\{2,4\} & -x+y+z & \leq 1 \\
\{5,6\} & -y & \leq 0
\end{array}$$

Remarquons encore que le critère de Cernikov ne nous permet de suprimer aucune contrainte.

pas 3 *On élimine maintenant x on obtient:*

$$\begin{array}{lcl}
\{1,2,3,4\} & y+z & \leq 1 \\
\{1,4\} & y+z & \leq 1 \\
\{2,3\} & y+z & \leq 1 \\
\{5,6\} & -y & \leq 0
\end{array}$$

Une autre fois, remarquons que le critère de Cernikov ne nous permet de suprimer aucune contrainte.

pas 4 *Dans le dernier pas de Fourier on élimine y on obtient:*

$$\begin{array}{ll} \{1,2,3,4,5,6\} & z \leq 1 \\ \{1,4,5,6\} & z \leq 1 \\ \{2,3,5,6\} & z \leq 1 \end{array}$$

Cette fois le critère de Cernicov nous permet de supprimer la première contrainte (car son cardinale est 6 qui vérifie bien que $6 \geq n+2$*, ici n est le nombre de pas qui est égale à* 4 *) et alors on obtient:*

$$\begin{array}{ll} \{1,4,5,6\} & z \leq 1 \\ \{2,3,5,6\} & z \leq 1 \end{array}$$

Dans ce dernier pas, le critère de Cernikov ne nous permet de suprimer aucune des deux contraintes, en plus l'algorithme de Fourier est déjà términé et le système simplifié est:

$$z \leq 1$$

Remarquons que le critère de Cernikov nous a permis de supprimer la contrainte redondante $z \leq 1$**.**

On peut penser que les modifications dûes à Cernikov pourraient être augmentées par la supression d'autres contraintes redondantes après chaque pas de l'algorithme de Fourier, mais le résultat est alors faux en général en effet:

Exemple 18 *On reprend l'exemple précèdent:*

$$\begin{array}{lrrrl} \{1\} & & w+x+y+z & & \leq 1 \\ \{2\} & & w-x+y+z & & \leq 1 \\ \{3\} & & -w+x+y+z & & \leq 1 \\ \{4\} & & -w-x+y+z & & \leq 1 \\ \{5\} & v & & -y & \leq 0 \\ \{6\} & -v & & & \leq 0 \end{array}$$

pas 1 *On applique l'algorithme de Fourier pour éliminer v on obtient:*

$$\begin{array}{lrl}
\{1\} & w+x+y+z & \leq 1 \\
\{2\} & w-x+y+z & \leq 1 \\
\{3\} & -w+x+y+z & \leq 1 \\
\{4\} & -w-x+y+z & \leq 1 \\
\{5,6\} & -y & \leq 0
\end{array}$$

pas 2 *Ensuite on élimine w on obtient:*

$$\begin{array}{lrl}
\{1,3\} & x+y+z & \leq 1 \\
\{1,4\} & y+z & \leq 1 \\
\{2,3\} & y+z & \leq 1 \\
\{2,4\} & -x+y+z & \leq 1 \\
\{5,6\} & -y & \leq 0
\end{array}$$

pas 3 *Le critère de Cernikov ne nous permet de suprimmer aucune contrainte. Mais puisque la seconde et la troixième contrainte sont dupliquées, on peut en supprimer une, mais on ne va pas la faire. Une fois x éliminé, on obtient :*

$$\begin{array}{lrl}
\{1,2,3,4\} & y+z & \leq 1 \\
\{1,4\} & y+z & \leq 1 \\
\{2,3\} & y+z & \leq 1 \\
\{5,6\} & -y & \leq 0
\end{array}$$

Les trois premières contraintes sont identiques, on suprime la seconde et la troixième et on obtient:

$$\begin{array}{lrl}
\{1,2,3,4\} & y+z & \leq 1 \\
\{5,6\} & -y & \leq 0
\end{array}$$

pas 4 *Dans le pas final de Fourier, on élimine y, on obtient:*

$$\{1,2,3,4,5,6\} \quad z \leq 1$$

Le critère de Cernikov nous permet de supprimer cette contrainte (car $6 \geq 4 + 2$*) et alors on obtient un ensemble vide de contraintes. Ce résultat est faux car on vient de montrer plus haut que l'on a:*

$$\exists\ v, w, x, y\ \mathcal{A} \longleftrightarrow (z \leq 1)$$

.

Combinaison de la méthode basée sur Fourier avec l'élimination des contraintes strictes redondantes

Définition 22 *On définit* $C \rightsquigarrow c$ *ssi pour toute contrainte c',* $C \rightarrow c'$ *et* $c' \rightarrow c$ *mais* $c \not\rightarrow c'$*. Si* $c \in C$ *alors c est dite strictement redondante. Géométriquement la contrainte redondante stricte c détérmine un hyperplan qui n'intersecte pas le volume définie par C.*

Remarque:

• Il est claire que la redondance quasi syntaxique est une sorte de redondance stricte.

Dans [10], Jaffar et Maher ont montré que les modifications de Cernikov à l'algorithme de Fourier rajoutées à l'élimination de contraintes stictement redondantes améliorent cet algorithme qui reste correct et qui est représenté dans la figure 4.2**.**

4.3 Contraintes sur les Arbres (termes)

Les contraintes proposées sont des équations sur les termes.

Exemple 19 *Si x et y sont des variables cibles* $x = f(z), y = z + 2$ *peut être simplifiée en* $x = f(y - 2)$*.*

Un problème bien connu de la simplification sur les équations est que cette sortie peut être exponentiellement plus grande que les termes originaux. En effet:

Exemple 20 *Soit* $x_1 = f(x_2, x_2), x_2 = f(x_3, x_3), \ldots, x_{n-1} = f(x_n, x_n), x_n = a,$
où $x_1, \ldots x_n$ *sont des variables cibles, dans telle cas* x_1 *est un terme d'ordre* $O(2^n)$*.*

La simplification des contraintes sur les fonctions a besoin d'une autre approche. On rappelle qu'il n'est pas toujours possible d'éliminer toutes les variables non cibles apparaissant dans une équation. En effet, l'exemple suivant montre la difficulté du problème.

```
C= initial set of inegalities(after linear substitutions);
label(c) = {c} for each c ∈ C;
n = 0;
While (there exists an auxiliairy variable x in C) {
   choose a variable x with minimal measure(x.C);
   D = C_x^0;
   if(|C_x^+| > 0 and |C_x^-| > 0 ) {
      n = n + 1 /* count active eliminations */
      for (each pair c_k ∈ C_x^+, c_l ∈ C_x^-)
        if (label(c_k) ∪ label(c_l)) ≥ n + 2) continue /*first Cernikov method *
        d is the contrainte obtained from c_k and c_j eliminating x;
        label(d) = label(c_k) ∪ label(c_l);
        if (d is not quasi-syntaxic redundant wrt D) {
            E=quasi syntactic redundant contraints in D wrt d;
            D = D ∪ {d} − E;
/* * second cernikov method
            if (d is label-subsumed in D )
              D = D − {d};
            else {
              F= contraints in D label-subsumed by d;
              D = D − F;
            }
* */
        }
   }
   C = D
}
return C
```

Figure 4.2 : Inégalités linéaires

Exemple 21 *L'élimination de z dans $x = f(z)$.*

Les variables cibles secondaires ont une priorité inférieure à celles des variables cibles originales dans le sens de minimiser leurs occurences dans la sortie voir section (4.5).

4.4 Contraintes non linéaires

En général toutes les contraintes non linéaires doivent être affichées, indépendamment des variables cibles, en effet:

Exemple 22 *Les deux contraintes $x < 0$, $y * y = -2$ où x est la variable cible, ne peuvent être sorties comme $x < 0$.*

On voit que la sortie serait satisfiable alors que l'entrée ne l'est pas. Ainsi comme dans les équations sur les fonctions, le problème des variables cibles secondaires se pose. Ce sont tout simplement les variables qui restent dans les contraintes non linéaires et on leur donne une priorité inférieure à celles des variables cibles mais supérieure à celles des variables supplémentaires dans les équations sur les fonctions.

Cependant il existe une observation qui peut significativement réduire le nombre des équations non linéaires affichées et le nombre de variable cibles supplémentaires: Elle suppose que la variable non cible y figure exactement une fois dans les contraintes, disons dans c, et $p(\tilde{x})$ implique $\exists y c(\tilde{x}, y) \leftrightarrow c'(\tilde{x})$, pour une contrainte c' et une condition p, alors c peut être remplacée par c', pourvu que le reste des contraintes impliquent que $p(\tilde{x})$ tient. La suite de ce paragraphe est des applications spécifiques de ces observations.

• Si y figure sous la forme $y = f(\tilde{x})$, alors cette contrainte peut être éliminée pourvu que f soit total sur les nombres réels.

• Si y figure sous la forme $y = x^z$, alors cette contrainte peut être éliminée pourvu que $x > 0$ ou z entier autre que zéro. Simulairement, on peut suprimer $x = z^y$ pourvu que $x > 0$ ou $z > 0$ et $z \neq 1$ et suprimer $x = y^z$ pourvu que $x > 0$ et $z \neq 0$.

• La contrainte $x = |y|$ peut être remplacée par $x \geq 0$. La contrainte $x = siny$ peut être remplacée par $-1 \leq x \leq 1$, $x \leq min(y, z)$ peut être remplacée par $x \leq z$, $x \leq y$. La contrainte $x = y * z$ peut être éliminée pourvu que $z \neq 0$.

4.5 Sommaire du module sortie

On présente maintenant l'algorithme de sortie comme collection des différents sous algorithmes décrits plus haut correspondant aux différentes sortes de contraintes. On note que l'ordre dans lequel les sous algorithmes sont invoqués est important:

Pas 1 *Processus des équations sur les termes, dans le but d'obtenir les variables cibles secondaires. Ce sont essentielement des variables non cibles apparaissant dans l'obligation des variables cibles primaires. On obtient une collection d'équations sur les termes.*

Pas 2 *Simplification des équations non linéaires et ajout de la collection simplifieé aux variables cibles secondaires. Obtention d'une collection d'équations non linéaires. Ce pas peut aussi produire des équations linéaires supplémentaires.*

Pas 3 *Processus des équations linéaires en respectant les variables cibles primaires et secondaires, utilisant des priorités telle que les variables primaires sont de priorité supérieure aux variables secondaires et les variables auxiliaires sont de priorité inférieure. Obtention d'une collection d'équations linéaires finale contenant seulement les variables cibles.*

Pas 4 *Processus des inégalités linéaires, et notons qu'elles peuvent être modifiées comme le résultat du* 3^{eme} *pas au dessus, utilisant les variables cibles primaires et secondaires. Obtention d'une collection d inégalités linéaires contenant seulement les variables cibles.*

Pas 5 *Pour chaque variable cible secondaire y apparaîssant dans une équation de la forme* $y = t$*, substituer y par t partout ailleurs, et supprimer l'équation. Pour chaque variable cible secondaire y apparaîssant dans une equation non linéaire de la forme* $y = t$*, quand y apparaît ailleurs mais non dans t, substituer y par t partout ailleurs, et supprimer l'équation.*

Pas 6 *Sortie de toutes les contraintes restantes.*

4.6 Conclusion

Dans ce chapitre le module sortie de CLP(R) a été décrit. L'élément essentiel était l'extension de l'algorithme de (Fourier,Cernikov) à l'élimination des contraintes redondantes strictes. Le reste du chapitre était consacré aux équations sur les termes, aux équations non linéaires et comment elles pouvaient être simplifiées avec les contraintes linéaires. Ce qui est enfin obtenu est le module sortie pour CLP(R) qui s'avère pratique et efficace.

Chapitre 5

Sorties dans les domaines finis

Ce chapitre, est une extension du chapitre précèdent, son but est de simplifier un ensemble de contraintes données sur les domaines finis. Par simplification, en entend réduire le plus possible un ensemble de contraintes en un autre plus simple et plus lisible. Cependant, il faut bien faire la différence entre simplification et résolution qui ne signifient pas la même chose. Comme dans le cas de la programmation linéaire, le problème de simplification de contraintes sur les domaines finis est plus complexe que sur les domaines réels.

5.1 Introduction

Dans le chapitre précèdent, le module sortie de CLP(R) a été décrit en détail. La formulation du problème de simplification de contraintes sur les domaines finis est analogue à celle sur les domaines réels ou plus précisement:

Etant donné un ensemble $\tilde{x}$ de variables cibles et une conjonction $C(\tilde{x}, \tilde{y})$ de contraintes, où $\tilde{x}$, $\tilde{y}$ sont des variables domaines finis[1], on veut exprimer $\exists\tilde{y}$ $C(\tilde{x}, \tilde{y})$ dans sa forme la plus simple. Exemples:

x et y désignent des variables cibles:

(a) les contraintes $x = z + 1$, $z = y + 1$ peuvent être simplifiées en $x = y + 2$.
(b) les contraintes $x < z$, $z \leq y$, $z \leq y + 2$ peuvent être simplifiées en $x < y$.
(c) la contrainte $x = z * y$, peut être simplifiée en x est multiple de y.

Comme dans le cas des domaines réels, on peut classer la simplification de

[1] $\tilde{x}$ et $\tilde{y}$ prennent leurs valeurs dans une partie finie de l'ensemble des entiers naturels.

ces contraintes en 3 catégories:

(I) L'élimination de variables auxilaires (comme dans (a) et (b)) ;
(II) L'élimination de contraintes redondantes (comme dans(b)) ;
(III) Le remplacement des expressions par d'autres équivalentes et simples (comme dans (c)).

Le problème de l'élimination de contraintes redondantes (II) sur les domaines finis semble ne pas être difficile du fait qu'il peut être en partie traîté par des méthodes analogues à celles sur les domaines réels. Sans aucun doute, l'élimination de variables auxiliaires (I) est la partie la plus compliquée pour la simplification de contraintes sur les domaines finis, en effet, l'exemple suivant montre bien la difficulté du problème:

Exemple:

Sur les domaines réels la formule $\exists\ x \mid y = 2x$ *peut être simplifiée en* y *réel.*
Sur les domaines finis la formule $\exists\ x \mid y = 2x$ *peut être simplifiée en* y *pair.*

Donc l'élimination de Fourier devient incorrecte.

Le problème de la recherche des expressions équivalentes est aussi en général un problème très difficile.

5.2 Elimination de contraintes redondantes

Le problème II d'élimination de contraintes redondantes est en général un problème très difficile, comme celui de la satisfiabilité de contraintes. Ce problème de redondances pour le cas réel était traité en détail dans le chapitre 3.

Dans le chapitre 3 on a défini les classes de redondances sur des domaines continues (réels) en détail dans le paragraphe 3.2.

Notre première approche est d'étendre toutes ces classes de redondances qui étaient définis sur des domaines réels aux domaines finis. On rappelle cette fois que notre polyèdre $P = \{Ax \leq b\}$, sera entier càd:

- **toutes les variables de la matrice A seront des entiers naturels.**
- **toutes les variables du vecteur x seront des entiers naturels.**

- **toutes les variables du vecteur b seront des entiers naturels.**

Les définitions seront les même que celles pour les domaines continus, sauf que cette fois, il faut à chaque fois rajouter les conditions ci-dessus. Par exemple pour la redondance indépendante, la définiton sera comme suit:

Définition 23 *la contrainte $a_i x \leq \beta_i$ (avec x un vecteur entier positif ou nul) est* **indépendante redondante** *dans le système $\{Ax \leq b, x$ vecteur entier positif ou nul $\}$ si son hyperplan correspondant intersecte l'enveloppe affine de l'ensemble polyèdre mais l'intersection ne contient aucune facette de ce polyèdre.*

Exemple 23 *On reprend le polyèdre $P = \{-x + y \leq 0, x + y \leq 4, -x + 2y \leq 2, y \leq 3, x$ entier $\geq 0, y$ entier $\geq 0\}$, défini dans le cas continu, sauf que cette fois on a rajouté les deux contraintes x entier ≥ 0 , y entier ≥ 0.*

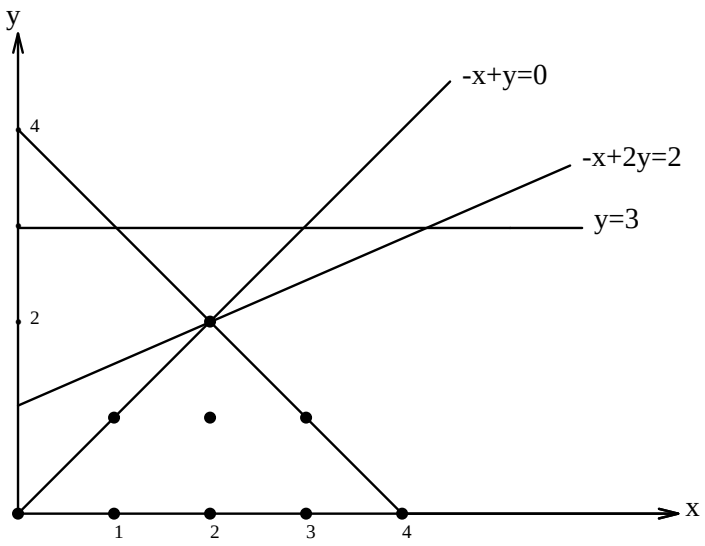

Figure 5.1 : Indépendante redondance

- *Le polyèdre P étant celui défini par les 8 points représentés en gras.*
- *l'enveloppe affine de P étant le plan (car c'est le plus petit espace contenant le polyèdre P).*
- *L'intersection entre l'enveloppe affine de P et la droite d'équation $-x + 2y = 2$ est cette droite elle même.*
- *Les facettes de P sont les trois segments d'extrimités $((0,0);(2,2))$, $((4,0);(2,2))$ et $((0,0);(4,0))$ représentées sur la figure ci-dessus.*

Donc l'hyperplan $-x + 2y = 2$ intersecte l'enveloppe affine de P et cette intersection ne contient aucune facette de P. Par conséquent la contrainte $-x+2y \leq 2$ est redondante indépandante. De même la contrainte $y \leq 3$ est redondante indépandante.

Cependant le problème de la détection de cette classe de redondance se pose, en effet, on a vu que pour détecter cette classe de redondance dans le cas continu, il fallait résoudre le problème de maximisation : $max\ \{a_i x \mid a_j x \leq \beta_j\ ,\ j \neq i\}$, qui indique que la contrainte $a_j x \leq \beta_j$ est redondante indépendante si la valeur retournée par le programme linéaire est inférieure ou égale à β_i.

Un autre problème est celui de la détection des égalités implicites, en effet, la contrainte $a_i x \leq \beta_i$ est une égalité implicite si et seulement si β_i est retournée comme solution optimale par le programme linéaire $min\ \{a_i x \mid Ax \leq b\}$.

Dans le cas réel, ces problèmes d'optimisation avec contraintes ne sont pas difficile à résoudre (en utilisant par exemple la méthode du simplexe), ce qui n'est pas le cas pour les domaines finis.

Dans le chapitre 2, on a traîté en détail une méthode permettant de résoudre ces problèmes de programmation linéaire en nombre entier. Le problème de la détection des égalités implicites et de la redondance indépendante est donc résolu.

On note que pour les tautologies, redondances syntaxique et redondance quasi syntaxique, ce problème de detection ne se pose pas du fait que leurs détection dans la cas domaine finis, est pareil que celle dans les domaines réels.

Les autres sortes de redondances: hull redondance, facette redondance, quasi facette redondance et redondance implicite, leurs détection n'est pas du tout facile, mais comme dans le cas continu, ces contraintes vont être transformées par la hull transformmation respectivement en: tautologie, redondance syntaxique, redondance quasi syntaxique, égalité redondante.

Cependant, le lecteur peut se demander si la procédure de la représentation paramétrique définie par la procédure de Gauss (dans le cas continu) peut être étendue au cas des domaines finis. Or, on sait que notre polyèdre initial P n'est pas vide. Donc la procédure de Gauss Jordan appliquée après que toutes les égalités implicites soient détectées, retournera au moins une solution entière. Au passage, on note que la procédure Eliminate Bound variable, elle aussi ne causera pas de problème.

Algorithme

L' algorithme sera le même que celui défini sur le cas domaine continu qui est basé sur la hull transformation pour la suppression des contraintes redon-

dantes. Il utilise 3 procédures prédéfinies:

• La première est une boite noire pour la résolution de problèmes de programmation linéaire appelée min ou max qui accepte une fonction objective avec un ensemble de contraintes et qui retourne une valeure optimale (elle ne peut pas retourner $\pm\infty$ car les domaines sont finis, le polyèdre est borné). Cette procédure est celle qui est défini dans le chapitre 2 et plus précisement dans le rappel.

• La deuxième est appelé Gauss-Jordan qui prend un ensemble d'égalités comme paramétre et les retourne en forme de lignes échelonnées après l'élimination de Gauss-Jordan.

• La troixième appelée Eliminate Bound variables est auto explicative. (c'est une substitution)

On suppose que les tautologies ont été suprimées après que toutes les variables liées aient été éliminées. Les redondances syntaxiques et quasi-syntaxiques sont éliminées dans la procédure Remove Syntactic Redundancy. Pour chaque contrainte $a_i x \leq \beta_i$ dans P, on associe une variable booléene appelée $redundant_i$ qui est utilisée pour indiquer si la contrainte est redondante.

Remarque 1

On rappelle que notre but n'est pas de résoudre un système de contraintes sur les domaines finis, mais plutôt de simplifier le plus possible ce système en un autre plus simple et plus lisible.

Dans [2], Bockmayr a proposé une méthode pour résoudre des systèmes de contraintes linéaires pour les pseudo booléens[2], en utilisant des approximations pour l'enveloppe convexe de l'ensemble des solutions entières, par des plans de coupes. L'une de nos approches est d'étendre ces techniques pour un système de contraintes linéaires sur les domaines finis en effet:

En remarquant qu'à toute variable x ne pouvant prendre que $k+1$ valeurs entières $0,\ 1,\ 2,\ \ldots, k$ on peut substituer une combinaison linéaire:

$$x = y_0 + 2y_1 + 4y_2 + \ldots + 2^p y_p$$

de $p+1$ variable: $y_0, \ldots, y_p$, chacune d'elles étant astreinte à ne prendre que deux valeurs 0 ou 1 (variables bivalentes) (p est le plus petit entier tel que $k \leq 2^{p+1} - 1$).

On peut de la sorte toujours se ramener au cas où le système linéaire sur les variables domaines finis est un système linéaire sur les variables bivalentes.

[2]Toute variable, est astreinte à ne prendre que deux valeurs 0 ou 1 (variables bivalentes)

(N.B. : la transformation précèdente, qui est toujours possible, n'est pas nécessairement toujours avantageuse en pratique.)

Remarque 2

Dans le chapitre 3 et plus précisement dans la section 3.2.6, on a vu que la contrainte $-x + y \leq 15$ était une contrainte quasi-facette redondante, mais la contrainte $-x + 2y \leq 16$ ne l'était pas. Pour le cas des domaines finis, on voit bien (voir figure 5.2) que cette contrainte est devenue redondante. Donc dans le cas des domaines finis, on peut définir de nouveaux types de redondance. Malheureusement, la détection de ces classes de redondances n'est pas facile. Donc, on a préféré se limiter aux seuls types de redondances déjà définis dans le cas du domaine réel et étendre l'algorithme donné pour le cas du domaine réel au cas du domaine fini.

Exemple 24 *Soit* $P = \{x + y \leq 5, -x - y \leq -5, -x + 2y \leq 16, -x + y \leq 15\}$.
(voir figure 5.2)

5.3 Contraintes linéaires

Soit C un ensemble de contraintes linéaires.

Soit $\tilde{x} = x_1, x_2, \ldots, x_N$ les variables cibles dans C.

Soit $\tilde{y} = y_1, y_2, \ldots, y_M$ les variables auxiliaires dans C.

Comme dans le cas des domaines continus, l'approche traditionnelle pour la simplification de contraintes, est d'utiliser la notion de *forme canonique* équippé par un algorithme efficace pour son calcul. Pour les équations linéaires, il est bien connu que la forme canonique est appelée forme paramétrique où les équations sont représentées dans la forme $\tilde{x} = t(\tilde{y})$ où les $\tilde{y}$ sont distinctes des $\tilde{x}$ et t est un tuple d'expressions linéaires.

Dans ce paragraphe, on décrit un algorithme pour la sortie de $\exists\ \tilde{y}\ C(\tilde{x}, \tilde{y})$, où C est linéaire, en fonction uniquement de variables cibles, traitant d'abord les équations et aprés les inéquations. L'algorithme doit cette fois produire une sortie contenant la variable cible particulière x qui apparaît dans C, par exemple si on élimine y de $\exists\ y \mid x = 2y$ (sinon l'information x est pair sera perdu). Donc contrairement au cas des domaines continus, l'algorithme ne va modifier que les contraintes de la forme $\tilde{x} = t'(\tilde{y})$ où t' est un tuple d'expressions linéaires formé de coefficients égale à 1 ou à -1 i.e: toute variable x_i est de la forme: $x_i = \sum_i \alpha_i y_i$ où $\alpha_i \in \{1, -1\}$.

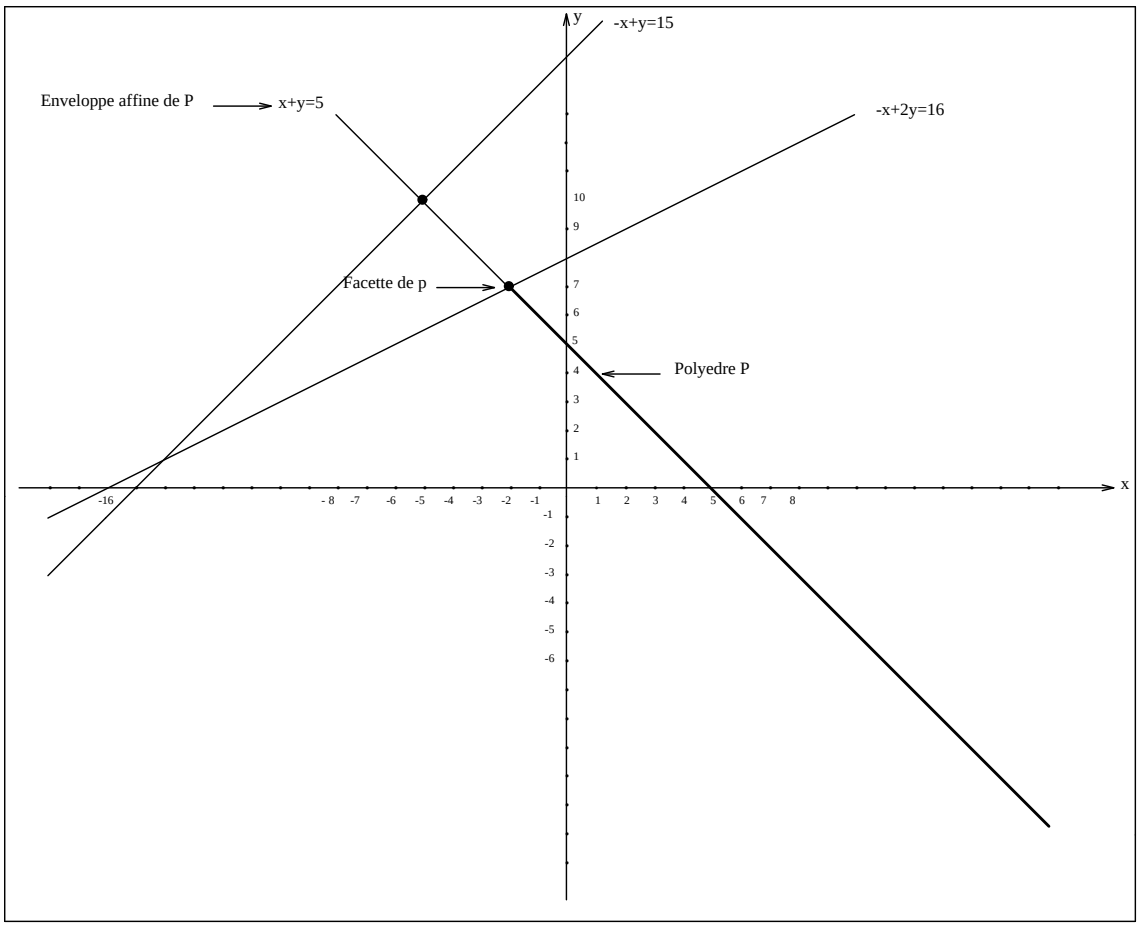

Figure 5.2 : Exemple de redondance dans les domaines finis

5.3.1 équations linéaires

Les équations sont toujours maintenues sous une *forme paramétrique*, $\tilde{u} = t(\tilde{v})$ où les variables de $\tilde{u}$ sont appelées variables objets et sont distincts des variables de $\tilde{v}$, appelés, quant à elles, paramétres. Pour chaque variable objet z on note par $r.h.s(z)$ l'expression linéaire qui contient z.

L'algorithme a la même forme que celui sur les domaines continus, sauf que cette fois, il ne va modifier que les sortes d'équations décrit ci-dessus. L'algorithme a la forme d'élimination de Gauss, il est décrit dans la figure 5.3, il suppose qu'il y a une priorité entre les variables, $\pi(x_1) > \ldots > \pi(x_N) > \pi(y_1) > \ldots > \pi(y_M)$, exprimant l'importance relative de chaque variable dans la sortie. L'algorithme assure que les variables de priorité supérieure sont représentées en fonction de variables de priorité inférieure. Les autres sortes d'égalités linéaires seront simplifiées avec d'autres contraintes.

```
for (i = 1; i ≤ N; i = i + 1){
      if (x_i is a parameter) continue;
      let ε denote the equation x_i = r.h.s(x_i) at hand ;
      if (r.h.s(x_i) contains a variable z of lower priority than x_i) {
            choose the z of lowest priority;
            rewrite the equation ε into the form z = t;
            if (z is a target variable) mark the equation ε as final;
            if (t is a linear combinatory not integer)
               { write z = t into the form x_i = r.h.s(x_i);
                 mark the equation ε as final;
                }
            substitue t for z in the other linear équations and inegalities;
      } else mark the equation ε as final;
}
return all final équations
```

Figure 5.3 : Equations linéaires

Remarque:

On peut penser à rendre la contrainte $x = 2y$ sous la forme $x = y + y$, qui devient du type des contraintes ci-dessus et ensuite appliquer l'algorithme décrit ci-dessus, malheureusement ces contraintes doivent être maintenues sous une forme paramétrique.

5.3.2 Inégalités linéaires

Dans ce paragraphe, on adopte les mêmes démarches que celles pour les domaines continus ie: algorithme de Fourier + Cernikov + élimination de contrainte redondante stricte. Du fait que notre but est de simplifier un système d'inégalités linéaires et non pas le résoudre, on peut étendre l'algorithme donné dans le cas continu au cas du domaine fini. Pour plus de précisions, voyons l'exemple suivant:

Exemple 25 *On veut simplifier la formule* $\exists\ y \mid x + y \leq 1$ *et* $x - y \leq 2$

En appliquant l'algorithme de Fourier (en sommant les deux inégalités), on obtient la formule équivalente $x \leq 3/2$.

Géométriquement, la contrainte $x \leq 3/2$ *n'est autre que la projection du polyèdre* $P = \{x + y \leq 1, x - y \leq 2\}$ *sur l'axe des abscisses. (voir figure* 5.2*)*

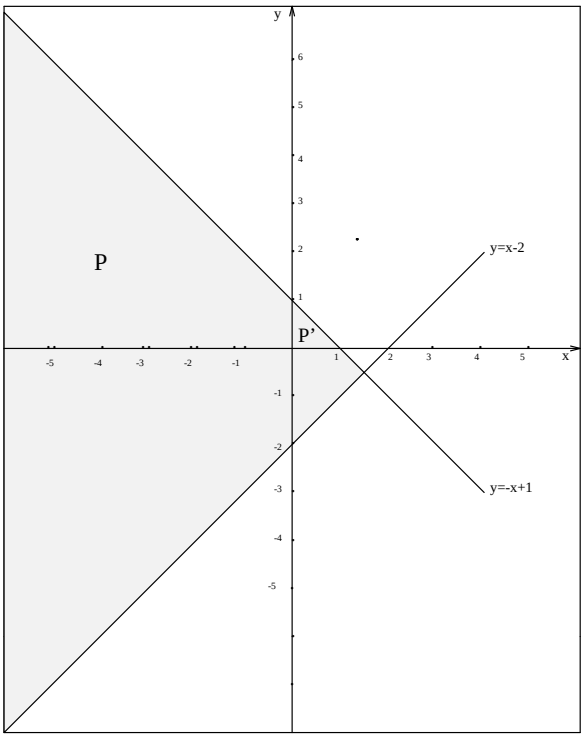

Figure 5.4 : Simplification d'inégalités linéaires

Dans le cas du domaine fini, la projection du polyèdre entier $P' = \{x + y \leq 1, -x - y \leq 2, x\ entier\ \geq 0, y\ entier\ \geq 0\}$ **sur l'axe des abscisses est la contrainte** $x \leq 1$. **(voir figure** 5.4**)**

Donc la simplification des inégalités linéaires sur les domaines finis est la même que celle sur les domaines finis, sauf que lorsqu'on rencontre une

inégalité de la forme $x \leq 3/2$ on la remplace par $x \leq E(3/2)$, où E(x) désigne la partie entière de x.

Remarque

Le processus de la simplification des contraintes sur les termes est analogue à celui sur les domaines continus. On note que les fonctions sont définis sur des entiers naturels et prennent leurs valeurs dans l'ensemble des entiers naturels.

5.4 Contraintes non linéaires

Dans la section 5.3, on était confronté au problème de la simplification de la contrainte $\exists y \mid x = 2y$. On a préféré garder cette contrainte telle qu'elle est, c'est à dire que ces sortes de contraintes ne vont pas être simplifiées avec les contraintes linéaires. Elles vont plutôt être simplifiées avec les contraintes de la forme $y \times z$, qui sont des contraintes non linéaires.

- **La contrainte $x = y \times z$, peut être remplacée par x est multiple de z.**
- **La contrainte $x = |y|$ peut être supprimée, dans la sortie final, on ajoutera entier (x) qui signifie tout simplement que x est un entier naturel.**
- **La contrainte $x \leq min(y, z)$ peut être remplacée par $x \leq z$, $x \leq y$.**
- **La contrainte $x = y^2$, peut être remplacée par x est un carée parfait.**
- **La contrainte $x = 2y$, peut être remplacée par x est paire.**
- **La contrainte $x = 2y + 1$, peut être remplacée par x est impaire.**

5.5 Sommaire du module sortie

On présente maintenant l'algorithme de sortie comme collection des différents sous algorithmes décrits plus haut correspondant aux différentes sortes de contraintes. On note que l'algorithme a la même forme que celui sur les domaines continus.

Pa 1 *Processus des équations sur les termes, dans le but d'obtenir les variables cibles secondaires. Ce sont essentielement des variables non cibles apparaissant dans l'obligation des variables cibles primaires. On obtient une collection d'équations sur les termes.*

Pa 2 *Simplification des équations non linéaires et ajout de la collection simplifieé aux variables cibles secondaires. Obtention d'une collection d'équations non linéaires. Ce pas peut aussi produire des équations linéaires supplémentaires.*

Pa 3 *Processus des équations linéaires en respectant les variables cibles primaires et secondaires, utilisant des priorités telle que les variables primaires sont de priorité supérieure aux variables secondaires et les variables auxiliaires sont de priorité inférieure. Obtention d'une collection d'équations linéaires finale contenant seulement les variables cibles.*

Pa 4 *Processus des inégalités linéaires, et notons qu'elles peuvent être modifiées comme le résultat du* 3^{eme} *pas au dessus, utilisant les variables cibles primaires et secondaires. Obtention d'une collection d inégalités linéaires contenant seulement les variables cibles.*

Pa 5 *Pour chaque variable cible secondaire y apparaîssant dans une équation de la forme* $y = t$*, substituer y par t partout ailleurs, et supprimer l'équation. Pour chaque variable cible secondaire y apparaîssant dans une equation non linéaire de la forme* $y = t$*, quand y apparaît ailleurs mais non dans t, substituer y par t partout ailleurs, et supprimer l'équation.*

Pa 6 *Sortie de toutes les contraintes restantes.*

Chapitre 6

Conclusion

Durant ce stage, d'une durée de quatre mois, j'ai été amené à étudier les solutions proposées pour la simplification de contraintes sur les domaines continus et à proposer des méthodes pour résoudre le problème de simplification de contraintes sur les domaines finis.

Ce travail a commencé par une synthèse des travaux sur la sortie (simplification, présentation) de contraintes sur les domaines continus.

On a rappelé les grandes classes de contraintes redondantes et les méthodes d'élimination de variables auxiliaires ainsi que des techniques pour les identifier. Cette synthèse s'achève par l'algorithme utilisé dans CLP(R) pour l'affichage des réponses.

Concernant les domaines finis, les techniques proposées pour le continu constituent un début de réponse. En effet, une contrainte redondante sur le polyèdre continu l'est également sur le polyèdre entier. Seul le problème de la détection de certaines contraintes redondantes se pose, pour cela, on a proposé une méthode pour résoudre ce problème qui se ramène à un problème d'optimisation sur les domaines finis. La difficulté majeure est pour l'élimination des variables auxiliaires pour lequel la méthode de Fourier ne peut être utilisée. On a proposé des modifications pour adapter l'élimination de Fourier aux domaines finis.

Ces travaux devraient permettre l'implantation d'un module de présentation (simplification et affichage) de contraintes sur les domaines finis dans le cadre du projet ESPRIT DiSCiPl.

Références

[1] **Alexander Bockmayr and Thomas Kasper,** *Pseudo-Boolean and Finite Domain Contraint Programming: A Case Study.* **Im Stadtwald, D-66123 Saarbrücken, Germany.**

[2] **Alexander Bockmayr,** *Cutting Planes in Constraint Logic Programming.* **Technical Report MPI-I-94-207, Max-Plank-Institut für Informatik, Saarbrücken, February 1994.**

[3] **Alexander Bockmayr, Peter Barth,** *Finite Domain and Cutting Plane Techniques in CLP(PB).* **Im Stadtwald, D-66123 Saarbrücken, Germany.**

[4] **A. Colmerauer.** *Introduction to PROLOG III.* **In 4th Annual ESPRIT Conference, Bruxelles. North Holland, 1987.**

[5] **Alexandre Tessier,** *Approche, en termes de squelettes de preuve, de la sémantique et du diagnostic déclaratif d'erreur des programmes logiques avec contraintes*. **Thèse de doctorat en Informatique. Université d'Orléans, 1997.**

[6] **A. Aiba, K. Sakai, Y. Sato, D.J. Kawley, and R. Hasegawa,** *Constraint Logic Programming langage CAL.* **In Fifth Generation Computer Systems, Tokyo, 1988. Springer, 1988.**

[7] **Bruno De backer,** *Méthodes de résolution de disjonctions de contraintes linéaires. Application à la Programmation Logique avec Contraintes.*. **Thèse de doctorat en Informatique. Université d'Orléans, 1995.**

[8] **Daniel Diaz, Björn Carlson and Mats Carlsson,** *Entailment of Finite Domain Contraints.* **INRIA-Rocquencourt, domaine de Voluceau, 78153 Le Chesnay, France.**

[9] **Daniel Diaz,** *Etude de la compilation des langages logiques de programmation par contraintes sur les domaines finis: le système CLP(FD)*. **Thèse de doctorat en Informatique. Université d'Orléans, 1995.**

[10] **J. Jaffar and M.J. Maher,** *Contraint logic programming:* **A survey. Journal of Logic Programming, 1994.**

[11] **J. Jaffar and J.L. Lassez,** *Contraint logic programming.* **In Proc. 14th ACM Symp. Principales of Programming Languages, Munich, 1987.**

[12] **J.L. Lassez and M.J. Maher,** *On Fourier's Algorithm for Linear Arithmetic Contraints.* **Journal of Automated reasonning 9: 373-379, 1992.**

[13] **J.L. Lassez and K. McAloon,** *Simplification and Elimination of Redundant Linear Arithmetic Contraints.* **In Proc. North American Conference on Logic Programming. Cleveland. 1989. pp. 35 51.**

[14] **J.L. Lassez and K. McAloon,** *Generalized Canonical Forms for Linear Contraints and Applications.* **In Proc. Int. Conf. On Fifth Generation Computer Systems. ICOT. Tokyo,1988. pp. 703 710.**

[15] **M.J. Maher and J. Jaffar,** *output in CLP(R).* **Proceedings of the international conference on fifth generation computer systems, 1992.**

[16] **M. Minoux,** *Programmation Mathématique - Théorie et algorithmes.* **Dunod, Paris, 1983.**

[17] **M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, and T. Graf,** *The contraint logic programming language CHIP.* **In Fifth Generation Computer Systems, Tokyo, 1988. Springer, 1988.**

[18] **O. Lhomme and M. Rueher,** *Application des techniques CSP au Raisonnement sur les intervalles.* **Rapport de recherche numéro 94-59.**

[19] **P.G. Ciarlet,** *Introduction à l'analyse numérique matricielle et à l'optimisation.* **Masson, Paris, 1990.**

[20] **R.J. Duffin,** *On Fourier's Analysis of Linear Inequality System.* **Mathematical Programming study. Vol. 1 (1974), pp. 71-95.**

[21] **S.N. Cernikov,** *Contraction of Finite Systems of Linear Inequalities.* **Doklady Akademiia Nauk SSSR. Vol. 152. No. 5 (1963). pp.1075-1078. (English translation in soviet Mathematics Doklady. Vol. 4, No.5 (1963). pp. 1520-1524.)**