TreeKs : un foncteur pour faire passer les domaines numériques à l'échelle

Mehdi Bouaziz – responsable : Antoine Miné
École normale supérieure, équipe Abstraction

20 août 2010

Le contexte général

Notre travail s'inscrit dans le domaine de l'interprétation abstraite initié par Patrick Cousot à la fin des années 70. L'interprétation abstraite est une théorie de l'approximation sûre de sémantiques de langages de programmation principalement utilisée pour l'analyse et la vérification statique de programmes.

Plus particulièrement, notre travail s'intéresse aux domaines numériques abstraits dont se servent ces analyseurs pour représenter les valeurs numériques d'un programme et en approximer les calculs. Dès la fin des années 70 coexistent le domaine simple et peu expressif des intervalles permettant de borner les valeurs de chaque variable et le domaine des polyèdres très coûteux mais permettant de représenter toutes les inégalités linéaires entre les variables d'un programme.

Depuis, et surtout cette dernière décennie, de nouveaux domaines numériques — dits domaines faiblement relationnels — ont été conçus pour répondre à différents besoins d'expressivité, de précision et de coût. Le plus connu d'entre eux étant certainement le domaines des octogones qui permet de borner les sommes et les différences de toute paire de variables.

Le problème étudié

Parallèlement, les besoins d'analyse et de vérification statiques se sont développés et l'on a maintenant à faire face à l'analyse de programmes de très grande taille, comme les programmes des avions Airbus. Néanmoins, seuls les intervalles sont capables, sans modification, de passer à l'échelle d'une telle analyse.

Il y a donc une nécessité de développer de nouveaux outils afin de répondre à cette demande, à la fois de manière précise et efficace. C'est pourquoi des analyseurs, comme Astrée, ont vu le jour. Pour faire passer à l'échelle les domaines numériques faiblement relationnels, ils découpent ces derniers en fragments, appelés *packs*. Cependant ce découpage reste incomplet car ils engendrent d'importantes pertes d'informations, qui peuvent rendre l'ana-

lyse plus longue et moins précise. Il nous semblait à la fois nécessaire et possible d'y apporter une amélioration.

La contribution proposée

S'il est impossible de conserver toutes les relations entre toutes les variables, il est néanmoins possible d'en conserver un certain nombre et de les choisir judicieusement afin de garder une précision maximale.

Là où le graphe des relations était complet, nous avons proposé de nous restreindre à un sous-graphe. À partir de là, nous avons étudié différentes formes de sous-graphe afin de trouver celles qui permettraient de garder un maximum de relations tout ayant un coût en temps le plus faible possible.

Ce graphe a la forme d'un arbre de graphes complets, ce qui explique le nom de notre domaine. Nous l'avons d'abord appliqué au domaine des octogones puis l'avons généralisé en définissant un cadre formel qui englobe tout domaine numérique abstrait d'inégalités linéaires.

Les arguments en faveur de sa validité

Cette solution permet assurément de réduire le nombre de relations stockées et est évidemment meilleure que toute solution sans relation. Cependant, il fallait également montrer son effectivité. Pour cela nous avons développé les algorithmes nécessaires aux différentes opérations d'un domaine abstrait et en avons prouver la correction et l'efficacité.

Même si de manière sûre notre solution est au moins aussi efficace et précise qu'une solution avec des packs sans relations, il reste à la mettre à l'épreuve d'une implémentation. Malheureusement celle-ci est plus complexe que l'implémentation d'un domaine quelconque, nous n'avons pas eu le temps de la terminer.

Le bilan et les perspectives

Notre approche est certes générale mais peut encore sans doute être étendue selon les besoins à d'autres classes de domaines numériques. Il faudra attendre la fin de l'implémentation pour mesurer exactement les retombées en termes d'efficacité et de précision. À ce moment, nous espérons que notre approche pourra être utilisée dans des analyseurs comme Astrée et que d'autres chercheurs pourront profiter de notre travail afin de développer des domaines abstraits qui passent à l'échelle.

Il n'y a pas une *bonne* question à poser, notre travail ouvre de nombreuses perspectives dans l'amélioration de l'efficacité des domaines relationnels, notamment dans la recherche de formes efficaces de sous-graphes, de mélanges de différents domaines numériques partageant des relations, et surtout dans l'étude de la meilleure expressivité d'un domaine en fonction des relations explicites ou implicites présentes dans un programme.

TreeKs : un foncteur pour faire passer les domaines numériques à l'échelle

Mehdi Bouaziz, École normale supérieure

Stage de master 2 (mars–juillet 2010)

1 Introduction et motivation

1.1 Interprétation abstraite et domaines numériques

L'interprétation abstraite est une théorie de l'approximation sûre de sémantiques de langages de programmation [6] principalement utilisée pour l'analyse et la vérification statique de programmes. Un point important dans la conception d'un interpréteur abstrait est le choix de domaines abstraits adaptés. Un domaine numérique abstrait est la donnée d'un ensemble permettant de représenter une sur-approximation des valeurs numériques d'un programme, et d'algorithmes efficaces pour calculer des abstractions des opérations du langage.

Intervalles Le domaine des intervalles est présent dès les origines de l'interprétation abstraite [5]. Il permet d'approximer les valeurs prises par une variable numérique par un intervalle, éventuellement infini. C'est sans doute le plus utilisé des domaines non relationnels, car il permet de représenter les propriétés intéressantes dans la vérification statique de bornes d'indices de tableaux, de dépassement de capacité, de division par zéro, etc.

Polyèdres Très vite, l'interprétation abstraite non relationelle a montré ses limites. Même si l'on ne souhaite observer que des intervalles, les preuves d'invariants de boucles peuvent nécessiter des relations plus complexes. Le domaine des polyèdres [17, 8] a été conçu pour exprimer toutes les inégalités linéaires existant entre les variables d'un programme. C'est donc un domaine très précis, mais cette précision a un coût — exponentiel en la taille du programme — qui le restreint à être utilisé dans l'analyse de programmes de petite taille.

Domaines intermédiaires De nombreux domaines numériques abstraits ont été conçus pour capturer des propriétés de différentes formes, dont des congruences, les grid [9], les quadtrees [15], les ellipsoïdes [10], etc.

Nous nous concentrerons sur les propriétés d'inégalités linéaires car la majorité des invariants prend cette forme dans la plupart des programmes. Depuis une dizaine d'années, on s'intéresse à des domaines numériques relationnels dont le but est de pouvoir trouver des propriétés plus complexes (par exemple $x < y$) qu'avec les intervalles mais dont le coût en temps est inférieur à celui des polyèdres afin de faire passer à l'échelle les analyses de programmes de grande taille.

1.2 Domaines numériques abstraits intermédiaires

Des polyèdres améliorés De nombreux travaux ont été effectués sur les polyèdres dans le but de les accélérer [12, 19]. Cependant le coût reste excessif et ce genre de domaine n'est pas utilisé dans l'analyse de programmes de très grande taille [7].

Domaines faiblement relationnels Les domaines faiblement relationnels permettent d'exhiber des propriétés sur un petit nombre de variables. Ce sont surtout des domaines 2-relationnels qui ont été développés car le nombre de relations est quadratique en le nombre de variables. Nous pouvons citer les domaines suivant : les zones (ou DBMs *difference bound matrices*) [21] qui permet de borner les différences entre toute paire de variables ; les octogones [22] qui, en plus, permet de borner les sommes de deux variables ; les logaèdres [14] qui permet de borner les différences et les sommes de deux variables coefficientées par une puissance de deux ; les TVPI (*two variables per inequality*) [25] où cette fois les coefficients peuvent être quelconques ; les octaèdres [4] qui est un mélange des polyèdres et des octogones. Le coût des opérations sur ces domaines est généralement polynomial (au minimum cubique). Ils donnent donc de très bons résultats sur des programmes de taille petite à moyenne mais finalement ne passent pas à l'échelle non plus s'ils sont utilisés tel quel.

À ces cinq domaines, s'ajoutent d'autres domaines faiblement relationnels qui ont été développés pour des besoins spécifiques comme les pentagones [20] ou les DBMs avec inégalités [23].

Un point commun important de ces domaines est l'opération de *complétion* — qui se fait généralement par propagation de contraintes — dont le but est de rendre explicite les relations implicites. C'est une opération essentielle car nécessaire à la plupart des autres opérations, sa complexité est donc un point déterminant à l'efficacité du domaine.

Un autre domaine d'inégalités linéaire est celui des templates [24] où la forme des contraintes trouvées est choisie par l'utilisateur. Cependant l'algorithme proposé utilise les méthodes mathématiques génériques de résolutions de systèmes d'inégalités linéaires et n'est pas optimisé pour certaines formes.

1.3 Bilan

Les domaines faiblement relationnels d'inégalités linéaires offrent un bon rapport précision/coût. Cependant ils se révèlent inadaptés à l'analyse de très grands programmes où seuls des domaines de complexité linéaire ou quasi-linéaire — comme les intervalles — passent à l'échelle.

Pour les besoins d'analyses de programmes de grande taille [2, 7, 26], les domaines faiblement relationnels comme les octogones ne sont pas utilisés sur l'ensemble des variables mais sur des regroupements plus petits que nous appellerons *packs*, assurant ainsi une complexité linéaire. Cependant cette séparation engendre des pertes d'informations comme le montrent les deux exemples suivants dans le domaine des octogones.

Exemple 1 On a les packs $P_1 = \{t, x, y\}$ et $P_2 = \{t, x, z\}$, et les relations $C_1 = \{t \leq y, y \leq x\}$ d'une part, et $C_2 = \{x \leq z, z \leq t\}$ d'autre part. La complétion du pack 1 ajoute la relation $t \leq x$, celle du pack 2 ajoute $x \leq t$. Mais en l'absence d'échange d'informations relationnelles entre les packs, on ne peut pas en déduire que $t = x = y = z$, ce qui est pourtant le cas.

Exemple 2 On a les relations suivantes : $C_1 = \{y - t \leq 2, x - y \leq 2\}$, $C_2 = \{x + z \leq 4, z - x \leq 2, t - z \leq 2\}$ sur les mêmes packs. La complétion de P_1 n'ajoute que la relation $x - t \leq 4$. Celle de P_2 ajoute, entre autres, les relations $z \leq 3, t + x \leq 6, t - x \leq 4, t \leq 5$. Si on injecte la relation $t \leq 5$ dans le pack 1 et qu'on effecture une complétion, il en résulte, entre autres, les relations $x + t \leq 14$ et $x \leq 9$. Cette dernière relation peut être ajoutée au pack 2 mais une nouvelle complétion n'apporte rien de plus. Si maintenant on injecte au pack 1 les relations $t + x \leq 6$ et $t - x \leq 4$ alors on trouve la relation $x \leq 5$! On a ainsi montré que, même si seules les contraintes d'intervalles nous intéressent, l'absence de transfert d'informations relationnelles entre les packs entraîne une perte de précision.

1.4 Vers des domaines plus efficaces

Grâce à [1], on sait que la complétion du domaine des octogones sur les réels comme sur les entiers peut se résumer à l'application d'un algorithme de clôture de plus courts chemins (par exemple l'algorithme de Floyd-Warshall) suivi d'une passe de renforcement en $O(n^2)$.

Le problème APSP (*all pairs shortest paths*) est très étudié en théorie des graphes. De nombreux algorithmes ont été développés, de $O(n^3)$ (Floyd-Warshall), à récemment $O(n^3 \log^3 \log n / \log^2 n)$ [3]. Ces algorithmes sont parfois difficiles à implémenter, ou alors les constantes cachées sont importantes. C'est pourquoi l'algorithme de Floyd-Warshall reste, sur les graphes complets (le graphe des relations est souvent dense [26]), l'un des meilleurs en pratique grâce à sa simplicité. On peut donc améliorer la complexité théorique du domaine des octogones mais difficilement sa vitesse en pratique.

Par ailleurs, certains algorithmes pour le problème APSP sont plus efficaces sur des graphes creux ou ont été conçus pour des formes de graphes spécifiques, par exemple pour les graphes planaires en $O(n^2)$ [11]. Ceci nous pousse à nous intéresser à d'autres formes de graphes pour les octogones. En effet, un certain nombre de relations calculées par le domaine sont inutiles et n'apportent aucun résultat intéressant, comme par exemple la différence ou la somme de deux variables dont on ne connait que les bornes [26].

Le domaine que nous avons développé (section 3) exploite cette idée, qui est généralisée, sous la forme d'un foncteur, à une classe importante de domaines numériques abstraits dont nous avons défini le cadre formel en section 2. Enfin dans la section 4, nous montrons comment optimiser ce domaine dans le cas spécifique des octogones.

2 Domaines d'inégalités linéaires

Notations Dans la suite, on suppose fixé un ensemble fini de variables $\mathcal{V} \stackrel{\text{def}}{=} \{X_1, \ldots, X_N\}$, N représentera le nombre de variables. On note $\mathbb{Q}^\star \stackrel{\text{def}}{=} \mathbb{Q} \setminus \{0\}$ et $\mathbb{Q}_+^\star \stackrel{\text{def}}{=} \{x \in \mathbb{Q} \,|\, x > 0\}$. Pour tout ensemble S, on note $S^{\{n\}}$ l'ensemble des multi-ensembles de cardinalité n formés d'éléments de S.

Définition 1. Un *support* $\mathscr{D}$ de domaine d'inégalités linéaires est un sous-ensemble de $\bigcup_{i\geq 1}(\mathbb{Q}^\star)^{\{i\}}$ clos par sous-ensemble et clos par multiplication par un scalaire positif de $\mathbb{Q}_+^\star$. On dit de plus qu'il est k-relationnel s'il est un sous-ensemble de $\bigcup_{i=1}^{k}(\mathbb{Q}^\star)^{\{i\}}$.

Les supports des domaines d'inégalités linéaires connus sont les suivants :

$$\begin{aligned}
\mathscr{I}ntervals &\stackrel{\text{def}}{=} \mathbb{Q}^\star \\
\mathscr{Z}ones &\stackrel{\text{def}}{=} \mathbb{Q}^\star \cup \mathbb{Q}_+^\star \{\{1,-1\}\} \\
\mathscr{O}ctagons &\stackrel{\text{def}}{=} \mathbb{Q}^\star \cup \mathbb{Q}_+^\star \{\{-1,-1\},\{-1,1\},\{1,-1\},\{1,1\}\} \\
\mathscr{L}ogahedra_B &\stackrel{\text{def}}{=} \mathbb{Q}^\star \cup \mathbb{Q}_+^\star \left\{ \left\{ \pm 1, \pm 2^k \right\} \,\middle|\, -B \leq k \leq B \right\} \text{ avec } B \in \mathbb{N} \cup \{+\infty\} \\
\mathscr{T}VPI &\stackrel{\text{def}}{=} \mathbb{Q}^\star \cup (\mathbb{Q}^\star)^{\{2\}} \\
\mathscr{O}ctahedra &\stackrel{\text{def}}{=} \bigcup_{k\geq 1} \mathbb{Q}_+^\star \{-1,1\}^{\{k\}} \\
\mathscr{P}olyhedra &\stackrel{\text{def}}{=} \bigcup_{k\geq 1} (\mathbb{Q}^\star)^{\{k\}}
\end{aligned}$$

Définition 2. L'ensemble des *inégalités d'un domaine* de support $\mathscr{D}$ sur l'ensemble de variables $\mathcal{V}$ est défini par

$$\mathscr{D}_{\mathcal{V}} \stackrel{\text{def}}{=} \{\mathsf{True}, \mathsf{False}\} \cup \bigcup_{k\geq 1} \left\{ \sum_{i=1}^{k} a_i x_i \leq d \,\middle|\, \{a_1, \ldots, a_k\} \in \mathscr{D}, x_i \in \mathcal{V}, d \in \mathbb{Q} \right\}$$

avec $\mathsf{True} \stackrel{\text{def}}{=} 0 \leq 1$ et $\mathsf{False} \stackrel{\text{def}}{=} 0 \leq -1$.

Définition 3. L'ensemble des *valeurs d'un domaine* de support $\mathscr{D}$ sur un ensemble de variables $\mathcal{V}$ est défini par $\mathcal{D}_\mathcal{V} \stackrel{\text{def}}{=} \mathcal{P}^f(\mathscr{D}_\mathcal{V})$.

Définition 4. Si $c = \sum_{i=1}^{n} a_i x_i \leq d$, et si $c \not\equiv \mathsf{True}$ et $c \not\equiv \mathsf{False}$ alors la *concrétisation* de c est un demi-espace de $\mathbb{Q}^N$

$$[\![c]\!] \stackrel{\text{def}}{=} \left\{ (U_1, \ldots, U_N) \in \mathbb{Q}^N \;\middle|\; \sum_{i=1}^{n} a_i u_i \leq d, x_j = X_i \Rightarrow u_j = U_i \right\}$$

$$[\![\emptyset]\!] = [\![\{\mathsf{True}\}]\!] = [\![\mathsf{True}]\!] \stackrel{\text{def}}{=} \mathbb{Q}^N \qquad [\![\mathsf{False}]\!] \stackrel{\text{def}}{=} \emptyset$$

La concrétisation d'une valeur $C = \{c_1, \ldots, c_n\}$ est l'intersection de la concrétisation de ses éléments $[\![C]\!] \stackrel{\text{def}}{=} \bigcap_{i=1}^{n} [\![c_i]\!]$.

Définition 5. On définit une relation de préordre sur les valeurs de $\mathcal{D}_\mathcal{V}$ par $C_1 \vDash C_2$ si $[\![C_1]\!] \subseteq [\![C_2]\!]$. On définit également une relation d'équivalence $C_1 \equiv C_2$ si et seulement si $C_1 \vDash C_2$ et $C_2 \vDash C_1$, c'est-à-dire $[\![C_1]\!] = [\![C_2]\!]$.

Définition 6. On définit les *variables* d'une inégalité linéaire comme l'ensemble des variables pour lesquelles le coefficient est non nul. Si $c = a_0 X_0 + \ldots + a_N X_N \leq d$, alors $vars(c) \stackrel{\text{def}}{=} \{X_i \in \mathcal{V} \mid a_i \neq 0\}$. Pour un ensemble d'inégalités linéaires, on définit $vars(C) \stackrel{\text{def}}{=} \bigcup_{c \in C} vars(c)$.

Définition 7. Soit un ensemble de variables $\mathcal{X} \subseteq \mathcal{V}$, on définit la *restriction* d'une valeur d'un domaine sur cet ensemble de variables par :

$$\pi_\mathcal{X}(C) \stackrel{\text{def}}{=} \{c \in C \mid vars(c) \subseteq \mathcal{X}\}$$

Définition 8. Un *opérateur d'oubli* (ou encore *opérateur de projection*) de $\mathcal{D}_\mathcal{V}$ est une fonction $\exists : \mathcal{P}(\mathcal{V}) \longrightarrow {\mathcal{D}_\mathcal{V}}^{\mathcal{D}_\mathcal{V}}$ telle que $\forall C_1 \in \mathcal{D}_\mathcal{V}$, $\forall \mathcal{X} = \{X_{i_1}, \ldots, X_{i_n}\} \subseteq \mathcal{V}$, on a $\exists_\mathcal{X}(C_1) = C_2$ tel que $vars(C_2) \subseteq \mathcal{V} \setminus \mathcal{X}$ et

$$[\![C_2]\!] = \{(u_1, \ldots, u_{i_1 - 1}, a_1, u_{i_1 + 1}, \ldots, u_{i_n - 1}, a_n, u_{i_n + 1}, \ldots, u_N) \mid a_1, \ldots, a_n \in \mathbb{Q}, (u_1, \ldots, u_N) \in [\![C_1]\!]\}$$

Définition 9. On dit qu'un domaine $\mathcal{D}_\mathcal{V}$ est *stable par élimination de variable* si $\forall c_1, c_2 \in \mathcal{D}_\mathcal{V}$ tels que $c_1 = \sum_{i=1}^{N} a_i X_i \leq d_1$, $c_2 = \sum_{i=1}^{N} b_i X_i \leq d_2$ et que $\exists j$ tel que $1 \leq j \leq N$, $a_j < 0$ et $b_j > 0$, on a $(b_j c_1 + a_j c_2) \in \mathcal{D}_\mathcal{V}$.

Les domaines $\mathcal{I}ntervals$, $\mathcal{Z}ones$, $\mathcal{O}ctagons$, $\mathcal{L}ogahedra_\infty$, $\mathcal{T}VPI$ ainsi que $\mathcal{P}olyhedra$ sont stables par élimination de variable.

Si $|\mathcal{V}| \geq 3$ et $1 \leq B < \infty$ alors le domaine $\mathcal{L}ogahedra_B$ n'est pas stable par élimination de variable. En effet $X_1 - 2^B X_2 \leq 0$ et $X_2 - 2X_3 \leq 0$ sont dans $\mathcal{L}ogahedra_B$ mais leur somme normalisée $X_1 - 2^{B+1} X_3 \leq 0$ ne l'est pas.

Si $|\mathcal{V}| \geq 3$ alors le domaine $\mathcal{O}ctahedra$ n'est pas stable par élimination de variable. En effet $X_1 + X_2 \leq 0$ et $-X_1 + X_2 + X_3 \leq 0$ sont dans $\mathcal{O}ctahedra$ mais leur somme $2X_2 + X_3 \leq 0$ ne l'est pas.

Théorème 10. Si $\mathcal{D}_\mathcal{V}$ est stable par élimination de variable alors l'élimination de Fourier-Motzkin est un opérateur d'oubli.

Abstraction Sur $\mathbb{Q}$, il n'existe pas de meilleure abstraction, par exemple pour $X \times X \leq 2$. Nous appellerons $\bar{\alpha}$ l'abstraction partielle définie, lorsque cela a un sens, par la clôture topologique de l'enveloppe convexe.

Définition 11. L'*intersection* entre valeurs de $\mathcal{D}_\mathcal{V}$ est définie de manière exacte comme l'union des ensembles d'inégalités : $C_1 \sqcap^{\mathcal{D}_\mathcal{V}} C_2 \stackrel{\text{def}}{=} C_1 \cup C_2$.

L'*union* entre valeurs est définie comme étant la meilleure abstraction $C_1 \sqcup^{\mathcal{D}_\mathcal{V}} C_2 \stackrel{\text{def}}{=} \bar{\alpha}([\![C_1]\!] \cup [\![C_2]\!])$.

Théorème 12. L'ensemble $(\mathcal{D}_\mathcal{V}/\equiv, \vDash, \sqcup^{\mathcal{D}_\mathcal{V}}, \sqcap^{\mathcal{D}_\mathcal{V}}, \mathsf{False}, \mathsf{True})$ est un treillis.

3 Le foncteur de domaine

Nous allons construire un nouveau domaine numérique abstrait en nous basant sur un domaine numérique abstrait relationnel, parmi ceux existant comme les zones, les octogones, les logaèdres, les TVPI, les octaèdres ou les polyèdres, mais auxquels pourront venir s'ajouter à l'avenir d'autres domaines présentant les bonnes propriétés.

3.1 Propriétés du domaine sous-jacent

On suppose que le domaine de base est un domaine numérique abstrait sur $\mathbb{Q}$, c'est-à-dire la donnée de :

- un support $\mathscr{D} \ni \{-1, 1\}$, ainsi les zones sont inclues dans le domaine, en particulier il est possible de représenter des égalités,
- le domaine $\mathcal{D}_\mathcal{V}$ correspondant à ce support, ainsi qu'une représentation de ses éléments sur un ordinateur,
- un algorithme efficace de comparaison des éléments abstraits,
- une correspondance de Galois partielle $\mathcal{P}(\mathcal{V} \mapsto \mathbb{Q}) \underset{\bar{\alpha}}{\overset{[\![.]\!]}{\leftrightarrows}} \mathcal{D}_\mathcal{V}$,
- des algorithmes efficaces pour calculer : l'élimination de variable $\exists^{\mathcal{D}_\mathcal{V}}$ et l'intersection $\sqcap^{\mathcal{D}_\mathcal{V}}$ de façon exacte, une abstraction correcte de l'union $\sqcup^{\mathcal{D}_\mathcal{V}}$, des élargissements $\triangledown^{\mathcal{D}_\mathcal{V}}$ et éventuellement des rétrécissements $\triangle^{\mathcal{D}_\mathcal{V}}$.

3.2 Packs et graphes

Nous voulons restreindre le domaine $\mathcal{D}$ à certaines relations choisies au lieu de toutes les relations exprimables dans le domaine.

Définition 13. Un *ensemble de packs* est un ensemble de sous-ensembles de variables de $\mathcal{V}$, $P = \{P_1, \ldots, P_m \,|\, P_i \subseteq \mathcal{V}\}$ tel que $\bigcup_{i=1}^m P_i = \mathcal{V}$, et pour tous $i \neq j$, on a $P_i \not\subseteq P_j$.

Définition 14. On définit le *graphe des packs* par $\mathcal{G}_P \stackrel{\text{def}}{=} (P, F)$ où $(P_i, P_j) \in F$ si et seulement si $i \neq j$ et $P_i \cap P_j \neq \emptyset$. On appelera *frontières* ces $P_i \cap P_j$ non vides.

On demande de plus que le graphe des packs soit un arbre, c'est-à-dire qu'il existe exactement un chemin d'un pack à un autre dans ce graphe (notez qu'il est possible, sans trop de difficultés, d'étendre les opérations aux cas où le graphe est une forêt).

Ceci implique qu'une variable apparaît dans au plus 2 packs. Pour faire apparaître une variable dans 3 packs P_1, P_2, P_3 (dans le graphe des packs P_1—P_2—P_3), on en fera une copie dans P_2 et on maintiendra une contrainte d'égalité entre ces deux copies.

Dans la suite, nous utiliserons les variables $m \stackrel{\text{def}}{=} |P| \leq N$ pour désigner le nombre de packs, $p \stackrel{\text{def}}{=} \max_{1\leq i\leq m} |P_i| \leq N$ pour la taille maximale d'un pack, et $f \stackrel{\text{def}}{=} \max_{1\leq i<j\leq m} |P_i \cap P_j| \leq p$ pour la taille maximale d'une frontière.

3.3 Le foncteur

Nous appelons ce domaine TreeKs car, dans le cas d'un domaine 2-relationnel, le graphe des relations a la forme d'un arbre de graphes complets (habituellement notés K).

$$\mathscr{T}reeKs_P^{\mathscr{D}} \stackrel{\text{def}}{=} \{c \in \mathscr{D}_{\mathcal{V}} \,|\, \exists i, vars(c) \subseteq P_i\} \qquad \mathcal{T}reeKs_P^{\mathcal{D}} \stackrel{\text{def}}{=} \mathcal{P}^f(\mathscr{T}reeKs_P^{\mathscr{D}})$$

L'ensemble $(\mathcal{T}reeKs_P^{\mathcal{D}}/\equiv, \vDash, \sqcup^{\mathcal{T}reeKs_P^{\mathcal{D}}}, \sqcap^{\mathcal{D}_{\mathcal{V}}}, \mathsf{False}, \mathsf{True})$ est un trellis et on a la correspondance de Galois : $\mathcal{D}_{\mathcal{V}} \underset{\tilde{\alpha}}{\overset{\tilde{\gamma}}{\leftrightarrows}} \mathcal{T}reeKs_P^{\mathcal{D}}$.

Définition 15. On définit l'*hypergraphe des contraintes* d'une valeur $C = \{c_1, \ldots, c_n\}$ de $\mathcal{T}reeKs_P^{\mathcal{D}}$ par $\mathcal{H}_P(C) \stackrel{\text{def}}{=} (\mathcal{V}, E, \ell)$ où les hyperarêtes non orientées sont $E \stackrel{\text{def}}{=} \bigcup_{i=1}^{m} \{\mathcal{X} \subseteq P_i \,|\, \exists\, c \in \mathcal{D}_{\mathcal{V}}, vars(c) = \mathcal{X}\}$ et ℓ est un étiquetage des hyperarêtes $\ell(e) \stackrel{\text{def}}{=} \{c \in C \,|\, vars(c) = e\}$.

3.4 Le foncteur de représentation

La représentation précédente des valeurs convient bien aux domaines où la représentation est creuse et où les contraintes sont stockées dans les arêtes d'un graphe, comme pour le domaine des TVPI. Mais, généralement, on conservera la représentation des contraintes du domaine sous-jacent, en associant à chaque pack une valeur abstraite.

Par exemple, pour les octogones nous préférerons garder la représentation originelle [22] où chaque variable utilise deux sommets et les contraintes sont stockées dans une demi-matrice d'adjacence.

Les relations aux frontières apparaitront donc deux fois, notre représentation sera redondante mais plus efficace à utiliser dans les algorithmes.

Nous définissons ce nouveau domaine comme le produit cartésien du domaine sous-jacent où tous les $\bot$ ont été fusionnés en un seul :

$$\mathsf{D}_P \stackrel{\text{def}}{=} (\mathsf{D}_{P_1} \setminus \{\bot^{D_{P_1}}\}) \times \ldots \times (\mathsf{D}_{P_m} \setminus \{\bot^{D_{P_m}}\}) \cup \{\bot_{\mathsf{D}_P}\}$$

L'ensemble $(\mathsf{D}_P/\equiv_{\mathsf{D}_P}, \sqsubseteq_{\mathsf{D}_P}, \sqcup_{\mathsf{D}_P}, \sqcap_{\mathsf{D}_P}, \bot_{\mathsf{D}_P}, \top_{\mathsf{D}_P})$ où

$\mathbf{v} \sqsubseteq_{\mathsf{D}_P} \mathbf{w} \overset{\text{def}}{\iff} \forall i, v_i \sqsubseteq_{\mathsf{D}} w_i$ $\qquad (\top_{\mathsf{D}_P})_i \overset{\text{def}}{=} \top_{\mathsf{D}}$

$\mathbf{v} \equiv_{\mathsf{D}_P} \mathbf{w} \overset{\text{def}}{\iff} \mathbf{v} \sqsubseteq_{\mathsf{D}_P} \mathbf{w} \sqsubseteq_{\mathsf{D}_P} \mathbf{v}$ $\qquad \bot_{\mathsf{D}_P} \sqsubseteq_{\mathsf{D}_P} \mathbf{x}$

$(\mathbf{v} \sqcup_{\mathsf{D}_P} \mathbf{w})_i \overset{\text{def}}{=} v_i \sqcup_{\mathsf{D}} w_i$ $\qquad \bot_{\mathsf{D}_P} \sqcup_{\mathsf{D}_P} \mathbf{x} \overset{\text{def}}{=} \mathbf{x} \sqcup_{\mathsf{D}_P} \bot_{\mathsf{D}_P} \overset{\text{def}}{=} \mathbf{x}$

$(\mathbf{v} \sqcap_{\mathsf{D}_P} \mathbf{w})_i \overset{\text{def}}{=} v_i \sqcap_{\mathsf{D}} w_i$ $\qquad \bot_{\mathsf{D}_P} \sqcap_{\mathsf{D}_P} \mathbf{x} \overset{\text{def}}{=} \mathbf{x} \sqcap_{\mathsf{D}_P} \bot_{\mathsf{D}_P} \overset{\text{def}}{=} \bot_{\mathsf{D}_P}$

est un treillis et on a la correspondance de Galois : $\mathcal{T}reeKs_P^{\mathcal{D}} \underset{\ddot{\alpha}}{\overset{\ddot{\gamma}}{\leftrightarrows}} \mathsf{D}_P$.

Définition 16. Comme cette représentation est redondante, nous dirons qu'une valeur est *cohérente* si et seulement si pour tous i, j, on a $\pi_{P_i \cap P_j}(v_i) \equiv \pi_{P_i \cap P_j}(v_j)$ (ceci ne prend vraiment de sens qu'au niveau des frontières).

Théorème 17. Si $\exists^{\mathcal{D}}$ et $\cap^{\mathcal{D}}$ sont exactes alors pour toute valeur $\mathbf{v}$, il est possible de construire une valeur cohérente $coh(\mathbf{v})$ telle que $\mathbf{v} \equiv coh(\mathbf{v})$.

En effet, il suffit de faire pour tous i, j, $v_i \leftarrow v_i \cap^{\mathcal{D}} \exists^{\mathcal{D}}_{P_i \setminus P_j}(v_j)$.

3.5 Complétion

Le but de l'opération de complétion est de rendre explicite les relations implicites. C'est une opération nécessaire à la plupart des autres opérations, sa complexité détermine l'efficacité du domaine.

Dans le cas de TreeKs, la complétion a but supplémentaire : celui de transférer les informations entre les différents packs.

Définition 18. On dit que C est $\mathscr{D}$*-complet* si pour tout $c \in \mathscr{D}_{\mathcal{V}}$, $C \vDash c$ implique $\pi_{vars(c)}(C) \vDash c$.

Définition 19. On dit qu'un domaine $\mathcal{D}$ est *complétable* si pour toute valeur C de $\mathcal{D}_{\mathcal{V}}$, il existe $C' \in \mathcal{D}_{\mathcal{V}}$ tel $C' \equiv C$ et C' est $\mathscr{D}$-complet.

Si $\mathcal{D}$ est complétable, on notera $\mathcal{D}'$ l'ensemble de ses valeurs $\mathscr{D}$-complètes. Si $\mathcal{D}$ possède une opération de complétion, on notera `complète` cette fonction. Dans les autres cas, on prendra $\mathcal{D}'$ égal à $\mathcal{D}$ et `complète` sera l'identité.

Un moyen simple de compléter une valeur $V = (C_1, \ldots, C_m) \in \mathsf{D}_P$ est d'utiliser la complétion de $\mathcal{D}_{\mathcal{V}}$. On pose $V^\star = \texttt{complète}^{\mathcal{D}_{\mathcal{V}}}(C_1 \cup \ldots \cup C_m)$. Alors $V' = (\pi_{P_1}(V^\star), \ldots, \pi_{P_m}(V^\star)) \in \mathsf{D}'_P$ et $V' \equiv V$. Mais on perd tout l'intérêt de la restriction à un sous-graphe de relations.

Une complétion pack à pack est cependant insuffisante. Supposons que $(C_1, C_2) \in \mathcal{Z}ones_{\{P_1,P_2\}}, P_1 = \{x, y, z\}, P_2 = \{y, z, t\}, C_1 = \{x \leq y, z \leq x\}$ et $C_2 = \{y \leq t\}$. $\texttt{complète}(C_1) = \{x \leq y, z \leq x, z \leq y\}$ et $\texttt{complète}(C_2) = \{y \leq t\}$. Alors que $\texttt{complète}(C_1 \cup C_2) = \{x \leq y, y \leq t, z \leq x, z \leq y, z \leq t\}$. La complétion de P_1 fournit des informations à injecter dans P_2 et vice versa.

Le théorème suivant, dont la preuve est basée sur le lemme de Farkas, montre que les échanges entre packs restent toutefois limités.

Théorème 20. Soient $C \in \mathcal{D}'_{\mathcal{V}}$, $\mathcal{X} \subseteq \mathcal{V}$, $C^+ \in \mathcal{D}_{\mathcal{X}}$ tel que $(\pi_{\mathcal{X}}(C) \cup C^+) \in \mathcal{D}'_{\mathcal{X}}$, et enfin $C' \in \mathcal{D}'_{\mathcal{V}}$ tel que $C' \equiv (C \cup C^+)$. Alors $\forall\ \mathcal{Y} \subseteq \mathcal{X}$, $\pi_{\mathcal{Y}}(C') \equiv \pi_{\mathcal{Y}}(C \cup C^+)$.

Ce théorème signifie que si l'on ajoute à une valeur complète des nouvelles contraintes complètes sur un sous-ensemble et que le tout est complété à nouveau alors les contraintes du sous-ensemble ne peuvent pas être améliorées. L'algorithme suivant utilise ce théorème pour construire efficacement une complétion.

Algorithme de complétion On choisit arbitrairement une racine dans l'arbre des packs et on oriente cet arbre de sorte que les arcs soient dirigés de la racine vers les feuilles. Quitte à réordonner les packs, on supposera que la racine est P_1 et que s'il y a un arc de P_i vers P_j alors $i < j$. $\texttt{père}(i)$ désignera le pack père du pack P_i dans cet arbre orienté (indéfini pour P_1).

Fonction $\texttt{complète}^{\mathsf{D}_P}(C_1, \ldots, C_m)$

pour $i \leftarrow m$ **à** 2 **faire** {des feuilles vers la racine}
  $C_i \leftarrow \texttt{complète}^{\mathcal{D}_{P_i}}(C_i)$
  si $C_i = \bot^{\mathcal{D}_{P_i}}$ **alors retourner** $\bot^{\mathsf{D}_P}$
  $C_{\texttt{père}(i)} \leftarrow C_{\texttt{père}(i)} \cup \pi_{P_{\texttt{père}(i)}}(C_i)$
$C_1 \leftarrow \texttt{complète}^{\mathcal{D}_{P_1}}(C_1)$
si $C_1 = \bot^{\mathcal{D}_{P_1}}$ **alors retourner** $\bot^{\mathsf{D}_P}$
pour $i \leftarrow 2$ **à** m **faire** {de la racine vers les feuilles}
  $C_i \leftarrow C_i \cup \pi_{P_i}(C_{\texttt{père}(i)})$
  $C_i \leftarrow \texttt{complète}^{\mathcal{D}_{P_i}}(C_i)$
  si $C_i = \bot^{\mathcal{D}_{P_i}}$ **alors retourner** $\bot^{\mathsf{D}_P}$
retourner $(C_1, \ldots, C_m)$

Complexité Si A_p est le coût de la complétion d'un pack de taille p et $B_{p,f}$ est le coût de la projection et de l'union d'un pack de taille $\leq p$ sur un pack de taille $\leq p$, avec une frontière de taille f, alors le coût $C_{p,f}$ de l'algorithme de complétion est borné par :

$$A_{|P_1|} + 2\sum_{i=2}^{m} A_{|P_i|} + 2\sum_{i=2}^{m} B_{max(|P_i|,|P_{\texttt{père}(i)}|),|P_i \cap P_{\texttt{père}(i)}|} \leq 2m(A_p + B_{p,f})$$

Pour une taille de packs fixée, notre algorithme de complétion a une complexité linéaire en le nombre de variables, et ce quel que soit le domaine sous-jacent. Si ce dernier propose une complétion incrémentale [13], alors celle-ci pourra remplacer la complétion globale dans la seconde boucle. Notre algorithme s'en trouvera accéléré mais sa complexité restera inchangée.

3.6 Opérateurs sur les ensembles

De manière générale, les opérateurs sur D_P seront définis à partir des opérateurs sur $\mathcal{D}$. Cependant, dans les opérateurs sur $\mathcal{D}$, chaque fois qu'une

complétion est nécessaire, c'est en fait notre complétion qui devra être utilisée. On devra donc s'assurer de la complétion des arguments avant d'appeler ces opérateurs — on considérera que c'est le cas dans la suite.

Les *tests d'inclusion et d'égalité* ne posent pas de problème et se font point-à-point sur des arguments complets. S'ils sont exacts sur $\mathcal{D}$ alors ils le sont aussi sur D_P.

L'*intersection* étant exacte (il s'agit d'une union de contraintes), elle est étendue point-à-point sur chaque pack et reste exacte.

Théorème 21. Si $\sqcup^{\mathcal{D}}$ est la meilleure abstraction de l'union dans $\mathcal{D}$ alors $\sqcup^{\mathsf{D}_P}$, extension point-à-point de $\sqcup^{\mathcal{D}}$ sur chaque pack de P, est la meilleure abstraction de l'union dans D_P.

3.7 Fonctions de transfert

Opérateur d'oubli *Oublier* des variables (c'est-à-dire projeter sur un espace qui ne contient pas ces variables) se fait facilement à partir d'une valeur complète en oubliant toutes les contraintes qui concernent ces variables.

Extraction de contraintes On part d'une valeur $(C_1, \ldots, C_m)$ complète. Supposons que l'on veuille extraire l'ensemble des contraintes existantes entre les variables d'un ensemble $\mathcal{X} \subseteq \mathcal{V}$, $1 \leq |\mathcal{X}| \leq N$. Si toutes ces variables sont dans le même pack (c'est notamment le cas si $|\mathcal{X}| = 1$ pour l'extraction d'intervalles) alors une simple projection suffit. Dans le cas contraire, les choses se compliquent. Pour chaque pack P_i, posons $\mathcal{X}_{(i)} \stackrel{\text{def}}{=} \mathcal{X} \cap P_i$. Quitte à changer de racine, on supposera que $\mathcal{X}_{(1)} \neq \emptyset$.

Fonction $\texttt{extrait}^{\mathsf{D}_P}(C_1, \ldots, C_m, \mathcal{X})$

$V_{1..m} \leftarrow \emptyset$
$D_{1..m} \leftarrow \emptyset$
pour $i \leftarrow m$ **à** 2 **faire**
  $V_i \leftarrow V_i \cup \mathcal{X}_{(i)}$
  si $V_i \neq \emptyset$ **alors**
    $V_{\texttt{père}(i)} \leftarrow V_{\texttt{père}(i)} \cup V_i$
    $D_{\texttt{père}(i)} \leftarrow D_{\texttt{père}(i)} \cup \pi_{V_i \cup (P_i \cap P_{\texttt{père}(i)})}(\texttt{complète}^{\mathcal{D}_{(P_i \cup V_i)}}(C_i \cup D_i))$
retourner $\pi_{\mathcal{X}}(\texttt{complète}^{\mathcal{D}_{(P_1 \cup \mathcal{X})}}(C_1 \cup D_1))$

Cette fonction a un coût borné par $D_{p,f} = m(A_{p+|\mathcal{X}|} + B_{(p+|\mathcal{X}|),(f+|\mathcal{X}|)})$. Ainsi, pour une taille de packs fixée et un nombre de variables d'intérêt borné, cette fonction a une complexité linéaire en le nombre total de variables. Si $|\mathcal{X}| = 2$ alors on peut même borner le coût par $D'_{p,f} = d(A_{p+2} + B_{(p+2),(f+2)})$, coût linéaire en le diamètre de l'arbre des packs.

Ajout de contraintes Si toutes les variables de la contrainte à ajouter sont dans un unique pack alors on peut ajouter la contrainte dans ce pack

uniquement. Dans le cas contraire, il faut extraire de cette contrainte d'autres contraintes que l'on peut ajouter aux packs de manière indépendante. L'algorithme suivant ajoute à une valeur $(C_1, \ldots, C_m)$ complète toutes les informations exprimables dans $\mathcal{T}reeKs^{\mathcal{D}}$ que l'on peut tirer de la contrainte c. On gardera les mêmes notations, mais cette fois $\mathcal{X} = vars(c)$.

Fonction $\texttt{ajout}^{\mathcal{D}_P}(C_1, \ldots, C_m, c)$

$V_{1..m} \leftarrow \emptyset$
$D_{1..m} \leftarrow \emptyset$
pour $i \leftarrow m$ **à** 2 **faire**
  $V_i \leftarrow V_i \cup \mathcal{X}_{(i)}$
  si $V_i \neq \emptyset$ **alors**
    $V_i \leftarrow V_i \cup (P_i \cap P_{\texttt{père}(i)})$
    $V_{\texttt{père}(i)} \leftarrow V_{\texttt{père}(i)} \cup V_i$
    $D_{\texttt{père}(i)} \leftarrow D_{\texttt{père}(i)} \cup \pi_{V_i}(\texttt{complète}^{\mathcal{D}_{(P_i \cup V_i)}}(C_i \cup D_i))$
$D_0 \leftarrow \pi_{V_1}(\texttt{complète}^{\mathcal{D}_{(P_1 \cup V_1)}}(C_1 \cup D_1))$
pour $i \leftarrow 1$ **à** m **faire**
  si $V_i \neq \emptyset$ **alors**
    $C_i \leftarrow C_i \cup \pi_{P_i}(D_0)$
retourner $(C_1, \ldots, C_m)$

Cette fonction a un coût en pire cas borné par $E_{p,f} = m(A_{mf+p+|\mathcal{X}|} + B_{mf+p+|\mathcal{X}|})$. Si $|\mathcal{X}| = 2$ alors on ce coût est généralement linéaire en le diamètre du graphe, on peut le borner par $E'_{p,f} = d(A_{df+p} + B_{df+p})$. On essaiera donc d'éviter l'ajout de contraintes entre variables éloignées.

3.8 Élargissement, rétrécissement

L'élargissement peut s'opérer point-à-point en appliquant l'élargissement du domaine sous-jacent à chaque pack indépendamment. La convergence en temps fini des chaînes croissantes est donc immédiatement assurée. Cependant les valeurs ainsi formées peuvent ne pas être cohérentes, ni complètes. Il ne faut néanmoins pas chercher à les rendre cohérentes ou à les compléter car cela peut mettre en danger la convergence [22]. En effet l'élargissement consiste à relâcher des contraintes vers $+\infty$ alors que la complétion a un but inverse, le problème est le même pour la cohérence car elle s'obtient par intersection.

Si le domaine sous-jacent possède un rétrécissement alors on peut également obtenir un rétrécissement en l'appliquant pack à pack indépendamment.

4 Application aux octogones : $\mathcal{T}reeKs^{\mathcal{O}ctagons}$

Nous montrons ici comment utiliser notre foncteur sur le domaine des octogones et donnons des algorithmes optimisés pour ce cas précis.

Tout d'abord, rappelons que le domaine des octogones [22] permet d'exprimer toute conjonction de contraintes de la forme $\pm X_i \pm X_j \leq d$. Un octogone sur $\mathcal{V}$ peut être représenté par une DBM $\mathbf{m}$ de dimension $2N$, c'est-à-dire une matrice $2N \times 2N$ sur $\overline{\mathbb{Q}} \stackrel{\text{def}}{=} \mathbb{Q} \cup \{+\infty\}$, ou de manière équivalente un graphe de potentiel avec des noeuds dans $\mathcal{V}' \stackrel{\text{def}}{=} \{X'_1, X'_2, \ldots, X'_{2N-1}, X'_{2N}\} \stackrel{\text{def}}{=} \{X_1^+, X_1^-, \ldots, X_N^+, X_N^-\}$, traduisant pour tous $1 \leq i, j \leq 2N$ la contrainte $V'_j - V'_i \leq \mathbf{m}_{ij}$. On définit $\bar{i} \stackrel{\text{def}}{=} i + 1$ si i est impair et $\bar{i} \stackrel{\text{def}}{=} i - 1$ si i est pair.

Les contraintes d'intervalles $X_i \in [a, b]$ sont codées sous la forme $X_i^- - X_i^+ \leq -2a$ et $X_i^+ - X_i^- \leq 2b$, c'est-à-dire $X'_{2i} - X'_{2i-1} \leq -2a$ et $X'_{2i-1} - X'_{2i} \leq 2b$. Toutes les autres contraintes ont une double représentation : $X'_i - X'_j \leq d$ et $X'_{\bar{j}} - X'_{\bar{i}} \leq d$, par exemple $X_i^+ - X_j^- \leq d$ et $X_j^+ - X_i^- \leq d$ pour $X_i + X_j \leq d$.

On rappelle également que la complétion d'un octogone se fait par une phase de clôture de plus courts chemins (qui assure que $\forall\, 1 \leq i, j, k \leq 2N$, $\mathbf{m}_{ij} \leq \mathbf{m}_{ik} + \mathbf{m}_{kj}$), puis par une phase de renforcement [1] qui assure que $\forall\, 1 \leq i, j \leq 2N, \mathbf{m}_{ij} \leq \frac{1}{2}(\mathbf{m}_{i\bar{i}} + \mathbf{m}_{\bar{j}j})$.

L'application de TreeKs au domaine des octogones donne le domaine :

$$\mathsf{Octagons}_P \stackrel{\text{def}}{=} (\mathsf{DBM}_{P_1} \times \ldots \times \mathsf{DBM}_{P_m}) \cup \{\bot_{\mathsf{Octagons}_P}\}$$

4.1 Complétion amélioriée

Pour les octogones, nous avons $A_p = O(p^3)$ et $B_{p,f} = O(f^2)$. La complétion a donc un coût $O(mp^3) = O(Np^2)$, qui, pour une taille de pack fixée, est linéaire en le nombre de variables. Plus précisément [1], $A_p = 16p^3 + 8p^2$ et $B_{p,f} = 4f^2 \leq 4p^2$. Soit un total $C_{p,f}^{\mathcal{O}ct} = 32mp^3 + 24mp^2$.

Une amélioration [1] consiste à n'utiliser dans un premier temps que la clôture de plus courts chemins (de coût $A'_p = 16p^3$) et seulement à la fin le renforcement (de coût $R_p = 8p^2$), mais le gain est négligeable. Par contre si l'on utilise dans la seconde boucle une clôture de plus courts chemins incrémentale (de coût $A_p^{inc} = 16f(p^2 - f^2)$) présentée ci-dessous alors le coût devient $C_{p,f}^{\mathcal{O}ct} = m(A'_p + A_{p,f}^{inc} + 2B_{p,f} + R_p) = 16mp^3 + 16mf(p^2 - f^2) + 16mp^2$.

Soit un gain de 30% (pour des frontières de taille proche de $p/2$) à 50% (pour de très petites frontières) !

Clôture de plus courts chemins incrémentale

```
pour i ← 1 à 2N − 2f faire
    pour j ← 2N − 2f + 1 à 2N faire
        pour k ← 2N − 2f + 1 à 2N faire
            m_ij ← min(m_ij, m_ik + m_kj)
            m_ji ← min(m_ji, m_jk + m_ki)
    pour j ← 1 à 2N − 2f faire
        pour k ← 2N − 2f + 1 à 2N faire
            m_ij ← min(m_ij, m_ik + m_kj)
```

Cet algorithme s'applique à une matrice $2N \times 2N$ close (pour la clôture des plus courts chemins) dont le carré $2f \times 2f$ droit inférieur a été amélioré (c'est-à-dire est point-à-point inférieur) et clos indépendamment.

4.2 Extraction de contraintes

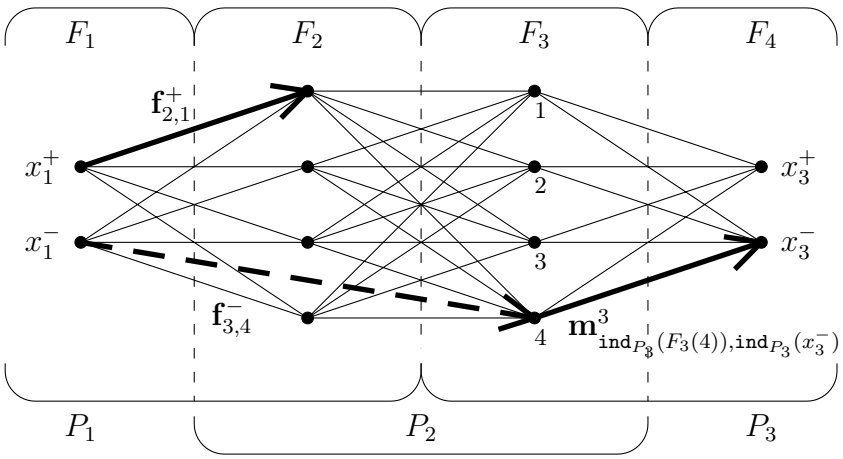
Supposons que l'on parte d'une valeur complète $\mathbf{o} = (\mathbf{m}^1, \ldots, \mathbf{m}^m)$ sur $\mathsf{Octagons}_P$ et que l'on veuille extraire toutes les contraintes concernant les deux variables x_1 et $x_q \in \mathcal{V}$, c'est-à-dire les bornes de $x_1 - x_q$ et de $x_1 + x_q$. Si les deux variables sont dans un même pack alors le résultat est immédiat. Sinon, quitte à renommer les packs, on supposera que $x_1 \in P_1$ et $x_q \in P_q$ et que le chemin de P_1 à P_q dans $\mathcal{G}_P$ est formé des packs $P_1, P_2, \ldots, P_{q-1}, P_q$ ($q \leq d$, le diamètre du graphe). On suppose donnée une fonction $\mathtt{ind}_{P_i}(x)$ qui donne l'indice de la variable x dans le pack P_i, et on note $F_1 = \{x_1^+, x_1^-\}$, $F_i = P_i \cap P_{i+1}$ si $2 \leq i \leq q$ et $F_{q+1} = \{x_q^+, x_q^-\}$ les frontières.

L'extraction de contraintes utilise la fonction auxiliaire suivante :

Fonction `plusCourtsCheminsAuxFrontières`($\mathbf{m}^1$, …, $\mathbf{m}^m$, x_1, x_q)

$\mathbf{f}^+_{1,1} \leftarrow 0 \quad \mathbf{f}^+_{1,2} \leftarrow +\infty$ { $\mathbf{f}^\pm_{i,j}$ représente le plus court chemin de
$\mathbf{f}^-_{1,2} \leftarrow 0 \quad \mathbf{f}^-_{1,1} \leftarrow +\infty$ $x_1^\pm$ à la j^{e} variable de la frontière F_i }
pour $i \leftarrow 1$ **à** q **faire**
  pour $j \leftarrow 1$ **à** $|F_{i+1}|$ **faire**
    $\mathbf{f}^+_{(i+1),j} \leftarrow \min_{1 \leq k \leq |F_i|} \mathbf{f}^+_{i,k} + \mathbf{m}^i_{\mathtt{ind}_{P_i}(F_i(k)),\mathtt{ind}_{P_i}(F_{i+1}(j))}$
    $\mathbf{f}^-_{(i+1),j} \leftarrow \min_{1 \leq k \leq |F_i|} \mathbf{f}^-_{i,k} + \mathbf{m}^i_{\mathtt{ind}_{P_i}(F_i(k)),\mathtt{ind}_{P_i}(F_{i+1}(j))}$
retourner f

En résumé, il s'agit d'un algorithme dynamique qui calcule les plus courts chemins de x_1^+ et x_1^- à chacune des variables des frontières qui séparent le pack P_1 du pack P_q. La complexité en pire cas de cet algorithme est $O(df^2)$. L'illustration ci-dessous montre un exemple avec $q = 3$ et $f = 2$. Les étiquettes ($\mathbf{f}^+_{2,1}$, $\mathbf{f}^-_{3,4}$, et $\mathbf{m}^3_{\mathtt{ind}_{P_3}(F_3(4)),\mathtt{ind}_{P_3}(x_3^-)}$) font référence aux flèches en gras.



L'extraction de contrainte a donc la même complexité et s'exprime simplement :

Fonction $\texttt{extrait}^{\mathsf{Octagons}_P}(\mathbf{m}^1, \ldots, \mathbf{m}^m, x_1, x_q)$

$\mathbf{f} \leftarrow \texttt{plusCourtsCheminsAuxFrontières}(\mathbf{m}^1, \ldots, \mathbf{m}^m, x_1, x_q)$
$\mathbf{b}_i^+ \leftarrow \frac{1}{2} m^i_{\texttt{ind}_{P_i}(x_i),\texttt{ind}_{P_i}(-x_i)} \qquad \mathbf{b}_i^- \leftarrow \frac{1}{2} m^i_{\texttt{ind}_{P_i}(-x_i),\texttt{ind}_{P_i}(x_i)}$
retourner $\begin{cases} -\mathbf{f}^-_{q+1,2} \leq x_q - x_1 \leq \mathbf{f}^+_{q+1,1} & -\mathbf{b}_1^- \leq x_1 \leq \mathbf{b}_1^+ \\ -\mathbf{f}^+_{q+1,2} \leq x_q + x_1 \leq \mathbf{f}^-_{q+1,1} & -\mathbf{b}_q^- \leq x_q \leq \mathbf{b}_q^+ \end{cases}$

4.3 Ajout de contraintes

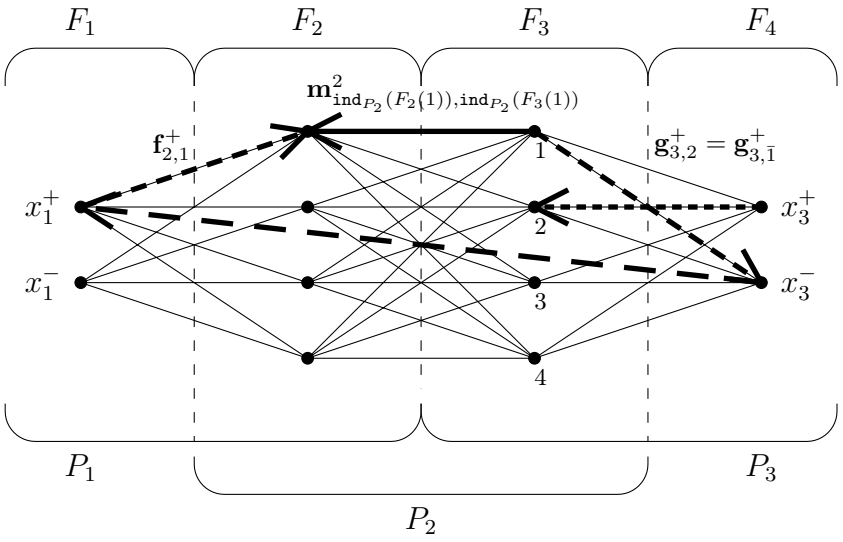
Encore une fois, il n'y a de problème que lorsque les variables ne sont pas dans un même pack. Lorsque l'on part d'une valeur complète, ajouter une contrainte a le même coût $O(df^2)$ qu'extraire une contrainte. Nous supposerons que c'est la contrainte $x_1 + x_q \leq d$ qui est ajoutée, mais tous les cas se traitent de la même façon. Rappelons que cette contrainte correspond à la fois à $x_1^+ - x_q^- \leq d$ et à $x_q^+ - x_1^- \leq d$.

Fonction $\texttt{ajout}^{\mathsf{Octagons}_P}(\mathbf{m}^1, \ldots, \mathbf{m}^m, x_1^+, x_q^-, d)$

$\mathbf{f} \leftarrow \texttt{plusCourtsCheminsAuxFrontières}(\mathbf{m}^1, \ldots, \mathbf{m}^m, x_1, x_q)$
$\mathbf{g} \leftarrow \texttt{plusCourtsCheminsAuxFrontières}(\mathbf{m}^1, \ldots, \mathbf{m}^m, x_q, x_1)$
pour $i \leftarrow 1$ **à** q **faire**
  pour $j \leftarrow 1$ **à** $|F_i|$ **faire**
    pour $k \leftarrow 1$ **à** $|F_{i+1}|$ **faire**
      $\mathbf{m}^i_{\texttt{ind}_{P_i}(F_{i+1}(k)),\texttt{ind}_{P_i}(F_i(j))} \leftarrow \min(\mathbf{m}^i_{\texttt{ind}_{P_i}(F_{i+1}(k)),\texttt{ind}_{P_i}(F_i(j))}, \mathbf{g}^+_{i+1,\bar{k}} + d + \mathbf{f}^+_{i,j})$
      $\mathbf{m}^i_{\texttt{ind}_{P_i}(F_i(j)),\texttt{ind}_{P_i}(F_{i+1}(k))} \leftarrow \min(\mathbf{m}^i_{\texttt{ind}_{P_i}(F_i(j)),\texttt{ind}_{P_i}(F_{i+1}(k))}, \mathbf{g}^-_{i+1,k} + d + \mathbf{f}^-_{i,\bar{j}})$
retourner $(\mathbf{m}^1, \ldots, \mathbf{m}^m)$

Cet algorithme projète sur tous les arcs, entre deux frontières successives dans le chemin de P_1 à P_q, la meilleure contrainte qui peut être déduite de l'ajout de la contrainte entre x_1^+ et x_q^-.

L'illustration ci-dessous montre un exemple avec $q = 3$ et $f = 2$, mais un seul sens est représenté. La flèche de x_3^- à x_1^+ indique la contrainte que l'on souhaite ajouter. La flèche de x_3^+ à $F_3(2)$ représente la même contrainte que la flèche de $F_3(1)$ à x_3^-.

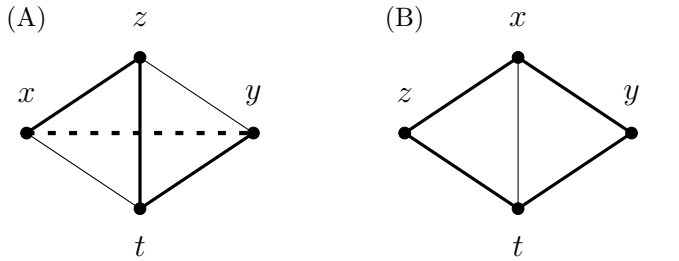


4.4 Élargissement

Au lieu de l'élargissement syntaxique de [22], [1] propose un élargissement sémantique basé sur l'élargissement standard des polyèdres. Il prétend qu'un tel élargissement s'applique également aux autres domaines d'inégalités linéaires et en donne une application pour les octogones. Comme cet élargissement est basé sur une réduction des plus courts chemins, il s'adapte également aux octogones restreints à un sous-graphe.

4.5 Exemple de packs

Reprenons l'exemple 2 page 3. Dans cet exemple (B), les contraintes que l'on possédait étaient toutes exprimables avec la répartition donnée des packs, donc il n'y a pas eu de perte de précision.



Si maintenant les packs sont comme en (A) et que l'analyseur génère la séquence de relations suivantes (arêtes en gras sur les diagrammes) : $c_1 = t - z \leq 2$, $c_2 = x + z \leq 4$, $c_3 = z - x \leq 2$, $c_4 = x - y \leq 2$ et $c_5 = y - t \leq 2$, alors, dans un premier temps, les trois premières contraintes sont simplement

ajoutées (et donnent par complétion $t + x \leq 6$, $t - x \leq 4$, $t + z \leq 8$, $z \leq 3$, $t \leq 5$). Lors de l'ajout de la quatrième contrainte, celle-ci n'est pas conservée mais permet tout de même d'ajouter les contraintes $z - y \leq 4$ et $t - y \leq 6$. La cinquième contrainte ajoute $y - z \leq 4$, $y + z \leq 10$, $y \leq 7$, $y + t \leq 12$ et $z - t \leq 6$. Mais sans la possibilité d'exprimer une relation entre x et y, il est impossible de donner une borne sur x !

Notre domaine est moins précis que les octogones, mais seulement dans les cas où les contraintes utiles ne sont pas exprimables. Il peut donc se révéler aussi précis que les octogones dans de nombreux cas, mais avec une vitesse bien supérieure.

5 Conclusion

5.1 Conclusion

Nous avons développé un nouveau domaine numérique abstrait sous la forme d'un foncteur. Ce nouveau domaine se base sur n'importe quel domaine numérique qui rentre dans le cadre des domaines d'inégalités linéaires que nous avons défini, qui inclut un certain nombre de domaines existants, notamment les domaines faiblement relationnels comme les octogones ou les TVPI, mais reste également ouvert à de futurs domaines.

Ce domaine est fondé sur une restriction de la forme et du nombre des relations exprimables afin d'accélérer les opérations du domaine tout en assurant que ces opérations restent les meilleures possibles.

Nous avons fourni des algorithmes prouvés corrects et efficaces pour les différentes opérations du domaine. Ces algorithmes, et notamment l'algorithme de complétion, se basent sur les fonctions du domaine sous-jacent mais, grâce à la réduction du nombre de contraintes, possèdent une complexité linéaire en le nombre de variables, permettant ainsi l'utilisation de ce domaine dans l'analyse de programmes de très grande taille.

Une implémentation de notre domaine est en cours au sein de la bibliothèque de domaines numériques Apron [16], utilisée dans l'analyseur Interproc [18]. Cependant notre implémentation demande certaines modifications importantes à la fois dans Apron et Interproc car ni l'un ni l'autre n'étaient prévus pour accueillir un tel domaine sous forme de foncteur, ni même de domaine paramétrable par une répartition des variables en packs.

5.2 Extensions

Stratégies de génération des packs Un point que nous n'avons pas abordé est la façon d'organiser les variables du programme en différents packs. D'un côté il y a des impératifs de coût que sont la restriction de la taille des packs, pour assurer une complétion en temps linéaire, mais aussi celle de la taille des frontières et du diamètre du graphe, lors de l'ajout de

contraintes. D'un autre côté la précision du domaine dépend beaucoup de ces paramètres. Plus les packs sont grands, plus il y a d'informations. Et ces informations sont d'autant mieux propagées que les frontières sont grandes. Le diamètre de l'arbre de packs peut par contre toujours être restreint à 3.

À ces contraintes de forme sur le graphe s'ajoutent des contraintes d'expressivité. Les variables qui forment des relations utiles doivent se trouver dans le même pack ou être proches. Des stratégies de génération de packs se basant sur des relations syntaxiques ont déjà été développées pour des packs non reliés [7, 26] et peut constituer une première approche à la formation des packs dans notre domaine.

Comme le propose [26], l'idéal sera peut-être à terme d'utiliser des packs dynamiques qui évoluent au cours de l'analyse en fonction des relations qui apparaissent.

Autres supports Il est légitime de se demander s'il est possible de former de nouveaux supports de domaines numériques effectifs. Par exemple, en fixant la dimension à 3 nous pouvons obtenir des octaèdres 3D. Cependant ce support ne permet pas d'effectuer une complétion par propagation d'un petit nombre de contraintes. Les contraintes $c_1 = x_1 + x_2 \leq 0$, $c_2 = x_3 + x_4 \leq 0$, $c_3 = x_2 + x_3 - x_4 \leq 3$, $c_4 = x_1 - x_4 \leq 0$ n'impliquent par somme de deux ou trois contraintes que $c_2 + c_4 = x_1 + x_3 \leq 0$ alors qu'une complétion donnerait aussi $2c_1 + 2c_2 + c_3 + c_4 = x_1 + x_2 + x_3 \leq 1$. La recherche de nouveaux supports entre les zones et les polyèdres permettra d'affiner le curseur précision/coût, et tout nouveau support pourra efficacement être utilisé grâce à TreeKs.

Il serait également intéressant d'adapter notre foncteur à d'autres types de domaines, en commençant par des domaines convexes, comme les ellipsoïdes [10]. En effet, les domaines convexes peuvent être considérés comme des domaines d'inégalités linéaires généralisés, où l'on autorise un nombre infini de contraintes par valeur. L'adaptation à des domaines non convexes demandera certainement plus de travail et ne pourra probablement pas être généralisée mais reste envisageable car c'est avant tout une méthode efficace pour faire de tout domaine un domaine *scalable*.

Bibliographie

[1] Bagnara, R., Hill, P. M., Mazzi, E., and Zaffanella, E. Widening operators for weakly-relational numeric abstractions. In *SAS'05* (2005), Springer-Verlag.

[2] Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X. A static analyzer for large safety-critical software. In *PLDI'03* (June 2003), ACM Press, pp. 196–207.

[3] Chan, T. More algorithms for all-pairs shortest paths in weighted graphs. In *Proc. of the 39th ACM symposium on Theory of computing* (2007), ACM, pp. 590–598.

[4] Clarisó, R., and Cortadella, J. The octahedron abstract domain. *Science of Computer Programming 64*, 1 (2007), 115–139.

[5] Cousot, P., and Cousot, R. Static determination of dynamic properties of programs. In *Proc. 2^{nd} Int. Symp. on Programming* (Paris, 1976), Dunod, pp. 106–130.

[6] Cousot, P., and Cousot, R. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4^{th} POPL* (Los Angeles, 1977), ACM Press, New York, pp. 238–252.

[7] Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., and Rival, X. Why does Astrée scale up ? *Formal Methods in System Design 35*, 3 (2009), 229–264.

[8] Cousot, P., and Halbwachs, N. Automatic discovery of linear restraints among variables of a program. In *5^{th} POPL* (Tucson, 1978), ACM Press, NY, pp. 84–97.

[9] Dobson, K. *Grid Domains for Analysing Software*. PhD thesis, The University of Leeds, School of Computing, 2008.

[10] Feret, J. Numerical abstract domains for digital filters. In *NSAD'05* (2005).

[11] Frederickson, G. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput. 16*, 6 (1987), 1004–1022.

[12] Halbwachs, N., Merchat, D., and Gonnord, L. Some ways to reduce the space dimension in polyhedra computations. *FMSD 29*, 1 (2006), 79–95.

[13] Howe, J., and King, A. Closure Algorithms for Domains with Two Variables per Inequality. Tech. rep., School of Informatics, City University London, 2009.

[14] Howe, J., and King, A. Logahedra : A new weakly relational domain. *Automated Technology for Verification and Analysis* (2009), 306–320.

[15] Howe, J., King, A., and Lawrence-Jones, C. Quadtrees as an Abstract Domain. *NSAD'10* (2010).

[16] Jeannet, B., and Miné, A. Apron : A library of numerical abstract domains for static analysis. In *CAV'09* (Grenoble, 2009), LNCS vol. 5643, Springer, pp. 661–667.

[17] Karr, M. Affine relationships among variables of a program. *Acta Informatica 6*, 2 (1976), 133–151.

[18] Lalire, G., Argoud, M., and Jeannet, B. The Interproc analyzer. `http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc`.

[19] Laviron, V., and Logozzo, F. Subpolyhedra : A (more) scalable approach to infer linear inequalities. In *VMCAI'09* (January 2009), Springer, pp. 229–244.

[20] Logozzo, F., and Fähndrich, M. Pentagons : A weakly relational abstract domain for the efficient validation of array accesses. In *ACM SAC'08* (2008), pp. 184–188.

[21] Miné, A. A new numerical abstract domain based on difference-bound matrices. In *Proc. of the PADO II* (Aarhus, 2001), LNCS vol. 2053, Springer, pp. 155–172.

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

[23] Péron, M., and Halbwachs, N. An abstract domain extending Difference-Bound Matrices with disequality constraints. In *VMCAI* (2007), Springer, pp. 268–282.

[24] Sankaranarayanan, S., Sipma, H., and Manna, Z. Scalable analysis of linear systems using mathematical programming. In *VMCAI* (2005), Springer, pp. 25–41.

[25] Simon, A., King, A., and Howe, J. Two variables per linear inequality as an abstract domain. *LOPSTR* (2003), 955–955.

[26] Venet, A., and Brat, G. Precise and efficient static array bound checking for large embedded C programs. In *PLDI'04* (Washington, 2004), ACM Press, pp. 231–242.