4 rue Léonard de Vinci
BP 6759
F-45067 Orléans Cedex 2
FRANCE
http://www.univ-orleans.fr/lifo

Rapport de Recherche

Un algorithme de décision dans l'algèbre des arbres finis ou infinis et des queues

Thi-Bich-Hanh Dao
LIFO, Université d'Orléans

Rapport n^o **RR-2009-02**

Un algorithme de décision dans l'algèbre des arbres finis ou infinis et des queues

Thi-Bich-Hanh Dao

27 février 2009

Résumé

Les structures des arbres, des listes, des piles et des queues sont souvent utilisées en programmation logique et programmation logique avec contraintes. Il est donc important de pourvoir résoudre ou décider des contraintes dans ces structures. Les listes et les piles peuvent être considérées comme des cas spéciaux des arbres, mais ce n'est pas le cas des queues avec les opérations d'ajout d'un élément à gauche ou à droite. Nous présentons dans ce papier un algorithme de décision dans l'algèbre des arbres finis ou infinis étendus avec des queues. De cet algorithme découle un résultat en logique : la complétude et la décidabilité de la théorie du premier ordre de cet algèbre.

The structures of trees, lists, stacks and queues are usually used in logic programming or constraint logic programming. It is then important to be capable to solve or to decide constraints in these structures. Lists and stacks can be considered as special cases of trees, but it is not the case for queues with left- and right-insert operations. We present in this paper a decision algorithm in the algebra of finite or infinite trees extended with queues. This algorithm gives a result in logic : the completeness and the decidability of the first-order theory of this algebra.

1 Introduction

Dans la programmation logique (PL) ou la programmation logique avec contraintes (PLC), la structure d'arbres et les structures de listes, de piles ou de queues sont souvent utilisées. On se trouve donc face à décider ou à résoudre des contraintes dans ces structures. Les listes et les piles avec l'opération d'ajout d'un élément à gauche pour les listes et à droite pour les piles peuvent être considérées comme des cas spéciaux des arbres. Une procédure de décision d'une algèbre des arbres s'applique donc habituellement aux arbres avec listes et piles. Les queues avec les opérations d'ajout à gauche ou à droite ne peuvent pas être considérées comme des arbres, car une queue peut être construite de différentes façons, ce qui contrarie une des propriétés principales des arbres. Dans des implantations de la PL ou la PLC, les queues sont souvent représentées par des différences de listes. Afin de préserver et de mieux exploiter les propriétés

spécifiques des queues, il est intéressant de les considérer comme une extension des arbres. Il est donc important de pouvoir décider et résoudre des contraintes dans cette extension.

La décision dans des algèbres des arbres font déjà l'objet de plusieurs études. Dans le cas des arbres finis nous référons aux travaux de A. Malcev [12], K. L. Clark [1], K. Kunen [9] et H. Comon [4]. Pour les arbres finis ou infinis, M. Maher a proposé une axiomatisation complète ainsi qu'une procédure de décision [11]. Un algorithme de résolution de contraintes du premier ordre dans les algèbres des arbres est proposé en [5, 7] et qui fait d'office la décision pour les formules closes.

Des extensions d'arbres avec d'autres structures sont étudiées dans différents travaux. Dans le cas d'extension de la théorie des arbres avec une théorie flexible, par exemple la théorie des rationnels additifs, une axiomatisation ainsi qu'une procédure de décision sont proposées [6]. Une procédure de décision dans l'algèbre des termes avec des contraintes d'entiers est proposée en [15].

Nous considérons dans ce papier l'algèbre des arbres finis ou infinis étendus avec des queues, définis sur un ensemble infini de symboles de fonction. Nous nous intéressons à cet algèbre car les arbres avec un ensemble infini de symboles de fonction modélisent la base de la PL ou de la PLC [3, 8]. L'extension avec des queues n'est pas simplement d'avoir des queues en plus des arbres, mais une structure où un arbre peut avoir une partie qui est une queue, ou les éléments d'une queue peuvent être des arbres ou de nouveau des queues. La théorie des queues n'étant pas flexible, le cadre de [6] ne peut s'appliquer pour déduire une procédure de décision dans l'algèbre. La combinaison des arbres et des queues étant fine et étroite, les méthodes générales de décision dans une combinaison de théories comme Nelson-Oppen [13] ne peuvent non plus s'appliquer. T. Rybina et A. Voronkov ont proposé un algorithme de décision dans l'algèbre des arbres finis avec queues [14]. Cependant l'algèbre étudiée est des arbres finis construits sur un ensemble fini de symboles de fonction, ce qui fait que des techniques utilisées pour la décision ne peuvent pas s'appliquer dans notre cas.

La contribution de ce papier est un algorithme de décision des formules du premier ordre dans cette algèbre. L'algorithme permet à conclure la décidabilité de la théorie du premier ordre de cette algèbre.

Le reste du papier est organisé comme suit. L'algèbre des arbres avec queues ainsi que ses propriétés sont présentées dans la section 2. Dans la section 3, nous présentons les formes résolue et basique et l'algorithme de transformation en forme résolue. Dans la section 4, nous présentons l'algorithme de décision dans l'algèbre et déduisons la décidabilité de la théorie du premier ordre de cette algèbre. Plusieurs exemples sont donnés pour illustrer l'algorithme. La section 5 est réservée à la conclusion.

2 L'algèbre des arbres avec queues

Langage Soit S un ensemble de deux *sortes arbre* et *queue*. Soit V un ensemble de variables dont chacune est associée à une sorte fixée. On suppose que

chaque sorte a un nombre infini de variables. On appelle un *type* chaque expression de la forme $\alpha_1 \times \cdots \times \alpha_n \to \beta$, où $\alpha_1, \ldots, \alpha_n$ et β sont des sortes de S. Soit F un ensemble infini de symboles de fonction. A chaque élément de F est associé un type $\alpha_1 \times \cdots \times \alpha_n \to \mathit{arbre}$ et le nombre n est appelé l'*arité* de f. On se donne trois autres symboles al, ar et ε où al est de type $\alpha \times \mathit{queue} \to \mathit{queue}$, ar est de type $\mathit{queue} \times \alpha \to \mathit{queue}$ et ε de type $\to \mathit{queue}$, avec $\alpha \in S$. Soit R un ensemble de symboles de relation contenant deux symboles *queue* et *arbre* d'arité 1.

Un *terme* de sorte β est soit une variable de cette sorte, soit de la forme $f(u_1, ..., u_n)$ où $f \in F$ de type $\alpha_1 \times \cdots \times \alpha_n \to \beta$ et $u_1, \ldots, u_n$ des termes de sortes $\alpha_1, \ldots, \alpha_n$ respectivement. Une *formule* est une expression de l'une des formes suivantes :

$$s = t,\ \mathit{arbre}(x),\ \mathit{queue}(t),\ \mathit{true},\ \mathit{false},$$
$$\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi), \exists x\varphi, \forall x\varphi.$$

Ici t, s sont des termes et φ, ψ sont des formules de taille plus petite. Nous utiliserons la notation $\overline{x}$ pour désigner un vecteur de variable $x_1...x_n$, $\exists\overline{x}$ pour $\exists x_1...\exists x_n$ et $\forall\overline{x}$ pour $\forall x_1..\forall x_n$. Nous utiliserons la notation $[u_1..u_n]t$ ou $[\overline{u}]t$ pour désigner le terme $al(u_1, ..., al(u_n, t)....)$ et la notation $t[u_1..u_n]$ ou $t[\overline{u}]$ pour le terme $ar(...ar(t, u_1)..., u_n)$. Nous utiliserons la notation $\exists?$ et $\exists!$ pour exprimer "'il existe au plus une valeur" et "il existe exactement un", respectivement.

Structure $\mathcal{S}$ Nous définissons la structure $\mathcal{S}$, appelée l'*algèbre des arbres finis ou infinis avec queues* comme suit. Le domaine de $\mathcal{S}$ est l'ensemble $D = A \cup Q$, où A est l'ensemble des arbres, qui sont finis ou infinis, dont les nœuds sont étiquetés par les éléments de F, et Q l'ensemble des queues d'éléments de $A \cup Q$. Une queue est essentiellement une liste, à laquelle nous pouvons ajouter un élément au début (opération standard de construction de liste) ou à la fin. Les arbres et les queues représentent deux sortes *arbre* et *queue* respectivement. Nous définissons maintenant l'interprétation de chaque symbole du langage.

Pour chaque symbole de fonction f de type $\alpha_1 \times \cdots \times \alpha_n \to \mathit{arbre}$, l'interprétation $\underline{f}$ de f est l'opération de construction d'arbre défini comme suit : avec $a_1, \ldots, a_n$ des éléments de sortes $\alpha_1, \ldots, \alpha_n$, $\underline{f}(a_1, \ldots, a_n)$ est un arbre dont la racine est étiquetée par f et les fils immédiats sont $a_1, \ldots, a_n$. Le symbole ε est interprété par la queue vide. Le symbole al est interprété par la fonction $\underline{al} : (A \cup Q) \times Q \to Q$ telle qu'avec $a \in A \cup Q$, $\langle b_1, .., b_n\rangle \in Q$, $\underline{al}(a, \langle b_1, .., b_n\rangle) = \langle a, b_1, .., b_n\rangle$. Le symbole ar est interprété par la fonction $\underline{ar} : Q \times (A \cup Q) \to Q$ telle qu'avec $\langle b_1, .., b_n\rangle \in Q$, $a \in A \cup Q$, $\underline{ar}(\langle b_1, .., b_n\rangle, a) = \langle b_1, .., b_n, a\rangle$.

Les symboles de relation *arbre* et *queue* sont interprétés par les relations d'appartenance à A et à Q respectivement.

Propriétés des arbres Les arbres satisfont l'ensemble des axiomes suivants [11, 7] :

$$\forall\overline{x}\forall\overline{y}\ \neg(f(\overline{x}) = g(\overline{y})) \qquad (a1)$$
$$\forall\overline{x}\forall\overline{y}\ (f(\overline{x}) = f(\overline{y}) \to \textstyle\bigwedge_i x_i = y_i) \qquad (a2)$$
$$\forall\overline{x}\exists!\overline{z}\ \textstyle\bigwedge_i z_i = t_i(\overline{xz}) \qquad (a3)$$

où $f, g \in F$ et sont distincts, $\overline{x}$ un vecteur de variables x_i, $\overline{y}$ un vecteur de variable y_i, $\overline{z}$ un vecteur de variables distinctes z_i de sorte *arbre* et $t_i(\overline{x}\overline{z})$ un terme de la forme d'un symbole de fonction suivi de variables prises dans $\overline{x}$ et $\overline{z}$.

Propriétés des queues Les queues satisfont les propriétés suivantes [14] :

$$\begin{array}{ll}
\forall x \forall \overline{u} \neg (x = t(x\overline{u})) & (q1) \\
\forall x \forall u (\neg(\varepsilon = [u]x) \wedge \neg(\varepsilon = x[u])) & (q2) \\
\forall u_1..\forall u_n \forall v_1..\forall v_m \forall x \neg([u_1..u_n]x = x[v_1..v_m]) & (q3) \\
\forall x (x = \varepsilon \vee \exists y \exists u (x = [u]y) \vee \exists y \exists u (x = y[u])) & (q4) \\
\forall x \forall y \forall u \forall v ([u]x = [v]y \rightarrow u = v \wedge x = y) & (q5) \\
\forall x \forall y \forall u \forall v (x[u] = y[v] \rightarrow u = v \wedge x = y) & (q6) \\
\forall x \forall u \forall v (al(u, ar(v, x)) = ar(al(u, x), v)) & (q7) \\
\forall u_1 ... \forall u_n ([u_1..u_n]\varepsilon = \varepsilon[u_1..u_n]) & (q8)
\end{array}$$

Ici x, y sont des variables de sorte *queue*, $t(x\overline{u})$ est un terme de sorte *queue* qui contient des occurrences de x. La propriété $(q1)$ exprime le fait que les queues sont de longueur finie et une queue ne peut se servir à définir elle-même, $q(2)$ et $(q3)$ le fait que les queues de longueurs différentes sont différentes, et de $(q4)$ à $(q8)$ la relation entre les ajouts à gauche et à droite dans une queue.

De plus, elles satisfont les propriétés suivantes concernant des mots. Les queues peuvent être considérées comme des mots finis sur D. Soit D^* l'ensemble des mots finis, D^+ l'ensemble des mots finis non vides sur D et ε le mot vide. Pour deux mots u et v, uv signifie le résultat de la concaténation du mot u avec le mot v. Pour un mot u, $|u|$ désigne sa longueur et u^n le résultat de la concaténation de n fois u, u^* (u^+) désigne l'ensemble $\{u^n | n \geq 0\}$ ($\{u^n | n > 0\}$). Un mot u est primitif si $u \neq v^n$ avec $n > 1$ pour tout mot v. Deux mots u et v sont conjugués s'il existe deux mots s, t avec $s \neq \varepsilon$ tels que $u = ts$ et $v = st$. Par exemple sur l'alphabet $\{a, b\}$, deux mots *baaba* et *ababa* sont conjugués.

Lemme 1 [10] *Soient $x, y \in D^*$ et z, t des mots primitifs tels que $x \in t^*$ et $y \in s^*$. On a x et y sont conjugués si et seulement si t et s sont aussi conjugués, en ce cas il existe un couple unique $(u, v) \in D^* \times D^+$ tel que $t = uv$ et $s = vu$.*

Propriété 1 [10] *Soient $t, s \in D^+$ et x une variable. L'équation $tx = xs$ a des solutions si et seulement si t et s sont conjugués, c'est-à-dire qu'il existe un couple $(u, v) \in D^* \times D^+$ tel que $t = uv$ et $s = vu$. De plus, si t et s sont primitifs alors $(uv)^*u$ est l'ensemble des solutions de l'équation $tx = xs$.*

Preuve : Si t et s sont conjugués, il existe $(u, v) \in D^* \times D^+$ tel que $t = uv$ et $s = vu$. Les mots de l'une des formes $(uv)^*u$ sont effectivement des solutions de l'équation $tx = xs$. Inversement, si x est une solution de l'équation $tx = xs$, on a pour chaque $n \geq 1$

$$t^n x = x s^n$$

Soit n tel que $n|t| > |x| \geq (n-1)|t|$. De la formule ci-dessus on déduit

$$x = t^{n-1}u, \quad t = uv, \quad vx = s^n, \text{ avec } |u| \geq 0.$$

Finallement $s^n = vx = vt^{n-1}u$ et est aussi égal à $(vu)^n$ et du fait que $|s| = |t|$ on obtient $s = vu$. Les mots t et s sont donc conjugués et du fait qu'ils sont primitifs, le couple (u, v) est unique, donc les solutions sont de la forme $(uv)^*u$. CQFD.

Lemme 2 [10] *Soient $t, s \in D^+$. Si $ts = st$ alors il existe $u \in D^+$ tel que $t \in u^*$ et $s \in u^*$.*

La preuve des deux propriétés 2 et 3 suivantes est inspirée du papier de T. Rybina et A. Voronkov [14].

Propriété 2 *Soit $t = uv$ et $t' = v'u'$ deux mots primitifs dans D^+ avec $|t| \neq |t'|$. On a*

$$uvx = xvu \wedge u'v'x = xv'u' \rightarrow |x| < |t| + |t'|.$$

Preuve : Soient $n = |t|$ et $m = |t'|$. Soit x un mot tel que $uvx = xvu \wedge u'v'x = xv'u'$ et supposons que $|x| \geq n + m$. Sans perdre généralité, supposons que $m > n$. Le mot x doit être des deux formes $(uv)^*u$ et $(u'v')^*u'$.

Soit $r = m \bmod n$. Puisque $|x| \geq n+m$, les $n+m$ premiers éléments doivent être coïncident

$$t'_1...t'_m t'_1...t'_n = t_1...t_n...t_1...t_r$$

d'où $t'_1...t'_n = t_1...t_n$ et donc $t_1...t_n t'_{n+1}...t'_m t_1...t_n = t_1...t_n...t_1...t_r$. Considérant les n derniers éléments des deux côtés

$$t_1...t_n = t_{r+1}...t_n t_1...t_r$$

D'après le lemme 2, soit $r = 0$ soit il existe $s \in A^+$ tel que $t_1...t_r, t_{r+1}...t_n \in s^*$. Puisque $t_1...t_n$ est un mot primitif, on déduit $r = 0$. En conséquent n divise m donc $t'_1...t'_m$ n'est pas un mot primitif, contradiction. CQFD.

Propriété 3 *Soit $t = uv$ et $t' = u'v'$ deux mots primitifs dans D^+ avec $|t| = |t'|$. Si $|u| = |v|$ alors*

$$\begin{array}{ll} & uvx = xvu \wedge u'v'x = xv'u' \\ \rightarrow & (u = u' \wedge x = u) \vee (u = u' \wedge v = v' \wedge uvx = xvu) \end{array}$$

sinon $uvx = xvu \wedge u'v'x = xv'u' \rightarrow$ false.

Preuve : Soient $t = t_1...t_n$, $t' = t'_1...t'_n$ et $k = |u|$, $l = |v|$. Supposons que $k = l$. Soit x un mot tel que $t_1...t_n x = xt_{k+1}...t_n t_1...t_k \wedge t'_1...t'_n x = xt'_{k+1}...t'_n t'_1...t'_k$. Le mot x est donc des forme $(t_1...t_n)^r t_1...t_k$ et $(t'_1...t'_n)^r t'_1...t'_k$. Si $r = 0$ alors $x = t_1...t_k$ et $t_i = s_i$ pour tout $1 \leq i \leq k$. Si $r > 0$ alors $(t_1...t_n)^r t_1...t_k = (t'_1...t'_n)^r t'_1...t'_k$, d'où $t_i = t'_i$ pour tout $1 \leq i \leq n$.

Considérons le cas où $k \neq l$. Puisque $t_1...t_n$ et $t'_1...t'_n$ sont primitifs, d'après le lemme 1, k et l sont uniques. Aucun mot de la forme $(t_1...t_n)^* t_1...t_k$ n'est donc de la forme $(t'_1..t'_n)^* t'_1...t'_l$ et vice versa. La conjonction $t_1...t_n x = xt_{k+1}...t_n t_1...t_k \wedge t'_1...t'_n x = xt'_{l+1}...t'_n t'_1...t'_l$ n'a donc pas de solution. CQFD.

3 Formules basiques

A partir d'ici nous supposons que les variables soient numérotées par des entiers naturels distincts, et pour une variable x soit $no(x)$ son numéro. Nous supposons que dans chaque formule et sous-formule, si x est une variable libre et y une variable quantifiée, alors $no(y) > no(x)$. Cette condition est satisfaisable car les variables quantifiées peuvent être renommées et l'ensemble de variables est infini.

En utilisant la propriété ($q7$) des queues, les termes construits avec les symboles al, ar sont réécrits dans la forme où les symboles les plus internes sont ar et ceux des plus externes sont al. Par exemple le terme $ar(ar(al(u_1, x), u_2), u_3)$ devient $al(u_1, ar(ar(x, u_2), u_3))$. La notation $[t_1..t_n]x[s_1..s_m]$ désignera ces termes et la variable x sera appelée le *noyau* du terme. Soit $\overline{u} = u_1..u_n$ et $1 \leq k \leq n$, on note par $\rho(k, \overline{u})$ la rotation circulaire à droite de k places de $\overline{u}$, c'est-à-dire $u_{k+1}..u_n u_1..u_k$. Notons que $\overline{u}$ et $\rho(k, \overline{u})$ sont conjugués.

Une *contrainte élémentaire* est

- soit une contrainte de sorte de la forme $queue(x)$ ou $arbre(x)$, avec x une variable,
- soit une équation de la forme $x = y$, avec x, y des variables,
- soit une équation de la forme $x = f(u_1, .., u_n)$ avec x et les u_i des variables et f de type $\alpha_1 \times \cdots \times \alpha_n \rightarrow arbre$,
- soit une équation de l'une des formes $x = t$ et $t = s$, où x est une variable, t et s sont des termes de l'une des formes ε, $[\overline{u}]\varepsilon[\overline{v}]$ et $[\overline{u}]y[\overline{v}]$, avec y une variable et $\overline{u}, \overline{v}$ des vecteurs de variables.

Une équation triviale est une équation de la forme $t = t$. Pour simplifier l'écriture, les équations de la forme $x[\overline{u}] = [\overline{v}]y$ sont réorganisées en $[\overline{v}]y = x[\overline{u}]$. Pour les équations de la forme $x = t$ ou $[\overline{u}]x = x[\overline{v}]$, x est appelée le *représentant* de l'équation.

Soit c une conjonction de contraintes élémentaires et soit u une variable. On définit $Acc_c(u)$ l'ensemble des variables accessibles depuis u dans c comme suit :

- si c contient une équation non triviale de la forme $u = t$ ou $x = t$ avec $x \in Acc_c(u)$, alors les variables dans t sont dans $Acc_c(u)$,
- si c contient une équation non triviale de la forme $[\overline{u}]u = u[\overline{v}]$ ou $[\overline{u}]x = x[\overline{v}]$ avec $x \in Acc_c(u)$, alors les variables de $\overline{u}$ et $\overline{v}$ sont dans $Acc_c(u)$.

Une conjonction de contraintes élémentaires est *complètement typée* si pour toute variable x ayant une occurrence dans c, soit $queue(x)$ soit $arbre(x)$ est dans c.

Définition 1 *Une conjonction c de contraintes élémentaires est sous* forme résolue *si et seulement si*

1. *chaque équation est de l'une des formes $x = t$, $[\overline{u}]x = x[\rho(k, \overline{u})]$, où x est une variable et $k \leq |\overline{u}|$,*
2. *les représentants des équations sont tous distincts,*
3. *pour chaque équation de la forme $x = y$, $no(x) > no(y)$,*
4. *c est complètement typée et les contraintes de sorte respectent le type des symboles de fonction dans c,*

5. *pour toute variable x de sorte queue ayant une occurrence dans c, $x \notin Acc_c(x)$.*

Propriété 4 *Soit c une conjonction résolue et soit $\overline{x}$ un vecteur des variables représentants des équations de c. On a $\mathcal{S} \models \exists \overline{x} c$.*

Preuve : D'après la propriété 1, chaque équation dans c de la forme $[\overline{u}]x = x[\rho(k, \overline{u})]$ a des solutions qui peuvent s'écrire sous la forme $x = t$, où x ne figure pas dans t. Soit c' la conjonction obtenu en remplaçant dans c chaque équation de la forme $[\overline{u}]x = x[\rho(k, \overline{u})]$ par une équation de la forme $x = t$, où t est une solution. Pour prouver $\mathcal{S} \models \exists \overline{x} c$ il suffit de prouver $\mathcal{S} \models \exists \overline{x} c'$. La conjonction c' ne contient que des équations de la forme $x = t$. Nous répartissons c' en deux conjonctions c'_1 et c'_2 qui contiennent tous les équations et contraintes de sorte *arbre* et de sorte *queue* de c, respectivement. Soient $\overline{x_1}$ et $\overline{x_2}$ des vecteurs des représentants de c'_1 et c'_2, respectivement. On a donc $\overline{x} = \overline{x_1 x_2}$. Puisque pour chaque variable x de sorte *queue*, on a $x \notin Acc_c(x)$, pour chaque équation $x = t$ de c'_2, on peut remplacer toute autre occurrence de x par t. La conjonction c'_2 est donc de la forme $\bigwedge_i x_i = t_i$, où x_i n'a aucune autre occurrence dans c'_1 et c'_2. On a donc $\mathcal{S} \models \exists \overline{x_2} c'_2$. D'après la propriété $(a3)$ des arbres, on a $\mathcal{S} \models \exists \overline{x_1} c'_1$. Puisque $\overline{x_1}$ et $\overline{x_2}$ sont disjoints, on conclut donc $\mathcal{S} \models \overline{x_1 x_2}(c_1 \wedge c_2)$, d'où $\mathcal{S} \models \exists \overline{x} c$. CQFD.

Par les propriétés $(a2)$ des arbres et $(q5)$, $(q6)$ des queues, une équation de la forme $x = t(\overline{u})$ ou $[\overline{u}]x = x[\rho(k, \overline{u})]$, avec u figurant dans $\overline{u}$, entraîne que u est déterminée d'une façon unique par rapport à x. Nous avons donc la propriété suivante :

Propriété 5 *Soit $\exists \overline{u} c$ avec c une conjonction résolue. Si pour chaque variable u de $\overline{u}$ il existe une variable libre x telle que $u \in Acc_c(x)$, alors $\mathcal{S} \models \exists ? \overline{u} c$.*

Propriété 6 *Soit T une théorie quelconque et soit p, q des formules. Si $T \models \exists ? \overline{u} p$ alors*

$$T \models \exists \overline{u}(p \wedge \neg q) \equiv \exists \overline{u} p \wedge \neg \exists \overline{u}(p \wedge q).$$

La preuve de cette propriété se trouve dans [5]. Du fait qu'elle est valide pour une théorie T quelconque, elle est valide pour la structure $\mathcal{S}$.

Définition 2 *Une* formule basique *est de la forme $\exists \overline{u}(c \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i)$, avec I éventuellement vide, où*

1. *c et les c_i sont des conjonctions résolues,*
2. *pour chaque variable u de $\overline{u}$, il existe une variable x libre dans $\exists \overline{u} c$ telle que $u \in Acc_c(x)$,*
3. *pour chaque variable u de $\overline{u_i}$, il existe une variable x libre dans $\exists \overline{u_i} c_i$ telle que $u \in Acc_{c_i}(x)$.*

Remarque : puisque chaque variable quantifiée dans une formule basique doit être accessible depuis une variable libre, la seule formule basique sans variable libre est la formule *true*.

3.1 Algorithme de transformation en forme résolue

Nous utiliserons la notation d'équation $[\overline{u}]x \stackrel{p}{=} x[\overline{v}]$ pour marquer que les mots $\overline{u}$ et $\overline{v}$ sont primitifs et la notation $[\overline{u}]x \stackrel{*}{=} x[\overline{v}]$ pour marquer que les mots $\overline{u}$ et $\overline{v}$ sont conjugués et primitifs.

L'algorithme est une combinaison de deux phases. La première phase transforme une conjonction de contraintes élémentaires complètement typée de sorte que les représentants des équations $x = t$ sont tous distincts. A la fin de cette phase, les représentants des équations de la forme $[\overline{u}]x = x[\overline{v}]$ ne sont pas encore distincts. La seconde phase traite ces équations, elle créera une disjonction de formules de la forme $\exists \overline{u} c$, où éventuellement sur c il sera nécessaire de relancer la phase 1 et la phase 2. Nous présentons dans ce qui suit les deux phases et montrons que bien que les deux phases puissent se lancer mutuellement, l'applications se termine toujours au bout d'un temps fini.

Première phase

- Concordance de type : assurer que le type des symboles de fonction est respecté, et qu'il n'y a pas de conflit de type.
- Equations de sorte *arbre* : Dans ces règles, a est une contrainte élémentaire, x, y sont des variables de sorte *arbre* avec $no(x) > no(y)$, les u_i, v_j des variables, t un terme de sorte *arbre*, f et g deux symboles de fonction différents.

$$\begin{array}{rcl}
\mathit{false} \wedge a & \Longrightarrow & \mathit{false} \\
x = x & \Longrightarrow & \mathit{true} \\
x = y & \Longrightarrow & y = x \\
x = f(u_1, .., u_n) \wedge x = g(v_1, .., v_m) & \Longrightarrow & \mathit{false} \\
x = f(u_1, .., u_n) \wedge x = f(v_1, .., v_n) & \Longrightarrow & x = f(u_1, .., u_n) \wedge \\
 & & u_1 = v_1 \wedge \cdots \wedge u_n = v_n \\
x = y \wedge x = t & \Longrightarrow & x = y \wedge y = t
\end{array}$$

- Equations de sorte *queue* : dès qu'il existe une variable x de sorte *queue* dans c telle que $x \in Acc_c(x)$, la conjonction est évaluée à *false*.

$$\begin{array}{lrcl}
1 & \mathit{false} \wedge a & \Longrightarrow & \mathit{false} \\
2 & t = t & \Longrightarrow & \mathit{true} \\
3 & t = x & \Longrightarrow & x = t \\
4 & y = x & \Longrightarrow & x = y \\
5 & x = y \wedge e(x) & \Longrightarrow & x = y \wedge e(y) \\
6 & x = t \wedge x = s & \Longrightarrow & x = t \wedge t = s \\
7 & x = t \wedge e(x) & \Longrightarrow & x = t \wedge e(t) \\
8 & [u]t = [v]s & \Longrightarrow & u = v \wedge t = s \\
9 & t[u] = s[v] & \Longrightarrow & u = v \wedge t = s \\
10 & [\overline{u}]\varepsilon = r[\overline{v}] & \Longrightarrow & \varepsilon[\overline{u}] = r[\overline{v}] \\
11 & [\overline{u}]x = \varepsilon[\overline{v}] & \Longrightarrow & [\overline{u}]x = [\overline{v}]\varepsilon \\
12 & \varepsilon = [\overline{u}]x[\overline{v}] & \Longrightarrow & \mathit{false} \\
13 & [\overline{u}]x[\overline{v}] = \varepsilon & \Longrightarrow & \mathit{false} \\
14 & [\overline{u}]x = x[\overline{v}] & \Longrightarrow & \mathit{false}
\end{array}$$

Ici a est une contrainte élémentaire, t, s sont des termes de sorte *queue*, x, y des variables de sorte *queue*, u, v des variables et $\overline{u}, \overline{v}$ des vecteurs de variables. Dans les règles 3, 6 et 7, t, s sont des termes qui ne sont pas une variable. Dans les règles 4 et 5, $no(x) > no(y)$. Dans la règle 5, $e(x)$ est une équation ayant une occurrence de x et $e(y)$ obtenue de $e(x)$ en remplaçant les occurrences de x par y. Dans la règle 7, $e(x)$ est une équation dans laquelle x est le noyau d'un terme et $e(t)$ l'équation obtenue en remplaçant le noyau x par t, et si $e(x)$ est écrite avec $\stackrel{*}{=}$, $e(t)$ est écrite avec $=$. Dans la règle 10, r est soit ε, soit une variable de sorte *queue*. Dans la règle 14, $|\overline{u}| \neq |\overline{v}|$.

Deuxième phase Cette phase s'effectue sur une conjonction c. Selon la forme des équations dans c, les règles suivantes s'appliquent. Lorsqu'il y a une disjonction qui se créée, la disjonction est distribuée pour maintenir la formule sous forme disjonctive $\bigvee_i \exists u_i c'_i$. La phase 1 est relancée sur des conjonctions c'_i.

1. Une équation $[u_1..u_{n+k}]x = y[v_1..v_n]$ avec x, y des variables distinctes, est remplacée par

$$\begin{array}{c} \exists u(y = [u_1..u_{n+k}]u \wedge x = u[v_1..v_n] \wedge queue(u)) \\ \bigvee_{i=0}^{n-1} \left(\begin{array}{l} y = [u_1..u_{k+i}]\varepsilon \wedge x = [v_{n-i+1}..v_n]\varepsilon \\ \wedge v_1 = u_{k+i+1} \wedge \cdots \wedge v_{n-i} = u_{n+k} \end{array} \right) \end{array}$$

   idem pour $[u_1..u_n]x = y[v_1..v_{n+k}]$.

2. Equation $[u_1..u_n]x = x[v_1..v_n]$: Pour chaque couple i, j avec $1 \leq i \neq j \leq n$, on ajoute $(u_i = u_j \wedge v_i = v_j) \vee \neg(u_i = u_j) \vee \neg(v_i = v_j)$. Distribuer pour créer une disjonction de conjonction d'équations et de diséquations. Pour chaque conjonction on peut déterminer si $u_1..u_n$ est une répétition de $u_1..u_k$.
   - dans les cas où $u_1..u_n$ est de la forme $(u_1..u_k)^{n/k}$ avec $u_1..u_k$ primitif, remplacer l'équation $[u_1..u_n]x = x[v_1..v_n]$ par $[u_1..u_k]x \stackrel{p}{=} x[v_1..v_k]$,
   - dans les autres cas, remplacer l'équation $[u_1..u_n]x = x[v_1..v_n]$ par $[u_1..u_n]x \stackrel{p}{=} x[v_1..v_n]$.

3. Equation $[\overline{u}]x \stackrel{p}{=} x[\overline{v}]$, avec $|\overline{u}| = |\overline{v}| = n$, remplacée par

$$\bigvee_{k=1}^{n} ([\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\rho(k, \overline{u})]\varepsilon = [\overline{v}]\varepsilon)$$

4. Deux équations $\stackrel{*}{=}$ sur une même variable x, avec $|\overline{u}| > |\overline{v}|$:

$$\begin{array}{l} [\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(l, \overline{v})] \Longrightarrow \\ \bigvee_{i \in I}(x = [\overline{u}^i u_1..u_k]\varepsilon \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(l, \overline{v})]) \end{array}$$

   Ici $I = \{0\}$ si $k \geq |\overline{v}|$ et $I = \{0, 1\}$ sinon.

5. Deux équations $\stackrel{*}{=}$ sur une même variable x, avec $|\overline{u}| = |\overline{v}|$ et $k \neq l$:

$$[\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(l, \overline{v})] \Longrightarrow \mathit{false}$$

6. Deux équations $\stackrel{*}{=}$ sur une même variable x, avec $|\overline{u}| = |\overline{v}|$:

$$\begin{array}{l}[\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(k, \overline{v})] \Longrightarrow \\ \left(\begin{array}{l}(\bigwedge_{i=1}^{k} u_i = v_i \wedge x = [u_1 ... u_k]\varepsilon) \vee \\ (\bigwedge_{i=1}^{|\overline{u}|} u_i = v_i \wedge [\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})])\end{array}\right)\end{array}$$

Exemple 1 *Transformer la conjonction ($\stackrel{i}{\equiv}$ signifie la transformation par la phase i) :*

$$\begin{array}{l}\left(\begin{array}{l}x = [u]y \wedge x = z[v] \wedge u = f(v) \wedge \mathit{queue}(x) \\ \wedge \mathit{queue}(y) \wedge \mathit{queue}(z) \wedge \mathit{arbre}(u) \wedge \mathit{arbre}(v)\end{array}\right) \\ \stackrel{1}{\equiv}\left(\begin{array}{l}x = [u]y \wedge [u]y = z[v] \wedge u = f(v) \wedge \mathit{queue}(x) \\ \wedge \mathit{queue}(y) \wedge \mathit{queue}(z) \wedge \mathit{arbre}(u) \wedge \mathit{arbre}(v)\end{array}\right) \\ \stackrel{2}{\equiv}\left(\begin{array}{l}x = [u]y \wedge u = f(v) \wedge \mathit{queue}(x) \\ \wedge \mathit{queue}(y) \wedge \mathit{queue}(z) \wedge \mathit{arbre}(u) \wedge \mathit{arbre}(v) \\ \wedge \left(\begin{array}{l}\exists w(y = w[v] \wedge z = [u]w \wedge \mathit{queue}(w)) \\ \vee (y = \varepsilon \wedge z = \varepsilon \wedge u = v)\end{array}\right)\end{array}\right) \\ \equiv\left(\begin{array}{l}\exists w \left(\begin{array}{l}x = [u]y \wedge u = f(v) \wedge y = w[v] \wedge z = [u]w \\ \wedge \mathit{queue}(w) \wedge \mathit{queue}(x) \wedge \mathit{queue}(y) \\ \wedge \mathit{queue}(z) \wedge \mathit{arbre}(u) \wedge \mathit{arbre}(v)\end{array}\right) \\ \vee \left(\begin{array}{l}x = [u]y \wedge u = fv \wedge y = \varepsilon \wedge z = \varepsilon \wedge u = v \\ \wedge \mathit{queue}(x) \wedge \mathit{queue}(y) \wedge \mathit{queue}(z) \\ \wedge \mathit{arbre}(u) \wedge \mathit{arbre}(v)\end{array}\right)\end{array}\right) \\ \stackrel{1}{\equiv}\left(\begin{array}{l}\exists w \left(\begin{array}{l}x = [u]w[v] \wedge u = fv \wedge y = w[v] \wedge z = [u]w \\ \wedge \mathit{queue}(w) \wedge \mathit{queue}(x) \wedge \mathit{queue}(y) \\ \wedge \mathit{queue}(z) \wedge \mathit{arbre}(u) \wedge \mathit{arbre}(v)\end{array}\right) \\ \vee \left(\begin{array}{l}x = [u]\varepsilon \wedge u = v \wedge y = \varepsilon \wedge z = \varepsilon \wedge v = f(v) \\ \wedge \mathit{queue}(x) \wedge \mathit{queue}(y) \wedge \mathit{queue}(z) \\ \wedge \mathit{arbre}(u) \wedge \mathit{arbre}(v)\end{array}\right)\end{array}\right)\end{array}$$

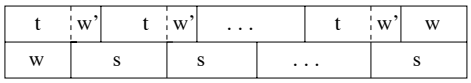
Théorème 1 *Soit c une conjonction de contraintes élémentaires complètement typée. L'application des 2 phases autant que possible sur c se termine et produit une formule équivalente qui est soit* false *soit de la forme $\bigvee_i \exists \overline{x_i}(c_i \wedge \bigwedge_j \neg e_{ij})$, où les c_i sont des conjonctions résolues, les e_{ij} sont des équations entre deux variables et chaque variable de $\overline{x_i}$ est accessible depuis une variable libre dans c_i.*

Avant de prouver ce théorème, nous prouvons le lemme suivant.

Lemme 3 *L'ensemble de solutions de l'équation $tx = xs$ et l'ensemble de solutions de l'équation $t^n x = xs^n$ sont identiques.*

Preuve : Supposons que chaque équation soit soluble. Il est évident qu'une solution de l'équations $tx = xs$ est solution de l'équation $t^n x = xs^n$. Montrons l'autre direction.

L'équation $t^n x = xs^n$ étant soluble, il existe les mots u, v tels que $uv = t^n$ et $vu = s^n$. On a $|u| < |t^n|$ donc $u = t^k w$ avec $|w| < |t|$. Le mot u étant une solution, on a $t^n w = ws^n$ donc w est aussi une solution. Accordant au schéma suivant :



on déduit que $t = ww'$ et $s = w'w$. En conséquence w est aussi une solution de l'équation $tx = xs$, d'où u est aussi solution de cette équation, d'où $(uv)^* u$ sont aussi solutions de cette équation. Ceci est vrai pour tout u, v tels que $uv = t^n$ et $vu = s^n$. On en conclut que toute solution de l'équation $t^n x = xs^n$ est aussi une solution de l'équation $tx = xs$. CQFD.

Preuve du théorème 1 Prouvons que (1) l'application des règles se termine toujours et (2) les règles produisent une formule équivalente ayant les propriétés dans l'énoncé.

Preuve de terminaison : L'application des règles de la première phase se termine. Prouvons pour les règles pour les équations de sorte *arbre*. Nous définissons 3 entiers non négatifs : n_1 le nombre d'équations de la forme $x = f(u_1, .., u_n)$, n_2 la somme des $no(x)$ pour toute occurrence de toute variable x et n_3 le nombre d'équations de la forme $y = x$ avec $no(x) > no(y)$. Pour chaque règle il existe un entier i tel que l'application de la règle diminue n_i et laisse inchangé les n_j avec $j < i$. Cet entier vaut 1 pour les règles 4 et 5, vaut 2 pour les règles 1, 2, 6 et vaut 3 pour la règle 3. Puisque les n_i sont non négatifs, les règles ne peuvent pas s'appliquer infiniment.

Il reste à le prouver pour les règles d'équations de sorte *queue*. Lorsqu'il existe des sous-ensembles d'équations circulaires, la conjonction est évaluée à *false* et l'application des règles s'arrête, nous prouvons que l'application s'arrête aussi lorsqu'il n'y a pas de sous-ensemble circulaire. Dans ce cas, il existe une hiérarchie suivant les constructions de queues telle que si $x = [\overline{u}]y[\overline{v}]$, alors x est de rang supérieur à y. Puisqu'aucune nouvelle variable est ajoutée, nous pouvons associer à chaque rang un numéro qui respecte la hiérarchie, donc à chaque variable x un numéro $rang(x)$. Nous définissons 6 entiers non négatifs : n_1 est la somme des $rang(x)$ pour toute occurrences de toute variable x, n_2 est la somme des $no(x)$ pour toute occurrence de toute variable x, n_3 la sommes $|t|+|s|$ pour toute équation $t = s$ avec t, s des termes qui ne sont pas des variables, n_4 le nombre d'équations de la forme $[\overline{u}]x = y[\overline{v}]$, n_5 le nombre d'équations de la forme $y = x$ avec $no(x) > no(y)$ et n_6 le nombre d'équations de la forme $t = x$ avec t un terme qui n'est pas une variable. Pour chaque règle il existe un entier i tel que l'application de la règle diminue le nombre n_i et laisse inchangé les n_j, avec $j < i$. Cet entier vaut 1 pour les règles 6, 7, vaut 2 pour les règles 1, 2, 5, 12, 13, 14 vaut 3 pour les règles 2, 8, 9, vaut 4 pour les règles 10, 11, vaut 5

pour la règle 4 et vaut 6 pour la règle 3. Puisque les n_i sont non négatifs, les règles ne peuvent pas s'appliquer infiniment.

Les règles de la deuxième phase se terminent, car chaque règle diminue le nombre d'équations de la forme $[\overline{u}]x = x[\overline{v}]$.

Toute combinaison de deux phases se termine. la phase 2 n'est applicable qu'avec la présence d'équations des formes $[\overline{u}]x = y[\overline{v}]$, $[\overline{u}]x = x[\overline{v}]$ ou au moins deux équations de la forme $[\overline{u}]x \stackrel{*}{=} x[\overline{v}]$ sur une même variable x. A chaque application, la phase 2 réduit le nombre de ces équations. Donc une combinaison des deux phases n'est infinie que lorsqu'il existe une création infinie d'équations de ces formes, c'est-à-dire lorsque la phase 1 s'applique, elle crée de nouvelles équations de ces formes. Remarquons que ces équations sont créées seulement par 2 règles

$$\begin{array}{rcl} x = t \wedge x = s & \Longrightarrow & x = t \wedge t = s \\ x = t \wedge e(x) & \Longrightarrow & x = t \wedge e(t) \end{array}$$

Ici t et s sont des termes qui ne sont pas une variable. Par conséquent, pour qu'il puisse exister une application infinie, il faut avoir une création infinie d'équations de la forme $x = t$, avec t un terme. Des équations de cette forme sont créées soit par la règle

$$y = x \wedge y = t \Longrightarrow y = x \wedge x = t$$

de la phase 1 soit par la règle 1 de la phase 2. Mais la règle de la phase 1 doit satisfait la condition $no(y) > no(x)$ et sans l'ajout de nouvelle variable, elle ne peut pas engendrer une création infinie. Une application infinie doit faire intervenir des nouvelles variables. Seule la règle 1 de phase 2 crée des nouvelles variables quantifiées existentiellement. Il faut également que pour chaque nouvelle variable u, l'application des règles crée une équation $u = t$. Mais puisque u est quantifiée, on a $no(u) > no(x)$ pour toute autre variable x, donc la règle permettant de créer $u = t$ ne peut s'appliquer.

Preuve de la correction Montrons que les transformations sont correctes, c-à-d. dans chaque transformation $p \Longrightarrow q$, nous avons $\mathcal{S} \models p \equiv q$. Considérons la première phase. Les transformations concernant les équations de sorte *arbre* sont :

$$\begin{array}{rcl} \mathit{false} \wedge a & \Longrightarrow & \mathit{false} \\ x = x & \Longrightarrow & \mathit{true} \\ x = y & \Longrightarrow & y = x \\ x = f(u_1, .., u_n) \wedge x = g(v_1, .., v_m) & \Longrightarrow & \mathit{false} \\ x = f(u_1, .., u_n) \wedge x = f(v_1, .., v_n) & \Longrightarrow & x = f(u_1, .., u_n) \wedge \\ & & u_1 = v_1 \wedge \cdots \wedge u_n = v_n \\ x = y \wedge x = t & \Longrightarrow & x = y \wedge y = t \end{array}$$

Les 3 premières et la dernière règle sont évidement correctes. La quatrième règle est correcte grâce à la propriété $(a1)$ des arbres et la cinquième est correcte grâce à la propriété $(a2)$ des arbres. En ce qui concerne les équations de sortes *queue*, le premier traitement est dès qu'il existe une variable x de sorte *queue* dans c telle que $x \in Acc_c(x)$, la conjonction est évaluée à *false*. En effet, d'après la définition de $Acc_c(x)$, en faisant des remplacement successifs, nous arrivons

à avoir une équation de la forme $x = t(x\overline{u})$, où $t(x\overline{u})$ est un terme de sorte *queue* qui contient des occurrences de x. D'après la propriété $(q1)$ des queues, l'équation est équivalente à *false*. Les autres transformations sont :

$$
\begin{array}{lrcl}
1 & \mathit{false} \wedge a & \Longrightarrow & \mathit{false} \\
2 & t = t & \Longrightarrow & \mathit{true} \\
3 & t = x & \Longrightarrow & x = t \\
4 & y = x & \Longrightarrow & x = y \\
5 & x = y \wedge e(x) & \Longrightarrow & x = y \wedge e(y) \\
6 & x = t \wedge x = s & \Longrightarrow & x = t \wedge t = s \\
7 & x = t \wedge e(x) & \Longrightarrow & x = t \wedge e(t) \\
8 & [u]t = [v]s & \Longrightarrow & u = v \wedge t = s \\
9 & t[u] = s[v] & \Longrightarrow & u = v \wedge t = s \\
10 & [\overline{u}]\varepsilon = t[\overline{v}] & \Longrightarrow & \varepsilon[\overline{u}] = t[\overline{v}] \\
11 & [\overline{u}]x = \varepsilon[\overline{v}] & \Longrightarrow & [\overline{u}]x = [\overline{v}]\varepsilon \\
12 & \varepsilon = [\overline{u}]x[\overline{v}] & \Longrightarrow & \mathit{false} \\
13 & [\overline{u}]x[\overline{v}] = \varepsilon & \Longrightarrow & \mathit{false} \\
14 & [\overline{u}]x = x[\overline{v}] & \Longrightarrow & \mathit{false}
\end{array}
$$

Les 7 premières règles sont évidement correctes. Les règles 8 et 9 sont correctes grâce aux propriétés $(q5)$ et $(q6)$ des queues respectivement. Les règles 10 et 11 sont correctes grâce à la propriétés $(q8)$. Les règles 12, 13 et 14 sont correctes car deux queues de longueurs différentes sont différentes (propriétés $(q2)$ et $(q3)$).

Après la première phase, nous obtenons soit *false*, soit une conjonction où les équations de la forme $x = t$ ont les membres gauches distincts. Les équations de sorte *queue* qui ne sont pas de cette forme doivent être de l'une des formes $[\overline{u}]x = y[\overline{v}]$, avec x, y des variables distinctes, et $[\overline{u}]x = x[\overline{v}]$, avec x une variable et $|\overline{u}| = |\overline{v}|$.

Considérons la deuxième phase. Prouvons que chaque transformation est correcte.

1. L'équation $[u_1..u_{n+k}]x = y[v_1..v_n]$, avec x, y des variables distinctes, est équivalente à

$$
\begin{array}{c}
\exists u(y = [u_1..u_{n+k}]u \wedge x = u[v_1..v_n] \wedge \mathit{queue}(u)) \\
\bigvee_{i=0}^{n-1} \left(\begin{array}{l} y = [u_1..u_{k+i}]\varepsilon \wedge x = [v_{n-i+1}..v_n]\varepsilon \\ \wedge v_1 = u_{k+i+1} \wedge \cdots \wedge v_{n-i} = u_{n+k} \end{array} \right)
\end{array}
$$

   En effet, si x, y sont des queues définies dans un des cas de la disjonction, alors x, y satisfont l'équation. Dans le sens inverse, soit x, y une solution de l'équation. Si la longueur de x est supérieure ou égale à n, la longueur de y doit être supérieure ou égale à $n + k$. D'après l'équation, les $n + k$ premiers éléments de y doivent être $u_1, .., u_{n+k}$ et les n derniers éléments de x doivent être $v_1, ..., v_n$, et les autres éléments de y et x doivent être en commun. Ce cas est en fait la première sous-formule de la disjonction. Les cas où la longueur de x vaut i, avec $0 \leq i \leq n - 1$ sont représentés dans les autres sous-formules de la disjonction.

2. Equation $[u_1..u_n]x = x[v_1..v_n]$. L'ajout pour chaque couple i, j la disjonction $(u_i = u_j \wedge v_i = v_j) \vee \neg(u_i = u_j) \vee \neg(v_i = v_j)$ préserve bien entendu l'équivalence. Après la distribution, dans les cas où $u_1..u_n$ est de la forme $(u_1..u_k)^{n/k}$, par la même restriction avec les v_i, $v_1..v_n$ doit être aussi de la forme $(v_1..v_k)^{n/k}$. D'après le lemme 3, les équations $[u_1..u_n]x = x[v_1..v_n]$ et $[u_1..u_k]x = x[v_1..v_k]$ ont le même ensemble de solutions. Le remplacement de l'équation $[u_1..u_n]x = x[v_1..v_n]$ par $[u_1..u_k]x = x[v_1..v_k]$ préserve donc l'équivalence de formules. Du fait que $u_1..u_k$ et $v_1..v_k$ sont primitifs, l'équation peut être dénotée par $\stackrel{p}{=}$.

3. Equation $[\overline{u}]x \stackrel{p}{=} x[\overline{v}]$, avec $|\overline{u}| = |\overline{v}| = n$, est équivalente à

$$\bigvee_{k=1}^{n} ([\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\rho(k, \overline{u})]\varepsilon = [\overline{v}]\varepsilon).$$

   En effet, d'après la propriété 1, l'équation a des solution si et seulement si $\overline{u}$ et $\overline{v}$ sont conjugués, c-à-d. qu'il existe une valeur k ($1 \leq k \leq n$) telle que $\rho(k, \overline{u}) = \overline{v}$. Ces possibilités sont présentées par la disjonction, nous avons donc l'équivalence entre deux formules. De plus, puisque $\overline{u}$ est primitif et $\overline{u}$et $\rho(k, \overline{u})$ sont conjugués, l'équation $[\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})]$ est bien notée par $\stackrel{*}{=}$.

4. Avec $|\overline{u}| > |\overline{v}|$, on a

$$[\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(l, \overline{v})] \equiv$$
$$\textstyle\bigvee_{i \in I}(x = [\overline{u}^i u_1..u_k]\varepsilon \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(l, \overline{v})])$$

   Ici $I = \{0\}$ si $k \geq |\overline{v}|$ et $I = \{0, 1\}$ sinon. En effet, d'après la propriété 2, toute solution x des deux équations à gauche doit avoir la longueur inférieure à $|\overline{u}| + |\overline{v}|$, et puisque $|\overline{u}| > |\overline{v}|$, la longueur de x doit être inférieure à $2|\overline{u}|$. Les solutions possibles sont donc $[u_1..u_k]\varepsilon$ si $k \geq |\overline{v}|$ et $[u_1..u_k]\varepsilon$ et $[\overline{u}u_1..u_k]\varepsilon$ sinon. Ces deux solutions sont représentées par la disjonction à droite.

5. Deux équations $\stackrel{*}{=}$ sur une même variable x, avec $|\overline{u}| = |\overline{v}|$ et $k \neq l$:

$$[\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(l, \overline{v})] \equiv \mathit{false}.$$

   L'équivalence est obtenue par la propriété 3.

6. Deux équations $\stackrel{*}{=}$ sur une même variable x, avec $|\overline{u}| = |\overline{v}|$:

$$[\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})] \wedge [\overline{v}]x \stackrel{*}{=} x[\rho(k, \overline{v})] \equiv$$
$$\left(\begin{array}{l} (\textstyle\bigwedge_{i=1}^{k} u_i = v_i \wedge x = [u_1...u_k]\varepsilon) \vee \\ (\textstyle\bigwedge_{i=1}^{|\overline{u}|} u_i = v_i \wedge [\overline{u}]x \stackrel{*}{=} x[\rho(k, \overline{u})]) \end{array}\right)$$

   L'équivalence est obtenue par la propriété 3.

A la fin des deux phases, nous obtenons soit *false* soit une formule de la forme $\bigwedge_i \exists \overline{x_i}(c_i \wedge \bigwedge_j \neg e_{ij})$. Ici chaque conjonction c_i doit être sous forme résolue. En effet, les conditions de forme résolues sont satisfaites, à savoir :

1. les équations sont soit de la forme $x = t$ soit de la forme $[\overline{u}]x = x[\rho(k, \overline{u}]$, avec $k \leq |\overline{u}|$, par phase 1 et 2 ;
2. les représentants des équations sont tous distincts, par phase 1 et 2 ;
3. pour chaque équation de la forme $x = y$, $no(x) > no(y)$, par phase 1 ;
4. c_i est complètement typée, car la conjonction initiale l'est, et seule la première transformation de la phase 2 introduit une nouvelle variable avec la contrainte *queue*, et la phase 1 assure que les contraintes de sorte respectent le type des symboles de fonction ;
5. la phase 1 assure que pour toute variable x de sorte *queue* ayant une occurrence dans c_i, $x \notin Acc_{c_i}(x)$.

Les négations $\neg e_{ij}$ sont ajoutées par la transformation 2 de la phase 2. Ce sont donc des négations d'équations entre deux variables. Les variables dans $\overline{x_i}$ sont également ajoutées par cette transformation. Puisqu'initialement toutes les variables existantes sont libres, chaque variable de $\overline{x_i}$ est accessible dans c_i depuis une variable libre. CQFD.

4 Algorithme de décision

4.1 Algorithme

Nous présentons un algorithme pour transformer une formule F en une disjonction de formules basiques. Tout d'abord, pour chaque variable libre x nous ajoutons à F la formule $queue(x) \vee arbre(x)$ et chaque sous-formule $\exists x F'$ est changée en $\exists x(F' \wedge (queue(x) \vee arbre(x))$. Les contraintes sont mises sous forme de contraintes élémentaires, des nouvelles variables sont introduites si nécessaires et sont quantifiées existentiellement, avec une contrainte de sorte respective. La formule obtenue est ensuite mise sous forme prénexe, c'est-à-dire tous les quantificateurs sont mis au début de la formule (éventuellement en renommant des variables quantifiées), puis le reste de la formule est mis en forme normale disjonctive $\bigvee(c \wedge \bigwedge \neg c_i)$, où les c et c_i sont des conjonctions de contraintes élémentaires. Il est claire que ces transformations préservent l'équivalence de formules.

Pour chaque sous-formule $c \wedge \bigwedge \neg c_i$, appliquer l'algorithme de transformation en forme résolue sur c et les c_i. Nous obtenons une formule sous forme

$$\bigvee \exists \overline{u}(c' \wedge \bigwedge_{i \in I'} \neg \exists \overline{u'_i}(c'_i \wedge \bigwedge_j \neg e_{ij})) \tag{1}$$

D'après la propriété 1, les variables de $\overline{u'_i}$ sont accessibles depuis une variable libre dans c'_i. D'après la propriété 5, $\mathcal{S} \models \exists ? \overline{u'_i} c'_i$. D'après la propriété 6, la formule (1) est équivalente à

$$\bigvee \exists \overline{u}(c' \wedge \bigwedge_{i \in I'} \neg(\exists \overline{u'_i} c'_i \wedge \bigwedge_j \neg \exists \overline{u'_i}(c'_i \wedge e_{ij})))$$

et donc à

$$\bigvee \exists \overline{u}(c' \wedge \bigwedge_{i \in I'} (\neg \exists \overline{u'_i} c'_i \vee \bigvee_j \exists \overline{u'_i}(c'_i \wedge e_{ij}))).$$

En faisant des distribution des $\vee$ sur des $\wedge$, la formule est transformée en une formule de la forme

$$\bigvee \exists \overline{u}(c \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i) \tag{2}$$

En mettant les conjonctions c en forme résolue pour celles qui ne le sont pas encore, on obtient une formule toujours de la forme (2) et où les conjonctions c et c_i sont sous forme résolue, où chaque variable de $\overline{u_i}$ est accessible dans c_i depuis une variable libre. En gardant dans les $\overline{u}$ seulement les variables accessibles dans c depuis une variable libre, les autres variables de $\overline{u}$ étant remontées dans la partie des quantificateurs de F, nous avons une formule de la forme $Q(F1 \vee \ldots \vee F_n)$, où Q dénote la partie des quantificateurs et les F_i sont des formules basiques. Nous montrons comment éliminer les quantificateurs et transformer la formule en une disjonction de formules basiques.

Considérons le quantificateur le plus interne de la formule est supposons que ce soit un quantificateur existentiel. C'est-à-dire la formule est $Q\exists x(F_1 \vee \ldots \vee F_n)$, où Q représente le reste des quantificateur. Cette formule est équivalente à $Q(\exists x F_1 \vee \ldots \vee \exists x F_n)$. Le travail devient à transformer chaque formule $\exists x F_i$ en disjonction de formules basique. Il fera l'objet de la sous-section 4.2.

Si le quantificateur le plus interne est un quantificateur universel, la formule est $Q\forall x(F_1 \vee \ldots \vee F_n)$, qui devient $Q\neg\exists x\neg(F_1 \vee \ldots \vee F_n)$ donc $Q\neg\exists x(\neg F_1 \wedge \ldots \wedge \neg F_n)$. Chaque formule F_i est une formule basique donc de la forme $\exists \overline{u}(c \wedge \bigwedge \neg \exists \overline{u_i} c_i)$. D'après la définition de formule basique et la propriété 5, $\mathcal{S} \models \exists ? \overline{u} c$. D'après la propriété 6, F_i est équivalente à $\exists \overline{u} c \wedge \bigwedge \neg \exists \overline{u} \exists \overline{u_i}(c \wedge c_i)$. Nous transformons donc $(\neg F_1 \wedge \ldots \wedge \neg F_n)$ en forme normale disjonctive et éliminons des quantificateurs existentielles introduits comme précédent. Après l'élimination du quantificateur $\exists x$, la négation du résultat est ensuite mise en forme de disjonction de formules basiques comme précédent.

4.2 Elimination de quantificateur

Soit x une variable et b une formule basique. Nous présentons l'élimination de quantificateur de $\exists x b$ en deux parties, dépendant de la forme de b : (1) lorsque b ne contient pas de négation, (2) b contient des négations. L'élimination du quantificateur $\exists x$ transforme $\exists x b$ en une disjonction de formules basiques.

4.2.1 Elimination dans une conjonction résolue

Soient x une variable et $\exists \overline{u} c$ une formule basique. Transformons la formule $\exists x \exists \overline{u} c$ en une formule basique de la forme $\exists \overline{u'} c'$.

S'il existe une variable libre y telle que $x \in Acc_c(y)$, alors la tâche est faite. Si x n'est pas accessible depuis une variable libre, la formule $\exists x \exists \overline{u} c$ peut s'écrire en $\exists \overline{u'}(c' \wedge \exists x \overline{v} c_x)$, où $\overline{v}$ est le vecteur des variables de $\overline{u}$ qui sont accessibles

depuis x et non accessibles depuis aucune autre variable libre, c_x est la conjonction des contraintes élémentaires qui font intervenir des variables de $x\overline{v}$ et c' la conjonction des autres contraintes de c. La conjonction c' ne contient donc pas d'occurrences des variables de $x\overline{v}$. Montrons que

$$\mathcal{S} \models \exists x \exists \overline{u} c \equiv \exists \overline{u'} c'.$$

Ceci est trivial si c_x n'a pas d'équations, car dans ce cas $\overline{v}$ est vide. Si c_x a une équation, du fait que c est résolue, et que x n'est pas accessible depuis une autre variable libre, x doit être le représentant d'une équation de c_x. Les variables de v sont séparées en deux parties $\overline{v_1}$ les représentants des autres équations de c_x et $\overline{v_2}$ les autres variables de $\overline{v}$. D'après la propriété 4, $\mathcal{S} \models \exists x \overline{v_1} c_x$, on a donc

$$\mathcal{S} \models \exists \overline{u'}(c' \wedge \exists x \overline{v} c_x) \equiv \exists \overline{u'} c',$$

d'où l'équivalence à démontrer.

Exemple 2 *On élimine le quantificateur $\exists x$ dans la formule $\exists x \exists u v_1 v_2([v_1 v_2]x = x[v_1 v_2] \wedge v_1 = f(u) \wedge y = f(u) \wedge queue(x) \wedge arbre(v_1) \wedge queue(v_2) \wedge arbre(y) \wedge queue(u))$. Dans cette formule x n'est pas accessible depuis une autre variable libre, u est accessible depuis x, y et v_1, v_2 sont accessibles depuis x. La formule est équivalente à*

$$\exists u \left(\begin{array}{l} y = f(u) \wedge arbre(y) \wedge queue(u) \wedge \\ \exists x v_1 v_2 \left(\begin{array}{l} [v_1 v_2]x = x[v_1 v_2] \wedge v_1 = f(u) \wedge \\ queue(x) \wedge arbre(v_1) \wedge queue(v_2) \end{array} \right) \end{array} \right)$$

et donc équivalente à

$$\exists u(y = f(u) \wedge arbre(y) \wedge queue(u)).$$

4.2.2 Elimination dans une formule basique

Soit x une variable et soit b une formule basique. Transformons $\exists x b$ en une disjonction de formules basiques. Si x n'a pas d'occurrence dans b le résultat sera b. Supposons que x ait des occurrences dans b. Soit b la formule basique $\exists \overline{u}(c \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i)$. S'il existe une autre variable libre y de b telle que $x \in Acc_c(y)$, alors $\exists x b$ est une formule basique. Dans le cas contraire, où x n'est accessible dans c depuis aucune autre variable libre, tous les cas possibles pour x sont listés suivant. Note : puisque les contraintes de type *queue* et *arbre* respectent le type des symboles de fonction, pour rendre les formules plus visibles dans les exemples, nous omettons ces contraintes dans les cas où cela ne change pas la sémantique des formules.

(a) x est de sorte *arbre* et x a des occurrences dans des équations de c Du fait que x a des occurrences dans c mais x n'est accessible dans c depuis aucune autre variable libre, x doit être le membre gauche d'une équation $x = t$. L'ensemble $Acc_c(x)$ est réparti en deux sous-ensemles disjoints $Acc_c^a(x)$

des variables de sorte *arbre* et $Acc^q_c(x)$ des variables de sorte *queue*. Soit $A^q_c(x)$ l'ensemble de tous les variables accessibles dans c depuis une variable de $Acc^q_c(x)$. Notons que $x \notin A^q_c(x)$, car sinon il doit exister une variable w de sorte *queue* dans $Acc^q_c(x)$ telle que $w \in Acc_c(w)$, ce qui contrarie le fait que c est résolue. Nous pouvons donc calculer un vecteur $\overline{v}$ des variables de sorte *arbre* de $\overline{u}$ qui sont dans $Acc^a_c(x) \backslash A^q_c(x)$ et qui sont des membres gauches d'équations de c. Soit c_x la conjonction de ces équations et des contraintes *arbre* concernant ces variables et soit c' le reste des contraintes de c. Soit $\overline{u'}$ le vecteur des variables de $\overline{u}$ qui ne sont pas dans $\overline{v}$. Notons que ni x ni aucune variable de $\overline{v}$ n'a d'occurrence dans c', car dans le cas contraire, puisque c est résolue, l'occurrence dans c' ne peut pas être un représentant, la variable concernée sera donc dans $A^q_c(x)$, ce qui est contraire à la définition de $\overline{v}$. La formule $\exists x b$ peut donc s'écrire en

$$\exists \overline{u'}(c' \wedge \exists x \overline{v} c_x \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i).$$

D'après la propriété $(a3)$ des arbres, $\mathcal{S} \models \exists! x \overline{v} c_x$ puis d'après la propriété 6, la formule $\exists x b$ est équivalente à

$$\exists \overline{u'}(c' \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} \exists x \overline{v}(c_x \wedge c_i)).$$

L'élimination de quantificateurs présentée en 4.2.1 est appliquée dans les négations. Le quantificateur $\exists x$ est donc éliminé. Pour rendre le résultat en forme basique il faudra éventuellement éliminer d'autres quantificateurs de $\exists \overline{u'}$.

Exemple 3 *Eliminer* $\exists x_1$ *:*

$$\exists x_1 \exists y w \begin{pmatrix} x_1 = f(y,v) \wedge y = g(z) \wedge z = g(y) \wedge w = g(x_1) \\ \wedge [z]v = v[z] \wedge queue(u) \wedge queue(v) \wedge arbre(x_1) \\ \wedge arbre(x_2) \wedge arbre(y) \wedge arbre(z) \wedge arbre(w) \\ \wedge \neg(u = [z]v)) \wedge \neg(x_2 = g(x_1)) \end{pmatrix}$$

Ici $Acc^a_c(x_1) = \{y, z, w\}$*,* $Acc^q_c(x_1) = \{v\}$ *et* $A^q_c(x_1) = \{y, z\}$*. La formule est transformée en*

$$\exists y \begin{pmatrix} y = g(z) \wedge z = g(y) \wedge [z]v = v[z] \wedge queue(u) \\ \wedge queue(v) \wedge arbre(x_2) \wedge arbre(y) \wedge arbre(z) \\ \wedge \neg \exists x_1 w \begin{pmatrix} x_1 = f(y,v) \wedge w = g(x_1) \wedge u = [z]v) \\ \wedge arbre(x_1) \wedge arbre(w) \end{pmatrix} \\ \wedge \neg \exists x_1 w \begin{pmatrix} x_1 = f(y,v) \wedge w = g(x_1) \wedge x_2 = g(x_1) \\ \wedge arbre(x_1) \wedge arbre(w) \end{pmatrix} \end{pmatrix}$$

puis en

$$\exists y \begin{pmatrix} y = g(z) \wedge z = g(y) \wedge [z]v = v[z] \wedge queue(u) \\ \wedge queue(v) \wedge arbre(x_2) \wedge arbre(y) \wedge arbre(z) \\ \wedge \neg(u = [z]v) \\ \wedge \neg \exists x_1 w \begin{pmatrix} x_1 = f(y,v) \wedge w = g(x_1) \wedge x_2 = g(x_1) \\ \wedge arbre(x_1) \wedge arbre(w) \end{pmatrix} \end{pmatrix}$$

Cette formule est une formule basique.

(b) x est de sorte *arbre* et x n'a pas d'occurrence dans les équations de c Nous prouvons le lemme suivant :

Lemme 4 *Soient x une variable de sorte arbre et $\bigwedge_{i\in I} \neg\exists\overline{u_i}c_i$ une formule basique avec I éventuellement vide. Si chaque conjonction c_i contient au moins une occurrence de x, alors $\mathcal{S} \models \exists x(\bigwedge_{i\in I} \neg\exists\overline{u_i}c_i)$.*

Preuve : La variable x apparaît dans des c_i
- soit sous forme $x = f(u_1, .., u_n)$,
- soit sous forme $x = y$, en ce cas y n'est pas dans $\overline{u_i}$, car du fait que c_i est résolue, $no(x) > no(y)$ et toute variable u de $\overline{u_i}$ a $no(u) > no(x)$,
- soit à droite d'une équation, en ce cas x doit être accessible depuis une autre variable libre y, car du fait que la formule donné est basique, toute variable u de $\overline{u_i}$ est accessible depuis une variable libre.

Dans le premier cas, la racine de l'arbre est fixé à f et dans les deux derniers cas, x dépend de y. Du fait que l'ensemble de symbole de fonction est infini, il existe donc une affectation qui associe à x un arbre dont la racine est étiqueté par un symbole différent de tous les symboles de fonction dans les c_i, et différent du symbole étiquetant la racine de y, quelque soit ce dernier. Les conjonctions c_i sont donc évaluées à *false* dans cette affectation. On conclut donc $\mathcal{S} \models \exists x(arbre(x) \wedge \bigwedge_{i\in I} \neg\exists\overline{u_i}c_i)$. CQFD.

L'ensemble I est séparé en deux sous-ensembles disjoints I_1 et I_2, où I_1 est l'ensemble des indices i tels que c_i contient au moins une occurrence de x. Soit c' la conjonction des contraintes de c à l'exception de $arbre(x)$. D'après le lemme 4, la formule $\exists xb$ est équivalente à

$$\exists\overline{u}(c' \wedge \bigwedge_{i\in I_2} \neg\exists\overline{u_i}c_i).$$

Cette formule est une formule basique.

Exemple 4 *Eliminant $\exists x_2$ dans la dernière formule de l'exemple 3, on obtient la formule basique :*

$$\exists y \begin{pmatrix} y{=}g(z) \wedge z{=}g(y) \wedge [z]v{=}v[z] \wedge queue(u) \\ \wedge queue(v) \wedge arbre(y) \wedge arbre(z) \wedge \neg(u = [z]v) \end{pmatrix}$$

(c) x est de sorte *queue* et une équation de la forme $x = t$ est dans c Du fait que x n'est pas accessible depuis une autre variable libre, x n'a pas d'autres occurrences dans les autres équations de c. Soit c' la conjonction des contraintes de c à l'exception de $x = t$ et de $queue(x)$. Du fait que $\mathcal{S} \models \exists!x(x = t \wedge queue(x))$, la formule $\exists xb$ est équivalente à

$$\exists\overline{u}(c' \wedge \bigwedge \neg\exists x\overline{u_i}(x = t \wedge queue(x) \wedge c_i)).$$

L'élimination de quantificateurs présentée en 4.2.1 est appliquée dans les négations. Le quantificateur $\exists x$ est donc éliminé. Pour rendre le résultat en forme basique il faut éventuellement éliminer d'autres quantificateurs de $\exists\overline{u}$.

Exemple 5 *Transformation pour éliminer* $\exists x$ *:*

$$\begin{aligned}
&\exists x \exists y v (x = [v]y \wedge \neg(x = \varepsilon) \wedge \neg \exists u z (x = [u]z)) \\
\equiv &\exists y v (\neg \exists x (x = [v]y \wedge x = \varepsilon) \wedge \neg \exists x u z (x = [v]y \wedge x = [u]z)) \\
\equiv &\exists y v (\neg \mathit{false} \wedge \neg \exists x u z (x = [u]z \wedge u = v \wedge z = y)) \\
\equiv &\exists y v (\neg \mathit{true}) \\
\equiv &\mathit{false}
\end{aligned}$$

(d) x **est de sorte *queue* et une équation de la forme** $[\overline{u}]x = x[\rho(k, \overline{u})]$ **est dans** c Nous pouvons supposer ici et dans le cas (e) que si x a une occurrence dans une conjonction c_i alors x a une occurrence dans une équation de c_i. En effet, si x n'est dans aucune équation de c_i et seule la contrainte $\mathit{queue}(x)$ est dans c_i, nous pouvons la supprimer grâce à l'équivalence $\mathit{queue}(x) \wedge \neg \exists \overline{u_i}(c_i' \wedge \mathit{queue}(x)) \equiv \mathit{queue}(x) \wedge \neg \exists \overline{u_i} c_i'$. Nous prouvons le lemme suivant :

Lemme 5 *Soient* x *une variable de sorte queue et* $\bigwedge_{i \in I} \neg \exists \overline{u_i} c_i$ *une formule basique avec* I *éventuellement vide. Si dans chaque conjonction* c_i *un des cas suivants se produit :*

- *il existe une variable* $y \notin \overline{u_i}$ *telle que* $x \in Acc_{c_i}(y)$*,*
- c_i *contient une équation de l'une des formes* $x = \varepsilon$*,* $x = y$ *et* $x = [\overline{v}]y[\overline{v'}]$ *avec* $y \notin \overline{u_i}$*,*

alors

$$\mathcal{S} \models \exists x([\overline{u}]x = x[\rho(k, \overline{u})] \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i).$$

Preuve : Respectant les contraintes de sorte, on affecte un élément quelconque du domaine de $\mathcal{S}$ à chaque variable libre de la formule basique. Dans le premier cas, pour la valeur associée à y, la valeur de x sera déterminée d'une façon unique dans c_i. Dans le second cas, si c_i contient une équation $x = \varepsilon$, la valeur de x est également unique dans c_i. Si c_i contient une équation $x = y$ ou $x = [\overline{v}]y[\overline{v'}]$ avec $y \notin \overline{u_i}$, pour la valeur associée à y, la valeur de x est aussi déterminée d'une façon unique dans c_i. Du fait que l'équation $[\overline{u}]x = x[\rho(k, \overline{u})]$ a un ensemble infini de solutions d'après la propriété 1, il existe donc une queue pour x qui rend tous les conjonctions c_i fausses. Il existe donc dans $\mathcal{S}$ une queue pour x telle que $[\overline{u}]x = x[\rho(k, \overline{u})] \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i$ soit vraie. On conclut donc

$$\mathcal{S} \models \exists x([\overline{u}]x = x[\rho(k, \overline{u})] \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i).$$

CQFD.

Du fait que x n'est pas accessible depuis une autre variable libre, x n'a pas d'autres occurrences dans les autres équations de c. Les cas possibles pour x sont :

d1. Dans chaque conjonction c_i qui contient une occurrence de x, il existe une variable libre y_i telle que $x \in Acc_{c_i}(y_i)$, ou c_i contient une équation

de l'une des formes $x = \varepsilon$, $x = y$ et $x = [\overline{v}]y[\overline{v'}]$ avec y libre. D'après le lemme 5, la formule $\exists xb$ est équivalente à

$$\exists \overline{u}(c' \wedge \bigwedge_{i \in I'} \neg \exists \overline{u_i} c_i)$$

où c' est la conjonction des contraintes de c à l'exception de $[\overline{u}]x = x[\rho(k, \overline{u})]$ et de $queue(x)$, I' est le sous-ensemble maximal de I tel que pour tout $i \in I'$, c_i n'a pas d'occurrence de x. Cette formule est une formule basique.

Exemple 6 *Eliminant $\exists v$ dans la formule résultante de l'exemple 4 on obtient la formule basique :*

$$\exists y \begin{pmatrix} y = g(z) \wedge z = g(y) \wedge queue(u) \\ \wedge arbre(y) \wedge arbre(z) \end{pmatrix}$$

d2. Il existe une conjonction c_i dans laquelle x n'est pas accessible depuis une autre variable libre et qui contient une équation de la forme $[\overline{v}]x = x[\rho(l, \overline{v})]$. En utilisant la procédure de résolution de contraintes d'arbres [5, 7], nous montrons comment supprimer l'équation $[\overline{v}]x = x[\rho(l, \overline{v})]$. La formule $\exists xb$ est équivalente à

$$\exists x \overline{u}(c \wedge \neg \exists \overline{u_i}(c \wedge c_i) \wedge \bigwedge_{j \in I, j \neq i} \neg \exists \overline{u_j} c_j)$$

La conjonction $c \wedge c_i$ est mise sous forme résolue. Dans cette procédure, si $|\overline{u}| \neq |\overline{v}|$ ou $k \neq l$, les deux équations $[\overline{u}]x = x[\rho(k, \overline{u})]$ et $[\overline{v}]x = x[\rho(l, \overline{v})]$ sont supprimées et remplacées par des équations de la forme $x = t$, ce cas fait l'objet de l'item d3. suivant. Sinon, dans la règle 6 de la deuxième phase, l'équation $[\overline{u}]x = x[\rho(k, \overline{u})]$ est gardée. Les équations recopiées de c seront restaurées puis supprimées, d'après la procédure dans [5, 7]. L'équation $[\overline{v}]x = x[\rho(l, \overline{v})]$ est ainsi supprimée.

Exemple 7 *Eliminer $\exists x$:*

$$\begin{array}{rl} & \exists x([u]x = x[u] \wedge \neg \exists y(x = [u]y) \wedge \neg([v]x = x[v])) \\ \equiv & \exists x \left(\begin{array}{l} [u]x = x[u] \wedge \neg \exists y(x = [u]y) \wedge \\ \neg([u]x = x[u] \wedge [v]x = x[v]) \end{array} \right) \\ \equiv & \exists x \left(\begin{array}{l} [u]x = x[u] \wedge \neg \exists y(x = [u]y) \wedge \\ \neg(x = \varepsilon) \wedge \neg([u]x = x[u] \wedge u = v) \end{array} \right) \\ \equiv & \exists x \left(\begin{array}{l} [u]x = x[u] \wedge \neg \exists y(x = [u]y) \wedge \\ \neg(x = \varepsilon) \wedge \neg(u = v) \end{array} \right) \end{array}$$

L'équation $[v]x = x[v]$ est donc supprimée. La suite se trouve dans l'exemple 8.

d3. Il existe une conjonction c_i dans laquelle x n'est pas accessible depuis une autre variable libre et qui contient une équation de la forme $x = [\overline{v}]y[\overline{v'}]$,

avec $y \in \overline{u_i}$. Ici $\overline{v}\overline{v'}$ doit être non vide car sinon l'équation devient $x = y$, ce qui entraîne que $no(x) > no(y)$ contradictoire avec le fait que $y \in \overline{u_i}$. Si $\overline{v}$ n'est pas vide, soit a le plus petit nombre tel que $|\overline{u}|a \geq |\overline{v}|$. L'équation $[\overline{u}]x = x[\rho(k, \overline{u})]$, avec $\overline{u} = u_1...u_k...u_n$ est remplacée par la disjonction

$$\bigvee_{j=0}^{a-1} (x = [\overline{u}^j u_1..u_k]\varepsilon) \vee \exists z(x = [\overline{u}^a]z \wedge [\overline{u}]z = z[\rho(k, \overline{u})])$$

Sinon soit a le plus petit nombre tel que $|\overline{u}|a \geq |\overline{v'}|$. L'équation $[\overline{u}]x = x[\rho(k, \overline{u})]$ est remplacée par la disjonction

$$\bigvee_{j=0}^{a-1} (x = [\overline{u}^j u_1..u_k]\varepsilon) \vee \exists z(x = z[\overline{u}^a] \wedge [\overline{u}]z = z[\rho(k, \overline{u})])$$

La distribution des $\vee$ sur des $\wedge$ est effectuée et on se retrouve dans le cas (c). La variable z reste à être éliminée mais le fait d'associer à x une liste de longueur supérieure à $|\overline{v}|$ (ou $|\overline{v'}|$) permet de ne pas créer une équation de la forme $z = t$.

Exemple 8 *Suite de l'exemple 7 :*

$$\begin{array}{l} \exists x([u]x = x[u] \wedge \neg\exists y(x = [u]y) \wedge \neg(x = \varepsilon) \wedge \neg(u = v)) \\ \equiv \left(\begin{array}{l} \exists x(x = \varepsilon \wedge \neg\exists y(x = [u]y) \wedge \neg(x = \varepsilon) \wedge \neg(u = v)) \\ \vee \\ \exists xz \left(\begin{array}{l} x = [u]z \wedge [u]z = z[u] \wedge \neg\exists y(x = [u]y) \wedge \\ \neg(x = \varepsilon) \wedge \neg(u = v) \end{array} \right) \end{array} \right) \end{array}$$

Elimination de $\exists x$ dans la première formule de la disjonction :

$$\begin{array}{ll} & \exists x(x = \varepsilon \wedge \neg\exists y(x = [u]y) \wedge \neg(x = \varepsilon) \wedge \neg(u = v)) \\ \equiv & \left(\begin{array}{l} \neg\exists xy(x = \varepsilon \wedge x = [u]y) \wedge \neg\exists x(x = \varepsilon \wedge x = \varepsilon) \\ \wedge\neg\exists x(x = \varepsilon \wedge u = v) \end{array} \right) \\ \equiv & \neg false \wedge \neg true \wedge \neg(u = v) \\ \equiv & false \end{array}$$

Elimination de $\exists x$ dans la seconde formule :

$$\begin{array}{ll} & \exists xz \left(\begin{array}{l} x = [u]z \wedge [u]z = z[u] \wedge \neg\exists y(x = [u]y) \wedge \\ \neg(x = \varepsilon) \wedge \neg(u = v) \end{array} \right) \\ \equiv & \exists z \left(\begin{array}{l} [u]z = z[u] \wedge \neg\exists xy(x = [u]z \wedge x = [u]y) \wedge \\ \neg\exists x(x = [u]z \wedge x = \varepsilon) \wedge \\ \neg\exists x(x = [u]z \wedge u = v) \end{array} \right) \\ \equiv & \exists z \left(\begin{array}{l} [u]z = z[u] \wedge \neg\exists xy(x = [u]z \wedge y = z) \wedge \\ \neg false \wedge \neg(u = v) \end{array} \right) \\ \equiv & \exists z([u]z = z[u] \wedge \neg true \wedge \neg(u = v)) \\ \equiv & false \end{array}$$

Le résultat est donc la formule false.

(e) x est de sorte *queue* et x n'a pas d'occurrence dans les équations de c Nous avons le lemme suivant

Lemme 6 *Soit x une variable de sorte queue et $\bigwedge_{i\in I} \neg\exists\overline{u_i}c_i$ une formule basique avec I éventuellement vide. Si dans chaque conjonction c_i une des cas suivants se produit :*

- *il existe une variable $y \notin \overline{u_i}$ telle que $x \in Acc_{c_i}(y)$,*
- *c_i contient une équation de l'une des formes $x{=}\varepsilon$, $x{=}y$, $[\overline{v}]x{=}x[\rho(k,\overline{v'})]$ et $x{=}t(y)$, avec $y \notin \overline{u_i}$,*

alors

$$\mathcal{S} \models \exists x(queue(x) \wedge \bigwedge_{i\in I} \neg\exists\overline{u_i}c_i).$$

Preuve : Respectant les contraintes de sorte, on affecte un élément quelconque du domaine de $\mathcal{S}$ à chaque variable libre de la formule basique. Dans le premier cas, pour la valeur associée à y, la valeur de x sera déterminée d'une façon unique dans c_i. Dans le second cas, si c_i contient une équation $x = \varepsilon$, la valeur de x est également unique dans c_i. Si c_i contient une équation $x = y$ avec $y \notin \overline{u_i}$, pour la valeur associée à y, la valeur de x est aussi déterminée d'une façon unique dans c_i. Si c_i contient $x = t(y)$ avec $y \notin \overline{u_i}$, c_i fixe une partie de la valeur associée à x. Si c_i contient une équation $[\overline{v}]x = x[\rho(k,\overline{v'})]$, d'après la propriété 1, la forme des queues possibles pour x sera $\overline{v}^* v_1..v_k$. Du fait qu'il y a un nombre infini de queues et de formes de queue pour répondre à $queue(x)$, il existe donc une queue pour x qui rend tous les conjonctions c_i fausses. On conclut donc

$$\mathcal{S} \models \exists x(queue(x) \wedge \bigwedge_{i\in I} \neg\exists\overline{u_i}c_i).$$

CQFD.

Les cas possibles pour x sont :

e1. Pour chaque conjonction c_i qui contient une occurrence de x, soit x est accessible dans c_i depuis une autre variable libre, soit c_i contient une équation de l'une des formes $x = y$, $x = \varepsilon$, $[\overline{v}]x = x[\rho(k,\overline{v'})]$ et $x = [\overline{v}]y[\overline{v'}]$, avec y une variable libre. D'après le lemme 6, la formule $\exists x b$ est équivalente à

$$\exists\overline{u}(c' \wedge \bigwedge_{i\in I'} \neg\exists\overline{u_i}c_i)$$

où c' est la conjonction des contraintes de c à l'exception de $queue(x)$, I' est le sous-emsemble maximal de I tel que pour tout $i \in I'$, c_i n'a pas d'occurrence de x. Cette formule est une formule basique.

Exemple 9 *Eliminer $\exists x$:*

$$\begin{array}{rl} & \exists x\exists u \begin{pmatrix} y{=}f(u) \wedge queue(x) \wedge \\ \neg\exists v(u{=}[v]z) \wedge \exists v([v]x{=}x[v]) \end{pmatrix} \\ \equiv & \exists u(y{=}f(u) \wedge \neg\exists v(u{=}[v]z)) \end{array}$$

e2. Il existe une conjonction c_i dans laquelle x n'est pas accessible depuis une autre variable libre et qui contient une équation $x = [\overline{u}]y[\overline{u'}]$ où les

variables de $y\overline{u}\overline{u'}$ sont tous quantifiées. Soit $n = |\overline{u}|+|\overline{u'}|$. Puisque $queue(x)$ est dans c, en gardant toujours l'équivalence nous pouvons ajouter dans c la disjonction

$$\begin{array}{l}(x = \varepsilon)\vee\\ \bigvee_{1\leq i<n} \exists v_1..v_i(x = [v_1..v_i]\varepsilon)\vee\\ \exists y v_1...v_n(x = [v_1..v_{|\overline{u}|}]y[v_{|\overline{u}|+1}..v_n] \wedge queue(y))\end{array}$$

La distribution des $\vee$ sur des $\wedge$ est effectuée et on se retrouve dans le cas (c). Les variables $y\overline{u}\overline{u'}$ restent à être éliminées mais le fait d'associer à x une liste soit de longueur déterminée soit de longueur supérieure à $|\overline{u}|+|\overline{u'}|$ permet de supprimer des négations de ce cas.

Exemple 10 *Transformations pour éliminer* $\exists x$ *:*

$$\begin{array}{l}\exists x(queue(x) \wedge \neg(x{=}\varepsilon) \wedge \neg\exists uz(x{=}[u]z))\\ \equiv\begin{pmatrix}\exists x(x{=}\varepsilon \wedge \neg(x{=}\varepsilon) \wedge \neg\exists uz(x{=}[u]z))\vee\\ \exists xyv(x{=}[v]y \wedge \neg(x{=}\varepsilon) \wedge \neg\exists uz(x{=}[u]z))\end{pmatrix}\end{array}$$

D'après l'exemple 5, l'élimination de $\exists x$ *dans les deux formules de la disjonction retourne* false*. Le résultat est donc la formule* false*.*

4.3 Décidabilité

Dans la procédure d'élimination de quantificateur nous n'introduisons pas de nouvelle variable libre. Les cas d2, d3 et e2, où des nouvelles variables quantifiées sont ajoutées se ramènent toujours à un autre cas, où des variables quantifiées sont éliminées sans ajouter de nouvelles. La procédure va s'arrêter donc après un temps fini.

Théorème 2 *Soit* x *une variable et* b *une formule basique. La procédure d'élimination de quantificateur transforme* $\exists xb$ *en une disjonction de formules basiques.*

Si l'algorithme s'applique sur une formule close, nous obtiendrons soit la valeur de vérité *true* (le résultat est une disjonction contenant une seule formule basique *true*) soit la valeur de vérité *false* (le résultat est une disjonction vide, donc *false*). On conclut donc :

Corollaire 1 *La théorie du premier ordre de l'algèbre des arbres finis ou infinis avec queues est complète et décidable.*

5 Conclusion

Dans ce papier nous avons présenté l'algèbre des arbres finis ou infinis étendue avec des queues. L'ensemble des symboles de fonction de la signature est infini. Nous avons présenté un algorithme d'élimination de quantificateurs dans

cette algèbre. Cette algorithme permet de décider la valeur de vérité des formules du premier ordre closes dans cette algèbre. Il montre également que la théorie du premier ordre de l'algèbre est complète et décidable.

Un simple exemple d'application est dans l'analyse syntaxique. Pour reconnaître les mots $a^n b^n$ avec des listes en Prolog on peut utiliser la différence de listes. Avec les queues les contraintes sont plus naturelles et simples à écrire. D'un autre côté, les arbres avec les queues permettent aussi à modéliser les documents XML. Le traitement de contraintes dans ce domaine est fait dans le cadre de la programmation logique en [2]. Une des applications de notre algorithme sera de l'utiliser pour résoudre des contraintes du premier ordre dans ce domaine.

Remerciements Nous remercions Alain Colmerauer pour des nombreuses discussions sur la théorie des arbres avec queues et pour ses remarques et ses conseils sur notre travail.

Références

[1] K.L. Clark, Negation as failure, in Logic and Databases, Ed. H. Gallaire and J. Minker, Plenum Press, 293-322, 1978.

[2] J. Coelho, M. Florido, XCentric : logic programming for XML processing, WIDM2007, 1-8, 2007.

[3] A. Colmerauer, An introduction to Prolog III, Commun. ACM 33(7), 69-90, 1990.

[4] H. Comon, Résolution de contraintes dans des algèbres de termes. Rapport d'habilitation, Université de Paris Sud, 1991.

[5] T-B-H. Dao, Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis, Thèse d'Informatique, Université Aix-Marseille II, 2000.

[6] K. Djelloul, T-B-H. Dao, Extentions into trees of first-order theories, AISC06, 53-67, 2006.

[7] K. Djelloul, T-T-H. Dao, T. Frühwirth, Theory of finite or infinite trees revisisted, TPLP, 2008.

[8] J. Jaffar, M. Maher, K. Marriott, P. Stuckey, The semantics of constraint logic programs, The Journal of Logic Programming 37(1998), 1-46.

[9] K. Kunen, Negation in logic programming, Journal of Logic Programming, 4, 289-308, 1987.

[10] M. Lothaire, Combinatorics on words, Encyclopedia of Mathematics, Vol 17, Addison-Wesley, 1983. Reprinted in 1997 by Cambridge University Press, in Cambridge Mathematical Library.

[11] M. J. Maher, Complete axiomatization of the algebra of finite, rational and infinite trees, Technical report, IBM-T.J. Watson Research Center, 1988.

[12] A. Malcev, Axiomatizable classes of locally free algebras of various types, In The Metamathematics of Algebraic Systems : Collected Paper, chapter 23, 262-281, 1971.

[13] G. Nelson, D.C. Oppen, Simplification by cooperating decision procedures, ACM Transactions on Programming Languages and Systems 1(2), 245-257, 1979.

[14] T. Rybina, A. Voronkov, A Decision Procedure for Term Algebras with Queues, ACM Transactions on Computational Logic, Vol. 2, No. 2, April 2001, pages 155-181.

[15] T. Zhang, H. Sipma, Z. Manna, Decision procedures for term algebras with integer constraints, Inf. Comput. 204(10) : 1526-1574, 2006.