Inférer et compiler la terminaison des programmes logiques avec contraintes

Sébastien HOARAU, Fred MESNARD

IREMIA, Université de La Réunion
BP 7151 - 97715 Saint-Denis Messag. Cedex 9
FRANCE

E-mail : {seb, fred}@univ-reunion.fr

URL : http://www.univ-reunion.fr/~gcc

Résumé *: Ce papier présente une méthode automatisée qui traite en deux étapes la terminaison de programmes logiques contraints. Dans un premier temps, et en utilisant des techniques d'approximations de programmes et de mu-calcul sur les booléens, la méthode infère un ensemble de classes de requêtes sûres. Par "sûres", nous entendons que pour chacune de ces classes il existe un ordre statique des littéraux des clauses du programme garantissant la terminaison gauche universelle. Puis, étant donnée une classe parmi celles-là, la deuxième étape consiste à compiler le programme original en un nouveau par réordonnancement des littéraux à l'intérieur des clauses. Pour ce nouveau programme, la terminaison gauche, au sens universel, de toute requête de la classe considérée est assurée.*

Mots-clés *: Programmation logique avec contraintes, terminaison, analyse statique.*

1 Introduction

Depuis plusieurs années, de nombreux travaux de recherche se sont penchés sur l'analyse de la terminaison des programmes logiques (contraints) (voir [DES 93]). Pour la plupart des auteurs, le principal problème concerne la terminaison gauche universelle (finitude de l'arbre de preuves Prolog) d'une classe *donnée* de requêtes. Ce point de vue quelque peu restrictif pour la programmation logique est élargi dans [MES 96] : il s'agit cette fois d'inférer les classes de requêtes pour lesquelles la terminaison gauche est garantie. Dans cet article, nous allons encore plus loin et déterminons les classes de requêtes telles qu'il existe un réordonnancement statique du corps des clauses assurant la terminaison gauche.

Exemple 1 *Considérons le programme suivant dans CLP($\mathcal{N}$) :*

$$\texttt{carre}(\texttt{0},\texttt{0}).$$
$$\texttt{carre}(\texttt{X}+\texttt{1},\texttt{Y}+\texttt{2X}+\texttt{1}) \texttt{ :- carre}(\texttt{X},\texttt{Y}).$$
$$\texttt{p}(\texttt{X},\texttt{Y}) \texttt{ :- carre}(\texttt{X},\texttt{Z}),\texttt{carre}(\texttt{Z},\texttt{Y}).$$

La requête $\leftarrow X \leq 2, p(X,Y)$ *termine à gauche, mais pas* $\leftarrow Y \leq 16, p(X,Y)$. *Pourtant, si dans la clause définissant le prédicat p nous prouvons* $\leftarrow Y \leq 16, carre(Z,Y)$ *avant* $\leftarrow Y \leq 16, carre(X,Z)$ *(où nous ne possédons pas d'information sur les arguments) alors la preuve termine. Nous aimerions donc conclure qu'il existe deux classes de requêtes qui terminent :* $\leftarrow X$ *majorée*, $p(X,Y)$ *et* $\leftarrow Y$ *majorée,* $p(X,Y)$. $\triangleleft$

La suite du papier est organisée de la façon suivante : la section 2 rappelle quelques notions de base de la programmation logique par contraintes (PLC) et des approximations entre deux systèmes de PLC. La section 3 résume nos travaux antérieurs pour l'inférence des conditions de terminaison gauche. Puis les sections 4 et 5 présentent la méthode de cet article : comment, par réordonnancement des littéraux, nous pouvons considérer des classes plus larges de requêtes pour la propriété de terminaison. Enfin, dans la section 6, nous discutons des résultats de notre implantation de la méthode.

2 Préliminaires

2.1 *PLC(χ)*

À propos de la PLC, rappelons quelques notations et conventions introduites dans [JAF 94]. Dans un souci de concision, nous simplifions les notations lorsqu'il n'y a pas d'ambiguïté. Pour la même raison, nous ne donnons aucune preuve des propositions et théorèmes (elles figurent cependant dans une version longue de cet article disponible auprès des auteurs). Dans les exemples concrets de programmes, nous adoptons la syntaxe d'Edimbourgh.

Dans la suite, nous considérons des systèmes PLC idéaux[1] et sans élément limite. $\tilde{t}$ (resp. $\tilde{x}$) représente un n-uplet de termes (resp. de variables distinctes). Soit o un objet de PLC(χ). $var(o)$ dénote l'ensemble des variables de o et $o(\tilde{x})$ signifie o avec $var(o) = \tilde{x}$. Si o_1 et o_2 sont deux objets, $o_1 \equiv o_2$ signifie que o_1 et o_2 sont *syntaxiquement égaux*. Une χ-contrainte est une contrainte de la structure χ. La χ-contrainte c est *satisfiable* et θ est une χ-*solution* de c si θ est une χ-*valuation*

1. i.e. des systèmes qui incorporent une procédure de décision pour le problème : $\models \exists\, c$ (où c une contrainte).

telle que $\chi \models c\theta$. Sinon, c est χ-*insatisfiable*. Soit c_1 et c_2 deux χ-contraintes. Nous écrivons $c_1 \rightarrow_\chi c_2$ (resp. $c_1 \leftrightarrow_\chi c_2$) comme raccourci pour $\chi \models \forall[c_1 \rightarrow c_2]$ (resp. $\chi \models \forall[c_1 \leftrightarrow c_2]$). Il nous arrivera d'omettre le symbole χ. Nous utilisons les structures suivantes (où les symboles possèdent leurs interprétations usuelles) :

- $\mathcal{B} = \langle\{vrai, faux\}; \{0, 1, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}; \{\Rightarrow, =\}\rangle$.
- $\mathcal{N} = \langle\mathbb{N}; \{0, 1, +\}; \{\geq, =\}\rangle$ où $\mathbb{N}$ désigne l'ensemble des entiers naturels.

Dans $\mathcal{N}$, pour $n \in \mathbb{N}$, $n \geq 1$, nous écrivons n (resp. nx) comme raccourci pour $1+\ldots+1$ (resp. $x+\ldots+x$), où 1 (resp. x) apparaît n fois. Soit $c(\tilde{x})$ une $\mathcal{N}$-contrainte (nous parlons aussi de contrainte numérique), $y \in \tilde{x}$ et $m \in \mathbb{N}$. Nous disons que y est *majorée* (resp. *majorée par* m) dans c si il existe $n \in \mathbb{N}$ tel que $c(\tilde{x}) \rightarrow_\mathcal{N} n \geq y$ (resp. $c(\tilde{x}) \rightarrow_\mathcal{N} m \geq y$).

Sur l'ensemble Π des symboles de prédicats d'un programme P, nous définissons la relation binaire : $p \rightarrow q$ vraie si P contient une règle de la forme $p(\tilde{t}_1) \leftarrow \ldots, q(\tilde{t}_2), \ldots$. Soit $\rightarrow^*$ la fermeture réflexive transitive de la relation $\rightarrow$. La relation $p \sim q$ vraie si $p \rightarrow^* q \wedge q \rightarrow^* p$ est une relation d'équivalence. Nous notons $\bar{p}$ la classe d'équivalence contenant p.

Une règle $p(\tilde{x}) \leftarrow c, \tilde{B}$ est *récursive* s'il existe un symbole de prédicat $q \in \bar{p}$ tel que $\tilde{B} = \ldots, q(\tilde{y}), \ldots$. Un symbole de prédicat p est *directement récursif* si $\bar{p} = \{p\} \wedge (p \rightarrow p)$. Les symboles de prédicat $\{p_1, \ldots, p_n\}$ $(n \geq 2)$ sont *mutuellement récursifs* si $\bar{p_1} = \ldots = \bar{p_n} = \{p_1, \ldots, p_n\}$. Un prédicat p est *récursif* s'il est directement récursif ou mutuellement récursif. Sinon, p est *non-récursif*.

2.2 *De PLC(χ) à PLC($\mathcal{B}ool$)*

Nous invitons le lecteur à se référer à [COD 92, MES 96] pour les détails concernant les approximations entre deux structures PLC. Nous en rappelons brièvement la définition. Soit $\chi = \langle D_\chi; F_\chi; \{=_\chi\} \cup C_\chi\rangle$ où D_χ est le domaine de χ, F_χ est un ensemble de symboles de fonction et C_χ un ensemble de symboles de relation, une structure *solution-compacte*. Soit $\psi = \langle D_\psi; F_\psi; \{=_\psi\} \cup C_\psi\rangle$ une autre structure solution-compacte.

Définition 1 Une *approximation* $\mathcal{A}^\psi_\chi$ de χ vers ψ est une paire de fonctions $\langle\mathcal{A}_{sx}, \mathcal{A}_{sm}\rangle$ où :

1. $\mathcal{A}_{sx}$ est une application de $F_\chi \cup \{faux, vrai, =_\chi\} \cup C_\chi$ vers $F_\psi \cup \{faux, vrai, =_\psi\} \cup C_\psi$ telle que :

- pour toute fonction ou relation $s, arite(s) = arite(\mathcal{A}_{sx}(s))$,
- $\mathcal{A}_{sx}(F_\chi) \subseteq F_\psi$,
- $\mathcal{A}_{sx}(=_\chi) = =_\psi$,
- $\mathcal{A}_{sx}(C_\chi) = C_\psi$;

2. $\mathcal{A}_{sm}$ est une application de D_χ vers D_ψ telle que pour tout $\tilde{e}_\chi \in \tilde{D}_\chi$:

- pour toute fonction $f_\chi, \mathcal{A}_{sm}(f_\chi(\tilde{e}_\chi)) =_\psi \mathcal{A}_{sx}(f_\chi)(\mathcal{A}_{sm}(\tilde{e}_\chi))$;
- pour toute relation c_χ, si $\chi \models c_\chi(\tilde{e}_\chi)$ alors $\psi \models \mathcal{A}_{sx}(c_\chi)(\mathcal{A}_{sm}(\tilde{e}_\chi))$.

Remarquons que la définition d'approximation peut être étendue aux structures multi-sortes, aux termes, aux buts et aux programmes. La principale approximation que nous utiliserons est $\mathcal{A}^{\mathcal{B}}_{\chi}$ qui transforme des entités de PLC(χ) en des entités booléennes et qui est la composée de deux approximations :

1. $\mathcal{A}^{\mathcal{N}}_{\chi}$: qui, informellement, remplace toute structure de données par sa *taille* (par exemple les listes sont remplacées par leurs longueurs, les arbres par le nombre de nœuds...),

2. $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}$: $\mathcal{A}_{sx}(0) = 1, \mathcal{A}_{sx}(1) = 1, \mathcal{A}_{sx}(+) = \wedge, \mathcal{A}_{sx}(\geq) = \Longrightarrow, \mathcal{A}_{sm}(n) = vrai$.

Exemple 2 Soit $\mathcal{L}(\chi) = \langle \mathcal{D}^*_\chi; \{[\,], [\_|\_]\}; \{=\}\rangle$ la structure de listes, où [ ] dénote la liste vide et l'opérateur [_|_] le constructeur de listes.

Soit $\mathcal{A}^{\mathcal{N}}_{\mathcal{L}(\chi)}$, l'approximation de $\mathcal{L}(\chi)$ vers $\mathcal{N}$: $\mathcal{A}_{sx}([\,]) = 0, \mathcal{A}_{sx}([x|y]) = 1 + \mathcal{A}_{sx}(y)$ et $\mathcal{A}_{sm}([e_1, \ldots, e_n]) = n$.

Approximons le célèbre programme CONCATENE de PLC($\mathcal{L}(\chi)$) :

$$\texttt{concatene}([\,], \texttt{Ys}, \texttt{Ys}).$$
$$\texttt{concatene}([\texttt{X}|\texttt{Xs}], \texttt{Ys}, [\texttt{X}|\texttt{Zs}]) \texttt{ :} - \texttt{concatene}(\texttt{Xs}, \texttt{Ys}, \texttt{Zs}).$$

Son approximation dans PLC($\mathcal{N}$) est :

$$\texttt{concatene}(0, \texttt{Ys}, \texttt{Ys}).$$
$$\texttt{concatene}(1 + \texttt{Xs}, \texttt{Ys}, 1 + \texttt{Zs}) \texttt{ :} - \texttt{concatene}(\texttt{Xs}, \texttt{Ys}, \texttt{Zs}).$$

et son approximation booléenne :

$$\texttt{concatene}(1, \texttt{Ys}, \texttt{Ys}).$$
$$\texttt{concatene}(1 \wedge \texttt{Xs}, \texttt{Ys}, 1 \wedge \texttt{Zs}) \texttt{ :} - \texttt{concatene}(\texttt{Xs}, \texttt{Ys}, \texttt{Zs}).$$

◁

Donnons maintenant quelques résultats utiles sur les approximations.

Soit $\mathcal{A}^{\psi}_{\chi}$ une approximation, P un programme de PLC(χ) et S^{χ}_{P} la sémantique de ce programme [GAB 91]. Alors, l'image de la sémantique de P est incluse dans la sémantique de l'image de P :

Théorème 2 *[MES 96]* $\quad \mathcal{A}^{\psi}_{\chi}(S^{\chi}_{P}) \subseteq S^{\psi}_{\mathcal{A}^{\psi}_{\chi}(P)}$

Et voici une autre propriété très utile concernant l'approximation booléenne $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}$:

Proposition 3 *[MES 96] Soit c_1 une $\mathcal{N}$-contrainte, c_2 une $\mathcal{B}$-contrainte telles que $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c_1) \rightarrow_{\mathcal{B}} c_2$ et $t \equiv \vee_{j\in J}(\wedge_{i\in I_j} x_i)$ un terme booléen. Si $c_2 \rightarrow_{\mathcal{B}} t$ alors $\exists j \in J, \forall i \in I_j, x_i$ est majoré dans c_1.*

De plus nous pouvons calculer le *modèle booléen* d'un programme de PLC(χ) :

Remarque 1 *Soit P un programme de PLC(χ). Nous pouvons admettre (sans que cela impose de restriction) que $S^{\mathcal{B}}_{\mathcal{A}^{\mathcal{B}}_{\chi}(P)}$, le modèle de $\mathcal{A}^{\mathcal{B}}_{\chi}(P)$ (appelé aussi le modèle booléen de P), contient exactement une formule $p(\tilde{x}) \leftrightarrow Post_p(\tilde{x})$ pour chaque symbole de prédicat p de P.*

La relation booléenne $Post_p$ est le modèle booléen du prédicat p. Nous le notons ainsi par analogie à la notion de post-condition de De Schreye et Decorte [DES 93].

Exemple 3 Si nous considérons une fois de plus le programme CONCATENE, nous avons :

$$Post_{concatene}(x, y, z) = x \wedge (y \Leftrightarrow z).$$

Intuitivement, cela signifie que lorsque toute preuve d'un but $\leftarrow concatene(X, Y, Z)$ termine dans PLC($\mathcal{L}(\chi)$), d'une part, la longueur de X est majorée et, d'autre part, la longueur de Y est majorée si et seulement si celle de Z est majorée. $\triangleleft$

Remarque 2 *À partir de maintenant, dans les sections qui suivent, nous considérons uniquement des programmes dans PLC($\mathcal{N}$) (les autres s'y ramenant via la définition d'une notion de taille et l'utilisation d'une approximation).*

3 Inférence des conditions de terminaison gauche : la première approche

Dans [MES 96], pour chaque symbole de prédicat p d'un programme P, un terme booléen Pre_p est calculé. Ce terme est une condition de terminaison gauche i.e. tel que, pour toute requête $\leftarrow c, p(\tilde{x})$, si $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c) \rightarrow_{\mathcal{B}} Pre_p(\tilde{x})$ alors la dérivation de la dite requête termine à gauche.

Résumons ce résultat. À chaque appel récursif dans un programme, nous aimerions qu'une certaine quantité décroisse pour assurer la terminaison du processus. Dans [MES 96], cette quantité est appelée *mesure linéaire*.

Définition 4 Une *mesure linéaire* μ_p pour un symbole de predicat $p \in \bar{p}$, d'arité n, est une application définie par :

$$\begin{array}{rccc} \mu_p : & \mathbb{N}^n & \rightarrow & \mathbb{N} \\ & (x_1, \ldots, x_n) & \mapsto & \sum_{i=1}^{n} a_i x_i \end{array}$$

telle que les a_i sont des entiers relatifs (nous notons par ailleurs I_{μ_p} l'ensemble des indices i tels que $a_i \neq 0$) et pour chaque règle $p(\tilde{x}) \leftarrow c, \tilde{B}$ définissant p, pour chaque solution θ de c, pour chaque atome $q(\tilde{y})$ apparaissant dans $\tilde{B}$ avec $q \in \bar{p}$ nous avons : $\mu_p(\tilde{x}\theta) \geq 1 + \mu_q(\tilde{y}\theta)$ et $\mu_q(\tilde{y}\theta) \geq 0$.

Exemple 4 Si nous considérons la version numérique de CONCATENE, $\mu^1(x, y, z) = x$ et $\mu^2(x, y, z) = z$ sont deux mesures linéaires. $\triangleleft$

D'après un résultat de K. Sohn and A. Van Gelder [SOH 91], il existe une procédure de décision complète pour l'existence des mesures linéaires d'un prédicat. Cette procédure peut-être adaptée pour la calcul effectif des coefficients de μ.

Définition 5 Soit P un programme de PLC($\mathcal{N}$). Soit $p \in \bar{p}$ un prédicat récursif. L'ensemble des mesures maximales pour p est toujours fini (voir [MES 96] pour plus de détails sur cette notion de maximalité des mesures). Nous notons q_p le nombre de mesures maximales et $\Gamma_{\bar{p}} = \{\mu_p^j \mid p \in \bar{p}, 1 \leq j \leq q_p\}$ l'ensemble des mesures maximales pour $\bar{p}$. Pour chaque $p \in \bar{p}$, nous calculons un terme booléen appelé *mesure booléenne* de p et défini par :

$$\gamma_p(\tilde{x}) = \bigvee_{1 \leq j \leq q_p} \left(\bigwedge_{i \in I_{\mu_p^j}} x_i \right)$$

Si p est un prédicat non récursif, nous posons $\gamma_p(\tilde{x}) = 1$.

Exemple 5 Si nous continuons notre exemple 4, nous avons $\Gamma_{\overline{concatene}} = \{\mu^1, \mu^2\}$ et la mesure booléenne est $\gamma_{concatene}(x, y, z) = x \vee z$. $\triangleleft$

Il est temps de revenir à notre principale préoccupation : la terminaison gauche.

Définition 6 Un terme booléen $Pre_p(\tilde{x})$ est une *condition de terminaison gauche* pour $p(\tilde{x})$ si, pour toute contrainte c, $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c) \rightarrow_{\mathcal{B}} Pre_p(\tilde{x})$ implique $\leftarrow c, p(\tilde{x})$ termine à gauche.

Soit P un programme de PLC($\mathcal{N}$), p un symbole de prédicat de P défini par m_p règles. Soit $P^{\mathcal{B}}$ la version booléenne de P et r_k, la k-ième règle définissant p dans $P^{\mathcal{B}}$: $p(\tilde{x}) \leftarrow c^{\mathcal{B}}_{k,0}, p_{k,1}(\tilde{x}_{k,1}), \ldots, p_{k,j_k}(\tilde{x}_{k,j_k})$

Théorème 7 *Supposons que pour chaque prédicat $q \notin \bar{p}$ apparaissant dans au moins une règle définissant $\bar{p}$, nous possédons une condition de terminaison gauche $Pre_q(\tilde{y})$. Si $\{Pre_p(\tilde{x})\}_{p\in\bar{p}}$ vérifie :*

$$\forall p \in \bar{p} \begin{cases} Pre_p(\tilde{x}) \rightarrow_{\mathcal{B}} \gamma_p(\tilde{x}), \\ \forall 1 \leq k \leq m_p, \forall 1 \leq i \leq j_k, \left(Pre_p(\tilde{x}) \wedge c^{\mathcal{B}}_{k,0} \bigwedge_{j=1}^{i-1} Post_{p_j}(\tilde{x}_j)\right) \\ \qquad\qquad \rightarrow_{\mathcal{B}} Pre_{p_{k,i}}(\tilde{x}_{k,i}) \end{cases}$$

alors $\{Pre_p(\tilde{x})\}_{p\in\bar{p}}$ est une condition de terminaison gauche pour $\bar{p}$.

Notons que le système de formules booléennes ci-dessus peut-être utilisé non seulement pour tester que telles ou telles relations $\{Pre_p(\tilde{x})\}_{p\in\bar{p}}$ sont des conditions de terminaison gauche mais également pour *calculer* ces conditions de terminaison. Techniquement, nous utilisons un module de mu-calcul sur les booléens [COL 97] associé à notre moteur Prolog pour résoudre ce système.

Nous présentons à présent un exemple complet permettant de se faire une idée plus intuitive des résultats ci-dessus. De plus, cet exemple montre les limites des conditions inférées et justifie donc le contenu de la section suivante.

Exemple 6 Considérons à nouveau le programme PUISSANCE-4 dans CLP($\mathcal{N}$) :

```
carre(0,0).
carre(X + 1, Y + 2X + 1) :- carre(X,Y).

p(X,Y) :- carre(X,Z), carre(Z,Y).
```

Nous avons : $\gamma_{carre}(x, y) = x \vee y$, $Post_{carre}(x, y) = x \wedge y$, $\gamma_p(x, y) = 1$ et $Post_p(x, y) = x \wedge y$. Comme conditions de terminaison gauche nous trouvons :

$Pre_{carre}(x,y) = x \vee y$ and $Pre_p(x,y) = x$. De façon informelle, nous pouvons interpréter ces relations de la sorte :

- pour le prédicat s : pour toute requête $\leftarrow c, carre(x,y)$, si x **ou** y est majorée dans c,alors la requête considérée termine à gauche (information tirée de Pre_s) et après toute preuve x **et** y sont majorées ($Post_s$).

- pour le prédicat p : dans une requête $\leftarrow c, p(x,y)$, si x est majorée dans c, alors la dite requête termine à gauche et à la fin de toute preuve x **et** y sont majorées.

Cependant, nous pouvons remarquer que dans une requête $\leftarrow c, p(x,y)$ telle que y est majorée dans c, la requête termine si nous prouvons $carre(z,y)$ avant $carre(x,z)$ i.e. il y a terminaison gauche si nous considérons cette nouvelle version du programme :

$$\texttt{carre(0,0).}$$
$$\texttt{carre(X + 1, Y + 2X + 1) :- carre(X,Y).}$$
$$\texttt{p(X,Y) :- carre(Z,Y), carre(X,Z).}$$

Ainsi, on peut dire qu'un appel $p(x,y)$ termine si x **ou** y est majorée c'est-à-dire $Pre_p(x,y) = x \vee y$. Inférer de telles classes de requêtes et réordonner les littéraux pour une classe donnée afin d'assurer la terminaison gauche est le but des sections 4 et 5. $\triangleleft$

4 Inférer des conditions de terminaison gauche étendues

Cette section présente une méthode automatisable permettant d'inférer des conditions de terminaison gauche plus larges (voir l'exemple 6). L'idée principale consiste à introduire une notion d'ordre dans le corps des clauses.

Définition 8 Soit P un programme. Un terme booléen $Pre_p(\tilde{x})$ est une *condition de terminaison gauche étendue* pour $p(\tilde{x})$ si, pour toute contrainte c, $\mathcal{A}^{\mathcal{B}}_{\mathcal{N}}(c) \rightarrow_{\mathcal{B}} Pre_p(\tilde{x})$ implique qu'il existe un réordonnancement statique des littéraux des clauses de P tel que $\leftarrow c, p(\tilde{x})$ termine à gauche.

Définition 9 Soit P un programme. Pour chaque règle de P: $p(\tilde{x}) \leftarrow c, \tilde{B}$ où le corps $\tilde{B}$ contient les littéraux : $p_1, \ldots, p_n$ (dans un ordre quelconque), nous pouvons associer une séquence de $n(n-1)/2$ variables booléennes $(b_{ij})_{1 \leq i < j \leq n}$, que nous appelons *variables d'ordre* dont la sémantique est la suivante :

$$\forall\, 1 \leq i < j \leq n, \begin{cases} b_{ij} & = 1 \text{ si } p_i \text{ apparaît avant } p_j, \\ & = 0 \text{ si } p_j \text{ apparaît avant } p_i \end{cases}$$

Ces variables doivent vérifier les contraintes suivantes (exprimant la transitivité de la relation d'ordre) :

$$\forall 1 \leq i < j < k \leq n, \begin{cases} b_{ij} \wedge b_{jk} \rightarrow b_{ik} \\ b_{ik} \wedge \neg b_{jk} \rightarrow b_{ij} \\ \neg b_{ij} \wedge b_{ik} \rightarrow b_{jk} \\ b_{jk} \wedge \neg b_{ik} \rightarrow \neg b_{ij} \\ \neg b_{ik} \wedge b_{ij} \rightarrow \neg b_{jk} \\ \neg b_{jk} \wedge \neg b_{ij} \rightarrow \neg b_{ik} \end{cases}$$

Et nous en venons au principal théorème de cet article. Soit P un programme de PLC($\mathcal{N}$), p un symbole de prédicat de P défini par m_p règles. Soit $P^{\mathcal{B}}$ la version booléenne de P et r_k, la k-ième règle définissant p dans $P^{\mathcal{B}}$:

$$p(\tilde{x}) \leftarrow c^{\mathcal{B}}_{k,0}, p_{k,1}(\tilde{x}_{k,1}), \ldots, p_{k,j_k}(\tilde{x}_{k,j_k})$$

Théorème 10 *Supposons que pour chaque prédicat $q \notin \bar{p}$ apparaissant dans au moins une règle définissant $\bar{p}$, nous possédons une condition de terminaison gauche $Pre_q(\tilde{y})$. Si $\{Pre_p(\tilde{x})\}_{p\in\bar{p}}$ vérifie :*

$$\forall p \in \bar{p} \begin{cases} Pre_p(\tilde{x}) \rightarrow_{\mathcal{B}} \gamma_p(\tilde{x}), \\ \forall 1 \leq k \leq m_p, \exists (b^k_{eh})_{1\leq e<h\leq j_k} \bigwedge_{1\leq j\leq j_k} \Big[\Big(Pre_p(\tilde{x}) \wedge c^{\mathcal{B}}_{k,0} \wedge \\ \bigwedge_{i=1}^{j-1}(\neg b^k_{ij} \vee Post_{p_{k,i}}(\tilde{x}_{k,i})) \wedge \bigwedge_{i=j+1}^{j_k}(b^k_{ji} \vee Post_{p_{k,i}}(\tilde{x}_{k,i}))\Big) \rightarrow_{\mathcal{B}} \\ Pre_{p_{k,j}}(\tilde{x}_{k,j})\Big] \end{cases}$$

alors $\{Pre_p(\tilde{x})\}_{p\in\bar{p}}$ est une condition de terminaison gauche étendue pour $\bar{p}$.

La signification intuitive de ce système est analogue à celle du théorème 7. La première implication est une sorte de condition minimale. La deuxième exprime le fait suivant : à l'appel d'un littéral, le contexte d'appel (le premier membre de l'implication booléenne) doit impliquer la condition de terminaison du littéral (le second membre). Dans le théorème 7, comme l'ordre des littéraux est fixé, la traduction booléenne formelle est simple. Ici, la chose se complique : lorsque l'on traite un littéral (celui qui va être prouvé), pour chacun des autres ou bien il apparaît après le littéral courant ou bien il apparaît avant et dans ce cas le contexte d'appel est enrichi de la post-condition de ce littéral. C'est ce que traduit le terme $\bigwedge_{i=1}^{j-1}(\neg b^k_{ij} \vee Post_{p_{k,i}}(\tilde{x}_{k,i})) \wedge \bigwedge_{i=j+1}^{j_k}(b^k_{ji} \vee Post_{p_{k,i}}(\tilde{x}_{k,i}))$ (le fait que cette conjonction soit coupée en deux est simplement dû à la notation d'une variable d'ordre : dans l'écriture b_{ij} on a $i < j$).

La technique mise en œuvre pour la résolution de cette formule (donnant la relation Pre_p) est analogue à celle du théorème 7 : il s'agit d'un calcul de point fixe via un module de mu-calcul sur les booléens.

5 Réordonner pour terminer

5.1 *Environnement d'un programme*

Commençons par quelques définitions auxiliaires.

Définition 11 Soit n un entier naturel. Une *permutation* σ sur $\{1,\ldots,n\}$ est une bijection de $\{1,\ldots,n\}$ vers $\{1,\ldots,n\}$. Nous notons σ_i l'image de i par la permutation σ et σ^{-1} la permutation réciproque.

Remarque 3 *Une séquence $(b_{ij})_{1\leq i<j\leq n}$ de valeurs booléennes donne un ordre total sur les littéraux $p_1,\ldots,p_n$, explicitement : $p_{i_1},\ldots,p_{i_n}$. Ainsi, nous pouvons associer à une séquence une unique permutation sur $\{1,\ldots,n\}$ définie par : $\forall 1\leq j\leq n, \sigma_j = i_j$. Ceci est le lien entre la notion de variables d'ordre (définition 9) et la notion de permutation. L'exemple suivant illustre ce lien.*

Exemple 7 Soit p_1,p_2,p_3,p_4 quatre littéraux et $(b_{ij})_{1\leq i<j\leq 4}$ une séquence de variables d'ordre telle que $b_{23}=1, b_{13}=1, b_{14}=0$. Ici, nous avons une instanciation partielle des b_{ij}'s. Si nous voulons effectivement construire un corps de clause avec ces littéraux qui tienne compte des informations données par b_{23}, b_{13} et b_{14}, nous devons donner une valeur aux autres b_{ij}'s en respectant les contraintes de transitivité. Dans notre exemple il y a trois possibilités :

- $b_{12}=0, b_{24}=1, b_{34}=0$ correspondant à l'ordre suivant : $\langle p_2,p_4,p_1,p_3\rangle$,
- $b_{12}=1, b_{24}=0, b_{34}=0$, ce qui nous donne : $\langle p_4,p_1,p_2,p_3\rangle$,
- $b_{12}=0, b_{24}=0, b_{34}=0$, ce qui nous donne : $\langle p_4,p_2,p_1,p_3\rangle$. $\triangleleft$

Définition 12 Soit Pre_p la condition de terminaison gauche étendue d'un prédicat p. Pre_p est un terme booléen qui peut s'écrire comme la disjonction de conjonctions de littéraux. Nous appelons *classe de requêtes* une telle conjonction.

Voici maintenant la définition d'un environnement :

Définition 13 Nous définissons un *environnement* comme étant un triplet $\langle P, \{classes(p) \mid p\in\Pi\}, \phi\rangle$ où :

- P est un programme de PLC($\mathcal{N}$),

- $classes(p)$ est un ensemble $\{C_p^1,\ldots,C_p^{l_p}\}$ de classes de requêtes associées à chaque symbole de prédicat $p\in\Pi$ et

- ϕ est une fonction qui, à chaque couple $\langle$ règle r_k : $p\leftarrow c, p_{k,1},\ldots,p_{k,j_k}$; classe $C_p^l\rangle$ associe une permutation sur $\{1,\ldots,j_k\}$.

Exemple 8 Voici un exemple d'environnement :

- le programme P $\left\{ \begin{array}{ll} & \mathtt{p(0,0).} \\ \text{règle } r_1 \ : & \mathtt{p(X,Y)} \ \mathtt{:-} \ \mathtt{c, q(Y), p(Y,Z).} \\ \text{règle } r_2 \ : & \mathtt{p(X,Y)} \ \mathtt{:-} \ \mathtt{d, p(Y,X).} \\ & \\ & \mathtt{q(0).} \\ \text{règle } r_3 \ : & \mathtt{q(X)} \ \mathtt{:-} \ \mathtt{e, q(Y).} \end{array} \right.$

où c, d et e sont des contraintes numériques. Nous n'avons pas nommé les clauses unitaires car ce type de clause n'intervient pas directement dans la méthode.

- les classes : $classes(p) = \{C_p^1, C_p^2\}, classes(q) = \{C_q^1\}$,

- les permutations : $\phi(r_1, C_p^1) = \sigma$ (où σ est la permutation définie par $\sigma_1 = 1$ et $\sigma_2 = 2$), $\phi(r_1, C_p^2) = \sigma'$ (où σ' est définie par $\sigma'_1 = 2$ et $\sigma'_2 = 1$), $\phi(r_2, C_p^1) = \omega$ ($\omega_1 = 1$), $\phi(r_2, C_p^2) = \omega, \phi(r_3, C_q^1) = \omega$. $\triangleleft$

Et voici une propriété possible des environnements :

Définition 14 Un environnement $E = \langle P, \{classes(p) \mid p \in \Pi\}, \phi\rangle$ est *clos* si pour toute règle $r_k \ : p(\tilde{x}) \leftarrow c_{k,0}, p_{k,1}(\tilde{x}_{k,1}), \ldots, p_{k,j_k}(\tilde{x}_{k,j_k}) \in P$, pour tout $C_p^l \in classes(p)$, avec $\sigma = \phi(r_k, C_p^l)$, pour tout littéral $p_{k,j}(\tilde{x}_{k,j})$ $(1 \leq j \leq j_k)$, il existe une classe de requête C de $classes(p_{k,j})$ telle que :

$$\left(C_p^l(\tilde{x}) \wedge c_{k,0}^{\mathcal{B}} \wedge \bigwedge_{i=1}^{\sigma_j^{-1}-1} Post_{p_{k,\sigma_i}}(\tilde{x}_{k,\sigma_i}) \right) \rightarrow_{\mathcal{B}} C(\tilde{x}_{k,j})$$

Notons que $\sigma_1, \ldots, \sigma(\sigma_j^{-1} - 1)$ sont les indices de prédicats apparaissant avant $p_{k,j}$ dans la séquence $\langle p_{k,\sigma_1}, \ldots, p_{k,j}, \ldots, p_{k,\sigma_{j_k}}\rangle$, comme le montre le schéma ci-dessous (on passe du n-uplet supérieur au n-uplet inférieur par σ, on remonte par σ^{-1}) :

$$\begin{array}{l} \overbrace{\langle 1, \ldots, \sigma_j^{-1} - 1}^{\{i \leq \sigma_j^{-1}-1\}}, \sigma_j^{-1}, \ldots, j_k\rangle \\ \underbrace{\langle \sigma_1, \ldots, \sigma(\sigma_j^{-1} - 1)}_{\{\sigma_i \mid i \leq \sigma_j^{-1}-1\}}, \boxed{\mathrm{j}}, \ldots, \sigma_{j_k}\rangle \end{array}$$

Essayons de donner une idée intuitive de cette propriété. Cette propriété dit qu'après réordonnancement des littéraux d'une clause, au début de la preuve d'un littéral ($p_{k,j}$) de la dite clause, le contexte d'appel (le terme booléen correspondant aux grandes parenthèses) implique une des classes de requêtes connues du littéral (C). Ainsi,

les contextes d'appels successifs ne font jamais sortir de l'environnement, d'où cette notion d'environnement clos.

Nous présentons ci-dessous les grandes lignes de la construction d'un environnement clos pour un programme donné :

1. Pour chaque $p \in \Pi$, nous inférons Pre_p, sa condition de terminaison gauche étendue (théorème 10).
2. Nous avons ensuite deux cas possibles :
   - $Pre_p(\tilde{x}) = 1$, alors nous définissons les classes de p par $classes(p) = \{1\}$,
   - $Pre_p(\tilde{x})$ peut s'écrire comme une disjonction de conjonctions de littéraux : $Pre_p(\tilde{x}) = \bigvee_{1 \leq l \leq l_p}(C_p^l)$, où les C_p^l's sont des conjonctions de littéraux (ce sont les classes de requêtes de p). Nous posons $classes(p) = \{C_p^1, \ldots, C_p^{l_p}\}$.
3. Il nous reste à définir une fonction ϕ. Pour ce faire, pour chaque prédicat p, pour chaque règle r_k définissant p, pour chaque $C_p^l \in classes(p)$, nous calculons à l'aide d'un solveur booléen une séquence $(b_{eh}^k)_{1 \leq e < h \leq j_k}$ de booléens telle que la formule suivante est vraie :
   $$\bigwedge_{j \leq j_k} \Big[\Big(C_p^l(\tilde{x}) \wedge c_{k,0}^{\mathcal{B}} \wedge \bigwedge_{i=1}^{j-1} (\neg b_{ij}^k \vee Post_{p_{k,i}}(\tilde{x}_{k,i})) \wedge \bigwedge_{i=j+1}^{j_k} (b_{ji}^k \vee Post_{p_{k,i}}(\tilde{x}_{k,i})) \Big) \rightarrow_{\mathcal{B}} Pre_{p_{k,j}}(\tilde{x}_{k,j}) \Big]$$
   Nous obtenons alors une instanciation partielle des variables d'ordre b_{ij}^k's que nous étendons en une instanciation totale (voir remarque 3) correspondant à une permutation σ. Il est clair qu'à ce moment il peut y avoir plusieurs possibilités (voir exemple 7). Il suffit d'en choisir une. Nous posons alors $\phi(r_k, C_p^l) = \sigma$.

Proposition 15 *Soit P un programme de PLC($\mathcal{N}$) et pour chaque $p \in \Pi$, $classes(p) = \{C_p^1, \ldots, C_p^{l_p}\}$ l'ensemble des classes de requêtes associé calculé par l'algorithme ci-dessus. Soit ϕ la fonction liant les couples $\langle$règle, classe de requêtes$\rangle$ aux permutations, calculée par le même algorithme. Alors, l'environnement $E = \langle P, \{classes(p) \mid p \in Pred(p)\}, \phi \rangle$ est clos.*

5.2 *Réordonnancement*

Définition 16 Soit $\langle P, \{classes(p) \mid p \in \Pi\}, \phi \rangle$ un environnement. Nous définissons une *table de renommage* comme un ensemble de triplets $\langle p, C_p^l, p^l \rangle$ où $p \in \Pi$, $C_p^l \in classes(p)$ et p^l est un nouveau symbole de prédicat.

Exemple 9 Avec l'environnement de l'exemple 8, voici une table de renommage possible : $\{\langle p, C_p^1, p^1\rangle; \langle p, C_p^2, p^2\rangle; \langle q, C_q^1, q^1\rangle\}$. $\triangleleft$

Nous continuons par la présentation de la procédure *réordonner* (procédure 2) qui, étant donnés un environnement clos (d'un programme P), une table de renommage associée et un couple $\langle$prédicat, classe de requêtes$\rangle$, construit un nouveau programme P'. Le programme P' diffère du programme P par un renommage des symboles de prédicat et un certain nombre de copies des clauses de P avec des ordres différents sur les littéraux des clauses (toutefois dans P' n'apparaît aucune clause n'ayant rien de commun avec une clause de P).

Mais avant la présentation de cette procédure, nous introduisons la sous procédure *ro* (*ro* pour *renommer - ordonner*, voir procédure 1) qui, étant donné un environnement $\langle P, \{classes(p) \mid p \in \Pi\}, \phi\rangle$, une table de renommage T et un couple $\langle p, C_p^l\rangle$, crée un paquet de clauses sur la base des clauses définissant p dans P par renommage des symboles de tête et réordonnancement des littéraux des corps (en utilisant les permutations définies par ϕ).

Procédure 1 ro

Entrée : un environnement $E = \langle P, \{classes(p) \mid p \in \Pi\}, \phi\rangle$, une table de renommage T, un couple $\langle p, C_p^l\rangle$, tel que $p \in \Pi$, $C_p^l \in classes(p)$.

Sortie : R un ensemble de clauses : $p^l(\tilde{x}) \leftarrow c, p_{\sigma_1}(\tilde{x}_{\sigma_1}), \ldots, p_{\sigma_m}(\tilde{x}_{\sigma_m})$ tel que $p^l \notin \Pi$.

1: **Début**
2: $R \leftarrow \emptyset$;
3: **Pour toute** règle r_k : $p(\tilde{x}) \leftarrow c_{k,0}, p_{k,1}(\tilde{x}_{k,1}), \ldots, p_{k,j_k}(\tilde{x}_{k,j_k}) \in P$ **faire**
4: $\sigma \leftarrow \phi(r_k, C_p^l)$;
5: soit p^l tel que $\langle p, C_p^l, p^l\rangle \in T$;
6: $R \leftarrow R \cup \{p^l(\tilde{x}) \leftarrow c_{k,0}, p_{k,\sigma_1}(\tilde{x}_{k,\sigma_1}), \ldots, p_{k,\sigma_{j_k}}(\tilde{x}_{k,\sigma_{j_k}}) \in P\}$;
7: **Fin pour**
8: Renvoyer R;
9: **Fin**

Exemple 10 Considérons une dernière fois l'environnement de l'exemple 8, la même table de renommage que celle de l'exemple 9 et le couple $\langle p, C_p^2\rangle$. Voici, pas à pas, la construction de P' par la procédure *réordonner* (présentée ci-après) :

Tout d'abord, P' contient une copie de la règle r_1 (resp. r_2) avec un renommage de la tête et des littéraux de corps réordonnés suivant la permutation σ' (resp. σ). Il

Procédure 2 réordonner

Entrée : un environnement clos $E = \langle P, \{classes(p) \mid p \in \Pi\}, \phi\rangle$, une table de renommage T, un couple $\langle p, C_p^l\rangle$, tel que $p \in \Pi$, $C_p^l \in classes(p)$.
Sortie : un programme P'.

1: **Début**
2: $P' \leftarrow ro(E, T, \langle p, C_p^l\rangle)$;
3: **Tant que** $\exists$ une règle de P' tq un prédicat du corps non défini dans P' **faire**
4: soit r_k : $q^\alpha(\tilde{y}) \leftarrow c_{\alpha,0}, q_{\alpha,1}(\tilde{y}_{\alpha,1}), \ldots, q_{\alpha,j_\alpha}(\tilde{y}_{\alpha,j_\alpha}) \in P'$;
5: soit C_q^α une classe de requêtes telle que $\langle q, C_q^\alpha, q^\alpha\rangle \in T$;
6: **Pour** $i = 1$ à j_α **faire**
7: **Si** $q_{\alpha,i} \in \Pi$ **alors**
8: choisir $C^\beta \in classes(q_{\alpha,i})$ tq
$$C_q^\alpha(\tilde{y}) \wedge c_{\alpha,0} \bigwedge_{j=1}^{i-1} Post_{q_{\alpha,j}}(\tilde{y}_{\alpha,j}) \rightarrow_{\mathcal{B}} C^\beta(\tilde{y}_{\alpha,i});$$
9: soit $q_{\alpha,i}^\beta$ tel que $\langle q_{\alpha,i}, C^\beta, q_{\alpha,i}^\beta\rangle \in T$;
10: $P' \leftarrow P' \backslash \{r_k\}$;
11: $P' \leftarrow P' \cup$
$$\{q^\alpha(\tilde{y}) \leftarrow c_{\alpha,0}, q_{\alpha,1}(\tilde{y}_{\alpha,1}), \ldots, q_{\alpha,i}^\beta(\tilde{y}_{\alpha,i}), \ldots, q_{\alpha,j_\alpha}(\tilde{y}_{\alpha,j_\alpha})\};$$
12: **Si** $q_{\alpha,i}^\beta \notin \Pi'$ **alors**
13: $P' \leftarrow P' \cup\ ro(E, T, \langle q_{\alpha,i}, C^\beta\rangle)$;
14: **Fin si**
15: **Fin si**
16: **Fin pour**
17: **Fin tant que**
18: Renvoyer P';
19: **Fin**

s'agit de la ligne 2 de l'algorithme :

$$P' \quad \begin{cases} \mathtt{p^2(X,Y)} \ \mathtt{:-\ c, p(Y,Z), q(Y).} \\ \mathtt{p^2(X,Y)} \ \mathtt{:-\ d, p(Y,X).} \\ \mathtt{p^2(0,0).} \end{cases}$$

Puis, le test de la ligne 3 étant vrai, pour chaque clause de P', nous devons renommer les symboles de prédicat du corps :

- pour la première règle, nous savons que : $\exists C_p^l \in classes(p)$ tel que $C_p^2(X,Y) \wedge c \rightarrow_{\mathcal{B}} C_p^l(Y,Z)$ (parce que l'environnement est clos). Supposons, pour les besoins de

l'exemple, que $l = 2$ (ligne 8). Alors P' devient (lignes 9, 10, 11):

$$\begin{array}{l} \mathtt{p^2(X,Y)} \ :- \ \mathtt{c, p^2(Y,Z), q(Y).} \\ \mathtt{p^2(X,Y)} \ :- \ \mathtt{d, p(Y,X).} \\ \mathtt{p^2(0,0).} \end{array}$$

- Puisque p^2 n'est pas un nouveau prédicat dans Π' (test en ligne 12 faux) alors nous considérons le second littéral $q(Y)$. Puisque $classes(q)$ contient un seul élément nous avons : $C_p^2(X) \wedge c(X,Y,Z) \wedge Post_p(Z) \rightarrow_{\mathcal{B}} C_q^1(Y)$ et nous renommons q en q^1 :

$$P' \quad \left\{ \begin{array}{l} \mathtt{p^2(X,Y)} \ :- \ \mathtt{c, p^2(Y,Z), q^1(Y).} \\ \mathtt{p^2(X,Y)} \ :- \ \mathtt{d, p(Y,X).} \\ \mathtt{p^2(0,0).} \end{array} \right.$$

- à ce stade, le test ligne 12 est vrai (q^1 n'apparaît pas dans P') et la sous-procédure ro crée les règles suivantes qui sont ajoutées à P' (ligne 13):

$$\begin{array}{l} \mathtt{q^1(X)} \ :- \ \mathtt{e, q(Y).} \\ \mathtt{q^1(0).} \end{array}$$

- puis la deuxième règle de p^2 est traitée. Supposons que $C_p^2(X) \wedge d(X,Y) \rightarrow_{\mathcal{B}} C_p^2(Y)$ alors nous renommons p en p^2:

$$P' \quad \left\{ \begin{array}{l} \mathtt{p^2(X,Y)} \ :- \ \mathtt{c, p^2(Y,Z), q^1(Y).} \\ \mathtt{p^2(X,Y)} \ :- \ \mathtt{d, p^2(Y,X).} \\ \mathtt{p^2(0,0).} \\ \mathtt{q^1(X)} \ :- \ \mathtt{e, q(Y).} \\ \mathtt{q^1(0)} \end{array} \right.$$

- et finalement, le dernier littéral de la dernière règle récursive est renommé et la version finale de P' est :

$$P' \quad \left\{ \begin{array}{l} \mathtt{p^2(X,Y)} \ :- \ \mathtt{c, p^2(Y,Z), q^1(Y).} \\ \mathtt{p^2(X,Y)} \ :- \ \mathtt{d, p^2(Y,X).} \\ \mathtt{p^2(0,0).} \\ \mathtt{q^1(X)} \ :- \ \mathtt{e, q^1(Y).} \\ \mathtt{q^1(0)} \end{array} \right.$$

◁

Maintenant détaillons les deux principales propriétés concernant la méthode que nous venons d'exposer. Le décor commun à ces propriétés est le suivant : P un programme de PLC($\mathcal{N}$), $E = \langle P, \{classes(p) \mid p \in \Pi\}, \phi \rangle$ un environnement clos

relatif à P et obtenu par la méthode de la sous-section **5.1**, T une table de renommage associée à E, $\leftarrow c(\tilde{y}), p(\tilde{x})$ une requête relative à P (où $c(\tilde{y})$ est une $\mathcal{N}$-contrainte) telle que $c^{\mathcal{B}}$ (la version booléenne de c) vérifie : $\exists C_p \in classes(p), c^{\mathcal{B}}(\tilde{y}) \rightarrow_{\mathcal{B}} C_p(\tilde{x})$, $P' = reordonner(E, T, \langle p, C_p \rangle)$ et p' un symbole de prédicat tel que $\langle p, C_p, p' \rangle \in T$.

Proposition 17 *Pour tout symbole de prédicat q de P', il existe $p \in \Pi$ et $C \in classes(p)$ tels que $\langle p, C, q \rangle \in T$. Si nous renommons chacun de ces symboles de prédicat q par son symbole p correspondant alors P' et P ont même sens i.e. :*

$$S^{\mathcal{N}}_{P'} = S^{\mathcal{N}}_{P}$$

La deuxième propriété concerne la terminaison :

Théorème 18 *La preuve de la requête $\leftarrow c(\tilde{y}), p'(\tilde{x})$ relative à P' termine à gauche.*

Nous terminons cet article par l'exemple de l'introduction.

Exemple 11 Quand nous appliquons notre méthode sur le programme PUISSANCE-4 (exemples 1 et 6). Nous trouvons : $Pre_{carre}(x, y) = x \vee y$, $Pre_p(x, y) = x \vee y$, $\phi(r_3, x) = \omega$ (défini par $\omega_1 = 1, \omega_2 = 2$), $\phi(r_3, y) = \omega'$ ($\omega'_1 = 2, \omega'_2 = 1$).

Si nous reconsidérons la requête de l'introduction : $\leftarrow Y \leq 16, p(X, Y)$. Elle correspond à la classe "Y majorée" i.e. $Y = 1$ dans la version booléenne. Le nouveau programme P' construit est :

rule r'_1 : $\mathtt{p'(X,Y)\ :-\ carre'(Z,Y), carre'(X,Z).}$

rule r'_2 : $\mathtt{carre'(0,0).}$

rule r'_3 : $\mathtt{carre'(X+1, Y+2X+1)\ :-\ carre'(X,Y).}$

L'appel $\leftarrow Y \leq 16, p'(X, Y)$ termine à gauche avec : $\{X = 0\}, \{X = 1\}, \{X = 2\}$.
◁

6 Conclusion

Résumons notre approche. Il s'agit d'une extension de notre précédent travail sur l'inférence de classes de requêtes qui terminent à gauche [MES 96]. Ici nous jouons sur l'ordre des littéraux pour calculer l'ensemble des classes de requêtes qui terminent. L'idée principale est la prise en compte de tous les ordres possibles par l'introduction d'une séquence de nouvelles variables booléennes (les variables d'ordre). Une fois

calculé cet ensemble de classes de requêtes, pour toute classe qui termine, nous compilons statiquement le contrôle vers Prolog. En d'autres termes, nous produisons un programme tel que pour toute requête de la classe considérée, la terminaison universelle gauche est assurée.

L'approche décrite dans [MES 96] est maintenant complètement implémentée. Pour le travail présenté ici, nous avons une implémentation du calcul de $\{Pre_p\}_{p\in\bar{p}}$ comme défini au point 1 de la méthode de la sous section **5.1**. Voici résumés dans le tableau ci-dessous quelques tests sur des programmes extraits des travaux de N. Lindenstrauss et Y. Sagiv[2] (les temps sont donnés en secondes).

Prédicat principal	Hoarau/Mesnard		Lindenstrauss/Sagiv	
	classes inférées	temps	classes testées	temps
$append(x,y,z)$	$x \vee z$	0,47	$x \vee z$	0,51
$reverse(x,y,z)$	$(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$	0,89	$x \wedge z$	0,17
$permute(x,y)$	$x \vee y$	1,31	x	0,69
$hanoi(w,x,y,z,t)$	$(w \wedge x \wedge y \wedge z) \vee t$	53,2	$w \wedge x \wedge y \wedge z$	62,5
$fib(x,y)$	x	7,3	x	0,79
$occurall(x,y,z)$	$x \wedge y$	4,2	$x \wedge y$	2,45
$money(a,b,c,d,e,f,g,h)$	1	3,65	1	27,5
$zebra(a,b,c,d,e,f,g)$	1	3,26	1	0,58
machines	Ultra 1 SUN Station, 128 MB.		Super SPARC 51, 128 MB.	

Notons que, dans tous les cas, l'ensemble des classes de requêtes que nous *inférons* est au moins aussi grand que celui *testé* avec succés par Lindenstrauss et Sagiv. Et si les temps de calcul ne sont pas toujours en notre faveur par rapport à ceux de [LIN 97] ou [SPE 97], le problème que nous traitons est bien plus général. Notons d'autre part qu'une optimisation du contrôle du programme objet est possible : au point 3 de la méthode de construction d'un environnement clos (le calcul pour une classe donnée de l'ordre des littéraux d'une clause), nous pouvons avoir le choix entre plusieurs ordres ; il faudrait alors prendre celui qui "minimise" la taille de l'arbre de preuve. Cette voie est en cours d'exploration.

Remerciements

Nous remercions Antoine Rauzy de nous avoir fait profiter de sa connaissance et de son expérience du mu-calcul.

Bibliographie

[APT 90] K.R. APT and D. PEDRESCHI. Studies in pure Prolog: termination. In *Proceedings Esprit symposium on computational logic*, pages 150–176. Springer-Verlag, 1990.

2. il s'agit de la version avec tests de [LIN 97] disponible à l'adresse http://www.cs.huji.ac.il/∼naomil.

[COD 92] P. CODOGNET and G. FILÉ. Computations, abstractions and contraints in logic programs. In *Proc. of ICCL'92*. IEEE, 1992.

[COL 97] S. COLIN, F. MESNARD, and A. RAUZY. Constraint logic programming and mu-calculus. *ERCIM/COMPULOG Workshop on Constraints*, 1997.

[DES 93] D. DE SCHREYE and S. DECORTE. Termination of logic programs : the never-ending story. *The Journal of Logic Programming*, 12:1–66, 1993.

[GAB 91] M. GABBRIELLI and G. LEVI. Modelling answer constraints in constraint logic programs. In MIT Press, editor, *Proc. of ICLP'91*, pages 238–252, 1991.

[JAF 94] J. JAFFAR and M.J. MAHER. Constraint logic programming: a survey. *J. Logic Programming*, 19:503–581, 1994.

[LIN 97] N. LINDENSTRAUSS and Y. SAGIV. Automatic termination analysis of logic programs. *Proc. of the 14th ICLP*, pages 63–77, 1997.

[MES 96] F. MESNARD. Inferring left-terminating classes of queries for constraint logic programs by means of approximations. In *Proc. of JICSLP'96*, pages 7–21. MIT Press, 1996.

[SOH 91] K. SOHN and A. VAN GELDER. Termination detection in logic programs using argument sizes. *Proc. of PODS'91*, pages 216–226, 1991.

[SPE 97] C. SPEIRS, Z. SOMOGYI, and H. SØNDERGAARD. Termination analysis for Mercury. *Proc. of SAS'97*, pages –, 1997.