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

Thi Bich Hanh Dao

▶ **To cite this version:**

Thi Bich Hanh Dao. Un algorithme de décision dans l'algèbre des arbres finis ou infinis et des queues. Cinquièmes Journées Francophones de Programmation par Contraintes, Orléans, juin 2009, Jun 2009, France. pp.345-355. hal-00387817

**HAL Id: hal-00387817**
**https://hal.science/hal-00387817**

Submitted on 25 May 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

**Thi-Bich-Hanh Dao**

Laboratoire d'Informatique Fondamentale d'Orléans – Université d'Orléans
Bâtiment 3IA, Rue Léonard de Vinci – 45067 Orléans Cedex 2
thi-bich-hanh.dao@univ-orleans.fr

**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 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.

**Abstract**

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 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 sont des structures où nous pouvons ajouter un élément à la fin ou bien prendre l'élément du début. Ces deux opérations correspondent respectivement à des ajouts à droite et à gauche d'un élément dans une queue. Les queues avec ces opérations 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. Par exemple une liste $[a, b]$ est construite par l'unique construction $[a|[b|[]]]$, mais une queue $\langle a, b\rangle$ a plusieurs constructions différentes, par exemple en ajoutant $a$ à gauche de la queue vide puis $b$ à droite ou en ajoutant $b$ à gauche de la queue vide puis $a$ à gauche. Une procédure de décision dans les arbres ne peut donc pas s'appliquer aux queues. Tandis que la décidabilité de l'algèbre des arbres finis avec des queues est connu [13], celle des arbres finis ou infinis avec des queues à notre connaissance, reste une problème ouvert.

Un simple exemple d'utilisation de queues est dans l'analyse syntaxique, où l'on veut reconnaître les mots $a^n b^n$ générés par les règles $S \to \varepsilon \mid aSb$. Dans des implantations de la PL ou la PLC, avec des grammaires de clauses définies, les queues sont souvent représentées par des différences de listes. Une différence de liste est un terme $L1 - L2$, qui signifie la liste obtenue en coupant la liste $L2$ de la fin de $L1$, par exemple $[1, 2, 3, 4] - [3, 4]$ représente la liste $[1, 2]$. Voici un programme Prolog utilisant des différences de listes :

```
s(X0-X0).
s([a|X0]-X1) :- s(X0-[b|X1]).
```

Cependant, la différence de listes n'est qu'une astuce pour gérer la fin d'une liste. Nous nous plaçons ensuite dans l'algèbre des arbres étendus avec des queues. L'ensemble des symboles de fonctions est étendu de la

constante *nil* pour la queue vide et des symboles *al* et *ar* correspondant respectivement à l'ajout à gauche et l'ajout à droite d'un élément dans une queue. Les clauses seront plus simples et naturelles :

```
s(nil).
s(al(a,ar(X,b))) :- s(X).
```

Les arbres avec un ensemble infini de symboles de fonction modélisent la base de la PL ou de la PLC [2, 7]. L'algèbre qui modélise cette extension sera celle des arbres finis ou infinis étendus avec des queues, définis sur un ensemble infini de symboles de fonction. Il est important donc d'assurer la décidabilité dans cet algèbre. La décidabilité de cette extension a également un intérêt dans autres domaines de l'informatique comme la logique ou la vérification.

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 [11], K. L. Clark [1], K. Kunen [8] et H. Comon [3]. Pour les arbres finis ou infinis, M. Maher a proposé une axiomatisation complète ainsi qu'une procédure de décision [10]. Un algorithme de résolution de contraintes du premier ordre dans les algèbres des arbres est proposé en [4, 6] 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 dite flexible, par exemple la théorie des rationnels additifs, une axiomatisation ainsi qu'une procédure de décision sont proposées [5]. Cependant, les arbres avec des queues n'entrent pas dans ce cadre. De plus, 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. Par conséquent les méthodes générales de décision dans une combinaison de théories comme Nelson-Oppen [12] ne peuvent non plus s'appliquer. Dans le cadre de la vérification, T. Rybina et A. Voronkov ont proposé un algorithme de décision dans l'algèbre des arbres finis avec queues [13]. 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. A cause des contraintes d'espace, les preuves ne sont pas présentes. Une version complète est accessible en ligne à la page des rapports de recherche du Laboratoire d'Informatique Fondamentale d'Orléans (http ://www.univ-orleans.fr/lifo/).

## 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 $\alpha_1 \times \cdots \times \alpha_n \rightarrow \beta$, où $\alpha_1, \ldots, \alpha_n$ et $\beta$ sont des sortes de $S$. Soit $F$ un ensemble infini de symboles de fonction. A chaque élément de $F$ est associé un type $\alpha_1 \times \cdots \times \alpha_n \rightarrow arbre$ et le nombre $n$ est appelé l'*arité* de $f$. On se donne trois autres symboles $al, ar$ et $\varepsilon$ où $al$ est de type $\alpha \times queue \rightarrow queue$, $ar$ est de type $queue \times \alpha \rightarrow queue$ et $\varepsilon$ de type $\rightarrow queue$, 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 $\beta$ est soit une variable de cette sorte, soit de la forme $f(u_1, ..., u_n)$ où $f \in F$ de type $\alpha_1 \times \cdots \times \alpha_n \rightarrow \beta$ 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 :

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

Ici $t, s$ sont des termes et $\varphi, \psi$ 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 une suite d'éléments, à laquelle nous pouvons ajouter un élément à gauche (au début) ou à droite (à la fin). Les arbres et les queues représentent deux sortes

*arbre* et *queue* respectivement. Nous définissons maintenant l'interprétation de chaque symbole de fonction.

Pour chaque symbole de fonction $f$ de type $\alpha_1 \times \cdots \times \alpha_n \rightarrow$ *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 $\varepsilon$ est interprété par la queue vide. Le symbole $al$ est interprété par la fonction d'ajout à gauche $\underline{al} : D \times Q \rightarrow Q$ telle qu'avec $a \in D$, $\langle b_1, .., b_n \rangle \in Q$, $\underline{al}(a, \langle b_1, .., b_n \rangle) = \langle a, b_1, .., b_n \rangle$. Le symbole $ar$ est interprété par la fonction d'ajout à droite $\underline{ar} : Q \times D \rightarrow Q$ telle qu'avec $\langle b_1, .., b_n \rangle \in Q$, $a \in D$, $\underline{ar}(\langle b_1, .., b_n \rangle, a) = \langle b_1, .., b_n, a \rangle$.

**Propriétés des arbres** Elles sont formulées par l'ensemble des schémas d'axiomes suivants [10, 6] :

$$\begin{array}{ll} \forall \overline{x} \forall \overline{y}\ \neg(f(\overline{x}) = g(\overline{y})) & (a1) \\ \forall \overline{x} \forall \overline{y}\ (f(\overline{x}) = f(\overline{y}) \rightarrow \bigwedge_i x_i = y_i) & (a2) \\ \forall \overline{x} \exists! \overline{z}\ \bigwedge_i z_i = t_i(\overline{x}\overline{z}) & (a3) \end{array}$$

Ici $f, g \in F$ 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 $\overline{z}$. La forme $(a1)$ est appelée *conflit de symboles* qui indique que les arbres construits par des symboles de fonction différents sont différents. La forme $(a2)$ est appelée *explosion* qui indique que les arbres égaux doivent avoir l'égalité entre les sous-arbres respectifs. La forme $(a3)$ est appelée *solution unique*, qui indique que certaines formes de conjonctions d'équations ont une solution unique. Par exemple la formule $x = f(x)$ a une solution unique pour $x$, qui est l'arbre infini $f(f(f(...)))$.

**Propriétés des queues** Les queues satisfont les propriétés suivantes [13] :

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

Ici $n \neq m$, $x, y$ 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. Il est important de noter que $(q7)$ et $(q8)$ expriment le fait qu'une queue peut être construite de différentes façons, par exemple $\langle a, b \rangle$ est la queue construite par $al(a, ar(\varepsilon, b))$ ou par $ar(al(a, \varepsilon), b)$. Ces propriétés distinguent les queues des arbres.

De plus, les queues satisfont les propriétés suivantes concernant des mots. Les queues peuvent être considérées comme des mots finis sur $D$ et les opérations d'ajout à gauche et à droite peuvent être considérées comme des concaténations d'un mot avec une lettre. Soit $D^*$ l'ensemble des mots finis, $D^+$ l'ensemble des mots finis non vides sur $D$ et $\varepsilon$ 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 $\{u^n \mid n \geq 0\}$ $(\{u^n \mid n > 0\})$. 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.

**Propriété 1** [9] *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 $tx = xs$.*

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

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

**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.*

Ces propriétés montrent qu'avec des équations dans des queues, nous pouvons exprimer des queues avec des motifs répétés. Par exemple d'après la propriété 1, l'équation $[ab]x = x[ba]$ correspond à un ensemble infini des queues finis $\langle a \rangle$, $\langle a, b, a \rangle$, $\langle a, b, a, b, a \rangle$, ... (la forme $(ab)^*a$), d'après la propriété 2, la conjonction $[ab]x = x[ba] \wedge [aba]x = x[aba]$ donne une solution unique $\langle a, b, a \rangle$, et d'après la propriété 3, la conjonction $[aba]x = x[aba] \wedge [aba]x = x[aab]$ n'a pas de solution.

## 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 $\varepsilon$, $[\overline{u}]\varepsilon[\overline{v}]$ 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. Une conjonction $c$ de contraintes élémentaires est *complètement typée* si pour toute variable $x$ ayant une occurrence dans $c$, soit $queue(x)$ soit $arbre(x)$ est dans $c$.

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 $Acc_c(u)$.

**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 $c$, $x \notin Acc_c(x)$.*

Dans l'algorithme de décision présenté dans la section 4, nous allons maintenir les formules sous la forme d'une disjonction de formules basiques, dont la définition est comme suit.

**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*.

Notons que dans une formule basique, des conjonctions de contraintes élémentaires sont sous forme résolue. Nous présentons dans la sous-section 3.1 suivante l'algorithme permettant de transformer une conjonction de contraintes élémentaires en forme résolue.

### 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.

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

- 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*.

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

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 $\varepsilon$, 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

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

   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 $|\overline{u}| > |\overline{v}|$ :

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

   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 $k \neq l$ :

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

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

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

**Propriété 4** *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* $c_i$*.*

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

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

## 4 Algorithme de décision

Nous présentons dans cette section un algorithme pour décider la valeur de vérité d'une formule $F$ du premier ordre. L'algorithme consiste à transformer $F$ en forme prénexe $QF'$, à transformer la matrice $F'$ sans quantificateur en une disjonction de formules basiques et à éliminer successivement les quantificateurs de $Q$ du plus interne au plus externe. L'algorithme est détaillé dans la sous-section 4.1 et les techniques d'élimination de quantificateur sont présentées dans 4.2. Nous discutons la décidabilité dans la sous-section 4.3.

### 4.1 Algorithme

Soit $F$ une formule du premier-ordred. 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

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

Cette formule est transformée en la formule :

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

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

$$\bigvee \exists \overline{u}(c \wedge \bigwedge_{i \in I} \neg \exists \overline{u_i} c_i). \quad (1)$$

En mettant les conjonctions $c$ en forme résolue pour celles qui ne le sont pas encore, on obtient une formule toujours de la forme (1) 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 procédons ensuite l'élimination des quantificateurs de $Q$.

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 $\exists \overline{u}(c \wedge \bigwedge \neg \exists \overline{u_i} c_i)$. Elle est transformée en une conjonction $\exists \overline{u} c \wedge \bigwedge \neg \exists \overline{u} \exists \overline{u_i}(c \wedge c_i)$. Nous transformons ensuite $(\neg F_1 \wedge \ldots \wedge \neg F_n)$ 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. L'objectif de l'élimination du quantificateur $\exists x$ est de transformer $\exists x b$ en une disjonction de formules basiques. Nous présentons cette élimination en deux parties, dépendant de la forme de $b$ : (1) lorsque $b$ ne contient pas de négation, et (2) lorsque $b$ contient des négations.

#### 4.2.1 Elimination dans une conjonction résolue

Lorsque la formule basique $b$ ne contient pas de négation, il est de la forme $\exists \overline{u} c$, avec $c$ une conjonction résolue. S'il existe une variable libre $y$ telle que $x \in Acc_c(y)$, alors $\exists x \exists \overline{u} c$ est déjà une formule basique. 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 $c$. L'élimination de $x$ dans $\exists x \exists \overline{u} c$ retourne $\exists \overline{u'} c'$.

**Exemple 2** *Eliminons $\exists x$ dans $\exists x \exists u v_1 v_2([v_1 v_2]x = x[v_1 v_2] \wedge v_1 = f(u) \wedge y = f(u) \wedge queue(x) \wedge arbre(v_1) \wedge queue(v_2) \wedge arbre(y) \wedge queue(u))$. 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 se réécrit en :*

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

*et le résultat de l'élimination est*

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

#### 4.2.2 Elimination dans une formule basique

Soient $x$ une variable et $b$ une formule basique. Nous allons transformer $\exists x b$ 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 avec les transformations correspondantes. 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 $Acc^a_c(x)$ 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 si une variable $v \in x\overline{v}$ est dans $c'$, puisque $c$ est résolue, $v$ ne peut pas être un représentant de $c'$, on a donc $v \in A^q_c(x)$, ce qui est contraire à la définition de $\overline{v}$. La formule $\exists x b$ se réécrit en :

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

pour être transformée en :

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

La procédure 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 $c$** 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 $arbre(x)$. L'élimination de $x$ dans $\exists xb$ retourne :

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

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 $queue(x)$. La formule $\exists xb$ est transformée en :

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

La procédure 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 $\exists x$ :*

$$\begin{array}{ll} & \exists x \exists yv(x=[v]y \wedge \neg(x=\varepsilon) \wedge \neg\exists uz(x=[u]z)) \\ \equiv & \exists yv(\neg\exists x(x=[v]y \wedge x=\varepsilon) \wedge \neg\exists xuz(x=[v]y \wedge x=[u]z)) \\ \equiv & \exists yv(\neg false \wedge \neg\exists xuz(x=[u]z \wedge u=v \wedge z=y)) \\ \equiv & \exists yv(\neg true) \\ \equiv & false \end{array}$$

**(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 $queue(x)$ est dans $c_i$, nous pouvons la supprimer grâce à l'équivalence $queue(x) \wedge \neg\exists \overline{u_i}(c'_i \wedge queue(x)) \equiv queue(x) \wedge \neg\exists \overline{u_i} c'_i$. 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. L'élimination de $\exists x$ dans $\exists xb$ retourne :

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

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 [4, 6], nous montrons comment supprimer l'équation $[\overline{v}]x = x[\rho(l, \overline{v})]$. La formule $\exists xb$ est transformée en :

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

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 [4, 6]. L'équation $[\overline{v}]x = x[\rho(l, \overline{v})]$ est ainsi supprimée.

**Exemple 7** *Eliminer $\exists x$ :*

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

*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 éliminer 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 :*

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

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

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

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

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

*Le résultat est donc la formule false.*

**(e) $x$ est de sorte queue et $x$ n'a pas d'occurrence dans les équations de $c$** Les cas possibles 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. La formule $\exists xb$ est transformée en :

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

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 $\exists x$ :*

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

e2. Il existe une conjonction $c_i$ dans laquelle $x$ n'est pas accessible depuis une autre variable libre et $c_i$ 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 conservant l'équivalence nous ajoutons 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 éliminer 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 $\exists x$ :*

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

*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. La correction et la terminaison de cette procédure se résument dans le résultat suivant.

**Théorème 1** *Soit $x$ une variable et $b$ une formule basique. La procédure d'élimination de quantificateur transforme $\exists xb$ après un temps fini en une disjonction de formules basiques, équivalente à $\exists xb$ dans l'algèbre des arbres finis ou infinis avec des queues.*

Si l'algorithme s'applique sur une formule close, tous les quantificateurs seront éliminés par cette procédure. 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 étendus 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 travail restant à faire est de formuler une axiomatisation de la théorie du premier ordre des arbres finis ou infinis avec des queues. Une solution, comme en [13], est de rassembler les propriétés nécessaires pour assurer la correction de l'algorithme. Nous continuons à travailler sur cet aspect à fin de caractériser une formulation simple et concise. D'un autre côté, en se basant sur cet algorithme de décision, nous étudions des adaptations pour répondre à l'objectif de résoudre des contraintes du premier ordre dans cette algèbre.

**Remerciements** Nous remercions Alain Colmerauer pour les nombreuses discussions sur la théorie des arbres avec queues ainsi que 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] A. Colmerauer, An introduction to Prolog III, Commun. ACM 33(7), 69-90, 1990.

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

[4] 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.

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

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

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

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

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

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

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

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

[13] 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.