Résolution de contraintes du premier ordre dans la théorie des arbres évalués

Thi-Bich-Hanh Dao, Khalil Djelloul

▶ **To cite this version:**

Thi-Bich-Hanh Dao, Khalil Djelloul. Résolution de contraintes du premier ordre dans la théorie des arbres évalués. Journées Francophones de Programmation par Contraintes, 2006, Nîmes - Ecole des Mines d'Alès. inria-00085773

HAL Id: inria-00085773
https://inria.hal.science/inria-00085773

Submitted on 14 Jul 2006

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.

Résolution de contraintes du premier ordre dans la théorie des arbres évalués

Thi-Bich-Hanh Dao[1] et Khalil Djelloul[2]

[1]Laboratoire d'Informatique Fondamentale d'Orléans.
Bat. 3IA, rue Léonard de Vinci. 45067 Orléans, France.
[2]Laboratoire d'Informatique Fondamentale de Marseille.
Parc scientifique et technologique de Luminy. 13288 Marseille, France.
dao@univ-orleans.fr khalil.djelloul@lif.univ-mrs.fr

Résumé

Nous présentons dans ce papier un algorithme général de résolution de contraintes du premier ordre dans la théorie T des arbres évalués. Cette théorie est une combinaison de la théorie des arbres finis ou infinis et de la théorie des rationnels munis de l'addition, de la soustraction et d'une relation d'ordre dense sans extrême. L'algorithme est donné sous forme d'un ensemble de 28 règles de réécriture et transforme toute formule du premier ordre φ, qui peut éventuellement contenir des variables libres, en une disjonction ϕ de formules résolues, équivalente à φ dans T, sans nouvelles variables libres, et telle que ϕ est soit la formule $vrai$, soit la formule $faux$, soit une formule ayant au moins une variable libre et n'étant équivalente ni à $vrai$ ni à $faux$ dans T. En particulier, si φ est sans variables libres, ϕ est soit la formule $vrai$ soit la formule $faux$. Si ϕ contient des variables libres, les solutions sur ces variables sont exprimées d'une façon explicite et ϕ peut se transformer directement en une combinaison booléenne de conjonctions quantifiées de formules atomiques, qui n'acceptent pas d'élimination de quantificateurs. La correction de notre algorithme est une autre preuve de la complétude de la théorie T.

Abstract

We present in this paper a general algorithm for solving first-order constraints in the theory T of the evaluated trees which is a combination of the theory of finite or infinite trees and the theory of the rational numbers with addition and subtraction and a linear dense order relation. The algorithm is given in the form of 28 rewriting rules. It transforms a first-order formula φ - which can possibly contain free variables - into a disjunction ϕ of solved formulas which is equivalent in T, without new free variables and such that ϕ is either the formula $true$ or the formula $false$ or a formula having at least one free variable and being equivalent neither to $true$ nor to $false$ in T. In particular if φ does not contain free variables then ϕ is either $true$ or $false$. If ϕ has free variables then the solutions on free variables are expressed in an explicit way and ϕ can be directly transformed into a boolean combination of quantified conjunctions of atomic formulas which do not accept elimination of quantifiers. The correctness of our algorithm is another proof of the completeness of this theory.

1 Introduction

Les arbres finis ou infinis jouent un rôle fondamental en informatique, ils modélisent aussi bien des structures de données que des schémas ou des déroulements de programmes. Dès 1976, G. Huet a proposé un algorithme d'unification de termes infinis, c'est-à-dire la résolution d'équations dans cette algèbre [13]. B. Courcelle a étudié les propriétés des arbres infinis dans le cadre des programmes récursifs [10, 9]. A. Colmerauer a modélisé l'exécution des programmes de Prolog II, III et IV par la résolution d'équations et de diséquations dans cette algèbre [4, 3, 1].

L'unification des termes finis, ou la résolution d'équations dans la théorie des arbres finis a été étudiée au début par A. Robinson [21]. Des algorithmes de meilleure complexité ont été ensuite proposés par M.S. Paterson et M.N.Wegman [19] et A. Martelli et U. Montanari [18]. La résolution d'équations dans la théorie des arbres infinis a été étudiée par G. Huet [13], A. Colmerauer [5, 4] et J. Jaffar [14]. La résolution de conjonctions d'équations et de diséquations dans la théorie des arbres finis ou infinis a été étudiée par A.

Colmerauer [4] et H.J. Bürckert [2]. Un algorithme incrémental de résolution de conjonctions d'équations et de diséquations dans les arbres rationnels a été proposé par V.Ramachandran et P. Van Hentenryck [20]. Quant aux formules quantifiées, il existe un algorithme qui se base sur l'élimination de quantificateur et qui transforme une formule du premier ordre en une combinaison booléenne de formules basiques. Dans le cas des arbres finis, on peut citer le travail de A. Malcev [17], K. Kunen [15], M.J. Maher [16] et H. Comon [7, 8]. Dans le cas des arbres infinis avec un ensemble fini de symboles de fonction, on peut citer le travail de M.J. Maher [16] et H. Comon [7, 8] et avec un ensemble infini de symboles de fonction, l'article de M.J. Maher [16].

M.J. Maher a donné une axiomatisation complète de la théorie des arbres finis ou infinis munis d'un ensemble infini de symboles de fonction [16]. Nous avons ensuite introduit une axiomatisation complète de la théorie des arbres évalués T, qui est une combinaison de la théorie des arbres finis ou infinis munis des opérations de construction et de la théorie des rationnels munis de l'addition, de la soustraction et d'une relation d'ordre dense sans extrême [12]. Nous proposons dans ce papier un algorithme général pour la résolution de contraintes du premier ordre avec ou sans variables libres dans T. Bien entendu, l'objectif de ce travail n'est pas uniquement de décider la vérité de propositions, c'est-à-dire la valeur de vérité des formules sans variables libres, mais de pouvoir résoudre des formules avec variables libres et de présenter les solutions sur ces variables libres d'une façon simple et explicite comme il a été déjà fait dans [11] pour la théorie des arbres finis et la théorie des arbres éventuellement infinis. Par *résolution d'une contrainte φ dans T*, nous entendons transformer la formule φ, qui peut éventuellement contenir des variables libres, en une disjonction ϕ de formules résolues, équivalente à φ dans T, sans nouvelles variables libres, et telle que ϕ est soit la formule *vrai*, soit la formule *faux*, soit une formule contenant au moins une variable libre et n'étant équivalente ni à *vrai* ni à *faux* dans T. En particulier, si φ est une proposition, alors ϕ est soit la formule *vrai* soit la formule *faux*. Notons que les formules contenant des variables libres, mais étant toujours vraies, respectivement toujours fausses, dans T sont détectées par notre algorithme qui produit dans ce cas la formule *vrai*, respectivement la formule *faux*. Nous montrons également que si ϕ a au moins une variable libre alors ϕ peut se transformer d'une façon directe en une combinaison booléenne de conjonctions quantifiées de formules atomiques, qui n'acceptent pas d'élimination de quantificateurs. La correction de notre algorithme est une autre preuve de la complétude de T.

Notre algorithme n'est pas simplement une combinaison d'un algorithme sur les arbres et d'un sur les rationnels, mais un puissant mécanisme pour résoudre des contraintes mixes. Cet algorithme est capable de résoudre toute contraintes du premier ordre contenant des variables typées ou non typées et représente les solutions des variables libres d'une façon explicite. Une des difficultés réside aux faits que (1) la théorie des arbres n'admet pas d'élimination complète de variables quantifiées, (2) tout algorithme décidant la vérité de propositions dans la théorie des arbres doit être d'une complexité non élémentaire [22] et (3) les symboles de fonction $+$ et $-$ dans T possèdent deux comportements différents dépendant qu'ils s'appliquent sur des arbres ou des rationnels. Par exemple $+(1,1)$ est le rationnel 2, pendant que $+(1, f_0)$ est l'arbre dont la racine est étiquetée $+$ et dont les fils sont 1 et l'arbre f_0. Le résultat de complétude de T donné en [12] n'est pas suffisant pour déduire une forme résolues exprimant les solutions. Par exemple dans un jeu à deux joueurs nous ne contentons pas simplement de savoir s'il existe une stratégie gagnante, mais de pouvoir exprimer des stratégies gagnantes. Tant dis qu'en [12], l'algorithme traite seulement des propositions, notre algorithme est capable de traiter des formules générales du premier ordre avec des variables libres et des variables typées ou non typées.

Ce papier est organisé en quatre sections suivies d'une conclusion. Cette introduction constitue la première section. Dans la section 2, nous présentons la théorie T des arbres évalués et introduisons un exemple de contraintes dans cette théorie. Dans la section 3, nous définissons des formules basiques, blocs et blocs résolus, qui sont des conjonctions particulières de formules atomiques. Nous montrons que tout bloc résolu quantifié peut se décomposer en trois séquences imbriquées de quantifications ayant des propriétés particulières permettant d'éliminer des quantificateurs. Dans la section 4, nous présentons les formules de travail, les formules résolues générales et l'algorithme de résolution de contraintes dans T. L'algorithme est donné sous forme d'un ensemble de 28 règles de réécriture et transforme une formule de travail initiale de profondeur d en une formule de travail finale de profondeur inférieure ou égale à trois. La disjonction ϕ de formules résolues extraite de la formule de travail finale est soit la formule *vrai*, soit la formule *faux* soit une formule ayant au moins une variable libre et n'étant équivalente ni à *vrai* ni à *faux* dans T. A la fin de cette section, nous donnons un exemple de résolution de contraintes dans T. L'algorithme représenté par un ensemble de règles de réécriture, les blocs et les formules résolues générales sont notre contribution principale dans ce papier.

2 Théorie T des arbres évalués

2.1 Préliminaires

Soit F un ensemble infini de *symboles de fonction* contenant les symboles $+$, $-$, 0 et 1. A chaque élément de F est associé un entier non négatif, son *arité*. L'arité de $+$, $-$, 0 et 1 est respectivement 2, 1, 0 et 0. Soit $R = \{<, \mathit{num}\,, \mathit{arbre}\,\}$ l'ensemble de symboles de relation, d'arités respectives 2, 1 et 1. Soit V un ensemble infini énumérable de *variables*. Un *terme* est une expression de la forme x ou $ft_1 \dots t_n$ où $n \geq 0$, f un symbole n-aire de F et les t_i sont des termes plus courts. Une *formule* est une expression de l'une des formes

$s{=}t,\ rt_1..t_n,\ \mathit{vrai},\ \mathit{faux},$
$\neg(\varphi),\ (\varphi \wedge \psi),\ (\varphi \vee \psi),\ (\varphi{\rightarrow}\psi),\ (\varphi{\leftrightarrow}\psi),\ \exists x\varphi,\ \forall x\varphi,$

où $x \in V$, s, t et les t_i sont des termes, r est un symbole de relation n-aire et φ et ψ sont des formules plus courtes. Les quatre premières formes sont appelées atomiques. Une occurrence d'une variable x dans une formule est *liée* si elle apparaît dans une sous-formule de la forme $(\exists x\varphi)$ ou $(\forall x\varphi)$. Elle est *libre* sinon. Les *variables libres* d'une formule sont celles ayant au moins une occurrence libre dans la formule. Pour chaque formule φ, on note $\mathit{var}(\varphi)$ l'ensemble de toutes les variables libres de φ. Soient $\bar{x} = x_1 \dots x_n$ et $\bar{y} = y_1 \dots y_n$ deux vecteurs de variables de même longueur. Le vecteur vide est noté ε. Soient φ et $\varphi(\bar{x})$ des formules. Nous écrivons

$\exists\bar{x}\,\varphi$ pour $\exists x_1 ... \exists x_n\,\varphi$,
$\forall\bar{x}\,\varphi$ pour $\forall x_1 ... \forall x_n\,\varphi$,
$\exists?\bar{x}\,\varphi(\bar{x})$ pour $\forall\bar{x}\forall\bar{y}\,\varphi(\bar{x}) \wedge \varphi(\bar{y}) \rightarrow \bigwedge_{i\in\{1,\dots,n\}} x_i = y_i$,
$\exists!\bar{x}\,\varphi$ pour $(\exists\bar{x}\,\varphi) \wedge (\exists?\bar{x}\,\varphi)$.

2.2 Axiomatisation de T

Soit a un entier positif et soient $t_1, ..., t_n$ des termes. On note :

$t_1 < t_2$, le terme $< t_1 t_2$,
$t_1 + t_2$, le terme $+t_1 t_2$,
$t_1 + t_2 + t_3$, le terme $+t_1(+t_2 t_3)$,
$-at_1$, le terme $(-t_1) + \dots + (-t_1)$ avec a fois $-t_1$
$0t_1$, le terme 0,
at_1, le terme $t_1 + \dots + t_1$ avec a fois t_1,
a le terme $1 + \dots + 1$ avec a fois 1.

La théorie T des arbres évalués est l'ensemble de propositions de l'une des formes suivantes :

1 $\forall\bar{x}\forall\bar{y}\,(\mathit{arbre}\,f\bar{x} \wedge \mathit{arbre}\,f\bar{y} \wedge f\bar{x} = f\bar{y}) \rightarrow \bigwedge_i x_i = y_i$,
2 $\forall\bar{x}\forall\bar{y}\,f\bar{x} = g\bar{y} \rightarrow \mathit{num}\,f\bar{x} \wedge \mathit{num}\,g\bar{y}$,
3 $\forall\bar{x}\forall\bar{y}\,(\bigwedge_{i\in I} \mathit{num}\,x_i) \wedge (\bigwedge_{j\in J} \mathit{arbre}\,y_j)$
$\rightarrow (\exists!\bar{z}\ \bigwedge_{k\in K}(\mathit{arbre}\,z_k \wedge z_k = t_k(\bar{x},\bar{y},\bar{z})))$,
4 $\mathit{num}\,0$,
5 $\mathit{num}\,1$,
6 $\forall x\forall y\,x < y \rightarrow (\mathit{num}\,x \wedge \mathit{num}\,y)$,
7 $\forall x\forall y\,\mathit{num}\,x + y \leftrightarrow \mathit{num}\,x \wedge \mathit{num}\,y$,
8 $\forall x\,\mathit{num}\ -x \leftrightarrow \mathit{num}\,x$,
9 $\forall\bar{x}\ \mathit{arbre}\ h\bar{x}$,
10 $\forall x\forall y\,(\mathit{num}\,x \wedge \mathit{num}\,y) \rightarrow x + y = y + x$,
11 $\forall x\forall y\forall z\,(\mathit{num}\,x \wedge \mathit{num}\,y \wedge \mathit{num}\,z)$
$\rightarrow x + (y + z) = (x + y) + z$,
12 $\forall x\,\mathit{num}\,x \rightarrow x + 0 = x$,
13 $\forall x\,\mathit{num}\,x \rightarrow x + (-x) = 0$,
14_n $\forall x\,\mathit{num}\,x \rightarrow (nx = 0 \rightarrow x = 0)$,
15_n $\forall x\,\mathit{num}\,x \rightarrow \exists!y\,\mathit{num}\,y \wedge ny = x$,
16 $\forall x\,\mathit{num}\,x \rightarrow \neg x < x$,
17 $\forall x\forall y\forall z\,\mathit{num}\,x \wedge \mathit{num}\,y \wedge \mathit{num}\,z$
$\rightarrow ((x < y \wedge y < z) \rightarrow x < z)$,
18 $\forall x\forall y\,(\mathit{num}\,x \wedge \mathit{num}\,y) \rightarrow (x < y \vee x = y \vee y < x)$,
19 $\forall x\forall y\,(\mathit{num}\,x \wedge \mathit{num}\,y)$
$\rightarrow (x < y \rightarrow (\exists z\,\mathit{num}\,z \wedge x < z \wedge z < y))$,
20 $\forall x\,\mathit{num}\,x\ \rightarrow (\exists y\,\mathit{num}\,y \wedge x < y)$,
21 $\forall x\,\mathit{num}\,x\ \rightarrow (\exists y\,\mathit{num}\,y \wedge y < x)$,
22 $\forall x\,\forall y\,\forall z\,(\mathit{num}\,x \wedge \mathit{num}\,y \wedge\ \mathit{num}\ z)$
$\rightarrow (x < y \rightarrow (x + z < y + z))$,
23 $0 < 1$,

où n est un entier non nul, f et g sont deux symboles de fonction distincts de F, $h \in F - \{+, -, 0, 1\}$, x, y, z sont des variables, $\bar{x}$ est un vecteur de variables x_i, $\bar{y}$ est un vecteur de variables y_i, $\bar{z}$ est un vecteur de variables distinctes z_i et où $t_k(\bar{x},\bar{y},\bar{z})$ est un terme qui commence par un symbole de fonction $f_k \in F - \{0, 1\}$ suivi de variables prises dans $\bar{x}$ ou $\bar{y}$ ou $\bar{z}$, de plus, si $f_k \in \{+, -\}$ alors $t_k(\bar{x},\bar{y},\bar{z})$ contient au moins une variable de $\bar{y}$ ou $\bar{z}$.

Cette théorie T a été introduite en [12], où nous avons prouvé sa complétude. T a comme modèle les arbres éventuellement infinis dont les nœuds sont étiquetés par $\boldsymbol{Q} \cup F$, avec $\boldsymbol{Q}$ l'ensemble des rationnels, tels que chaque sous-arbre étiqueté par $\boldsymbol{Q} \cup \{+, -\}$ est évalué dans $\boldsymbol{Q}$ et est réduit à une feuille étiquetée par un élément de $\boldsymbol{Q}$.

Nous présentons maintenant un exemple de *contrainte* dans T. Considérons ce jeu à deux joueurs : une paire de rationnels non négatifs (n, m) est donnée. Chaque joueur à tour de rôle, soustrait 1 ou 2 de n ou de m sans les rendre négatifs. Le premier qui ne peut plus jouer perd.

Supposons que c'est au tour de A de jouer. Une position (n, m) est appelée *k-gagnante* si, quelque soit la façon de jouer de l'autre joueur B, il est toujours possible pour A de gagner après avoir fait au plus k coups. La contrainte $\mathit{gagnant}_k(x)$ définie dans [6] exprimant qu'une position x est k-gagnante est :

$$\left[\begin{array}{l} \quad \exists y \mathit{move}(x,y) \wedge \neg(\exists x \mathit{move}(y,x) \wedge \\ \neg(\exists y \mathit{move}(x,y) \wedge \neg(\exists x \mathit{move}(y,x) \wedge \neg(\dots \wedge \\ \neg(\exists y \mathit{move}(x,y) \wedge \neg(\exists x \mathit{move}(y,x) \wedge \neg(\mathit{faux}\underbrace{))\dots)}_{2k} \end{array}\right]$$

Chaque position (n, m) est représentée par $c(i, j)$ avec c un symbole de fonction d'arité 2 et $i, j \in \boldsymbol{Q}$. La contrainte $move(x, y)$ peut être définie par :

$$\left[\begin{array}{l}(\exists i \exists j\, x = c(i,j) \wedge y = c(i-1,j) \wedge i > 1 \wedge j > 0) \vee \\ (\exists i \exists j\, x = c(i,j) \wedge y = c(i-2,j) \wedge i > 2 \wedge j > 0) \vee \\ (\exists i \exists j\, x = c(i,j) \wedge y = c(i,j-1) \wedge i > 0 \wedge j > 1) \vee \\ (\exists i \exists j\, x = c(i,j) \wedge y = c(i,j-2) \wedge i > 0 \wedge j > 2) \vee \\ (\neg(\exists i \exists j\, x = c(i,j) \wedge num\, i \wedge num\, j) \wedge x = y)\end{array}\right]$$

En remplaçant la définition de *move* dans la contrainte $gagnant_k(x)$, nous obtenons une contrainte du premier ordre avec une variable libre x dans la théorie T des arbres évalués. Résoudre cette contrainte signifie trouver toutes les positions x qui sont k-gagnantes.

3 Blocs et blocs quantifiés dans T

3.1 Formules basiques et blocs dans T

Supposons que les variables de V soient ordonnées par un ordre strict, dense et sans extrêmes, noté $\succ$. Pour chaque formule φ, les variables liées sont renommées de telle façon que pour chaque sous-formules de φ on a $x \succ y$ pour chaque variable liée x et chaque variable libre y. On note $\Sigma_{i=1}^n t_i$ le terme $\overline{t_1 + \ldots + t_n} + 0$ avec $\overline{t_1 + \ldots + t_n}$ le terme $t_1 + \ldots + t_n$ où tous les termes 0 ont été supprimés.

On appelle *représentant* de l'équation $x_0 = f x_1 ... x_n$ ou $x_0 = x_1$, avec $f \in F - \{0, 1\}$, la variable x_0. On appelle représentant de la formule $\Sigma_{i=1}^n a_i x_i = a_0 1$, avec $a_0 \in \boldsymbol{Z}$ et $a_i \in \boldsymbol{Z}$ ($\boldsymbol{Z}$ l'ensemble des entiers), la plus grande variable x_k dans l'ordre $\succ$ telle que $a_k \neq 0$.

Soient $f \in F$, $a_0 \in \boldsymbol{Z}$ et $a_i \in \boldsymbol{Z}$. On appelle *formule basique* toute conjonction α de formules de la forme :

- *vrai*, *faux*, $num\, x$, $arbre\, x$,
- $x = y$, $x = f y_1 ... y_n$, $\Sigma_{i=1}^n a_i x_i = a_0 1$, $\Sigma_{i=1}^n a_i x_i < a_0 1$.

Les formules $num\, x$ et $arbre\, x$ sont appelées *contraintes de typage*. Les formules $x = y$, $x = f y_1 ... y_n$ et $\Sigma_{i=1}^n a_i x_i = a_0 1$ sont appelées *équations*. La formule $\Sigma_{i=1}^n a_i x_i < a_0 1$ est appelée *inéquation*. Soit α une formule basique :

On dit que $num\, x$ est *une conséquence* de α ssi α contient au moins une des sous-formules suivantes : $num\, x$, $x = y \wedge num\, y$, $y = x \wedge num\, y$, $x = -y \wedge num\, y$, $y = -x \wedge num\, y$, $z = y + x \wedge num\, z$, $z = x + y \wedge num\, z$, $x = y + z \wedge num\, z \wedge num\, y$, $x = 0$, $x = 1$, $\Sigma_i a_i x_i = a_0 1$, $\Sigma_i a_i x_i < a_0 1$, avec x figurant parmi les x_i.

On dit que $arbre\, x$ est *une conséquence* de α ssi α contient au moins une des sous-formules suivantes : $arbre\, x$, $x = y \wedge arbre\, y$, $y = x \wedge arbre\, y$, $x = -y \wedge arbre\, y$, $y = -x \wedge arbre\, y$, $x = y + z \wedge arbre\, z$, $x = z + y \wedge arbre\, z$, $y = x + z \wedge arbre\, y \wedge num\, z$, $y = z + x \wedge arbre\, y \wedge num\, z$, $x = h y_1 ... y_n$, avec $h \in F - \{+, -, 0, 1\}$.

On appelle la *section arborescente* α_t de α la conjonction des sous-formules de α de la forme : *vrai*, $arbre\, x$, $x = y$ ou $x = f y_1 ... y_n$, avec $f \in F - \{0, 1\}$ et où x est telle que $arbre\, x$ est une sous-formule de α. Cette section α_t est *formatée* ssi les membres gauches des équations de α_t sont tous distincts et pour chaque équation de la forme $x = y$ de α_t on a $x \succ y$.

On appelle *section numérique* α_n de α la conjonction des sous-formules de α de la forme :

- *vrai*, *faux*, $\Sigma_{i=1}^n a_i x_i = a_0 1$, $\Sigma_{i=1}^n a_i x_i < a_0 1$, $num\, x$,
- $x = y$, $x = -y$, $x = y + z$, où x est telle que $num\, x$ est une sous-formule de α.

Cette section α_n est *consistante* ssi $T \models \exists \bar{x}\, \alpha_n$ avec $\bar{x} = var(\alpha_n)$. Elle est dite *formatée* ssi :

- α_n ne contient pas de sous-formule de la forme $x = y$, $x = -y$, $x = y + z$, $0 = a_0 1$, $0 < a_0 1$, avec $a_0 \in \boldsymbol{Z}$,
- α_n est consistante et chaque représentant d'une équation de α_n a une occurrence dans une seule équation de α_n et pas d'occurrences dans les inéquations de α_n.

Une variable u est dite *accessible* dans $\exists \bar{x} \alpha$ si : soit u est une variable libre dans $\exists \bar{x} \alpha$ ou α a une sous-formule de la forme $y = t(u) \wedge arbre\, y$ avec $t(u)$ un terme contenant u et y est une variable accessible. Dans le dernier cas, l'équation $y = t(u)$ est aussi appelée accessible dans $\exists \bar{x} \alpha$.

Exemple 3.1.1 *Dans la formule $\exists x y z\, w = f x y \wedge z = v \wedge arbre\, w$, les variables w, v, x, y sont accessibles car w, v sont libres et x et y figurent dans la sous-formule $w = f x y \wedge arbre\, w$. La variable z n'est pas accessible et puisque z est liée et v est libre, elles doivent être telles que $z \succ v$. L'équation $w = f x y$ est accessible tant dis que $z = v$ ne l'est pas.*

On appelle *bloc* toute formule basique α telle que pour chaque variable x dans α soit $num\, x$ soit $arbre\, x$ est une sous-formule de α et α ne contient pas de sous-formules de la forme :

- $x = 0 \wedge arbre\, x$, $x = 1 \wedge arbre\, x$,
- $x = y \wedge num\, x \wedge arbre\, y$, $x = y \wedge arbre\, x \wedge num\, y$,
- $x = -y \wedge arbre\, x \wedge num\, y$, $x = -y \wedge num\, x \wedge arbre\, y$
- $x = y + z \wedge num\, x \wedge arbre\, y$, $x = y + z \wedge num\, x \wedge arbre\, z$, $x = h \bar{y} \wedge num\, x$,
- $x = y + z \wedge arbre\, x \wedge num\, y \wedge num\, z$,
- $\Sigma_{i=1}^n a_i x_i = a_0 1 \wedge arbre\, x_k$, $\Sigma_{i=1}^n a_i x_i < a_0 1 \wedge arbre\, x_k$

avec $h \in F - \{+, -, 0, 1\}$, $k \in \{1, ..., n\}$, $a_0 \in \boldsymbol{Z}$ et $a_i \in \boldsymbol{Z}$.

Du fait que chaque variable x dans un bloc a un typage $num\, x$ ou $arbre\, x$, tout bloc α peut être divisé en deux sections disjointes : une section arborescente et une section numérique.

Un bloc α sans équations est appelé *bloc relationnel*. Un bloc α sans inéquations et où chaque variable a une occurrence dans au moins une équation de α est appelé *bloc équationnel*. Un bloc α est *résolu* ssi sa section arborescente et sa section numérique sont formatées.

3.2 Décomposition de blocs quantifiés

Soit ψ une formule. Soient $\bar{x}$ un vecteur de variables et α un bloc résolu tels que pour chaque variable quantifiée inaccessible u dans $\exists\bar{x}\alpha$ et chaque variable quantifiée accessible v dans $\exists\bar{x}\alpha$ on a $u \succ v$. On appelle *décomposition* de la formule $\exists\bar{x}\alpha \wedge \psi$ la formule

$$\exists\bar{x}^1\,\alpha^1 \wedge (\exists\bar{x}^2\,\alpha^2 \wedge (\exists\bar{x}^3\,\alpha^3 \wedge \psi))), \qquad (1)$$

obtenue comme suit : soit X l'ensemble des variables de $\bar{x}$. On divise X en deux sous-ensembles disjoints X_{acc} (l'ensemble des éléments de X qui sont accessibles dans $\exists\bar{x}\alpha$) et X_{inacc}. Soit *Repr* l'ensemble des représentants des équations de α. On a :
– $\bar{x}^1$ est le vecteur des variables de X_{acc} ;
– $\bar{x}^2$ est le vecteur des variables de $X_{inacc} - Repr$;
– $\bar{x}^3$ est le vecteur des variables de $X_{inacc} \cap Repr$;
– α^1 est de la forme $\alpha^1_1 \wedge \alpha^1_2$ où α^1_1 est la conjonction de toutes les équations dans $\exists\bar{x}\alpha$ dont le représentant est accessible, α^1_2 est la conjonction de toutes les contraintes de typage de α qui concernent des variables de $var(\alpha^1_1)$;
– α^2 est de la forme $\alpha^2_1 \wedge \alpha^2_2$ où α^2_1 est la conjonction de toutes les inéquations de α et α^2_2 est la conjonction de toutes les contraintes de typage de α qui ne concernent pas les variables de $\bar{x}^3$;
– α^3 est de la forme $\alpha^3_1 \wedge \alpha^3_2$ où α^3_1 est la conjonction des autres équations et α^3_2 est la conjonction des contraintes de typage de α qui concernent des variables de $var(\alpha^3_1)$.

Soit A l'ensemble de blocs résolus. Soit A^1 l'ensemble de formules de la forme $\exists\bar{x}^1\alpha^1$, où α^1 est un bloc équationnel résolu et les variables de $\bar{x}^1$ sont toutes accessibles dans $\exists\bar{x}^1\alpha^1$. Soit A^2 l'ensemble de blocs relationnels résolus.

Propriété 3.2.1 *Pour toute formule décomposée de la forme (1) on a :* $\exists\bar{x}^1\alpha^1 \in A^1$, $\alpha^2 \in A^2$, $\alpha^3 \in A$ *et* $T \models \forall\bar{x}^2\,\alpha^2 \rightarrow \exists!\bar{x}^3\alpha^3$.

Exemple 3.2.2 *Soient* v, w, x, y, z *des variables telles que* $w \succ y \succ z \succ x \succ v$. *Décomposons la formule*

$$\begin{bmatrix} \exists wxyz \\ v = fvx \wedge w + 2x + (-2)z = 1 \wedge y + 3z = 0 \wedge \\ z < 1 \wedge 3z + 2x < 0 \wedge \\ arbre\, v \wedge num\, w \wedge num\, x \wedge num\, y \wedge num\, z \end{bmatrix}$$

Les variables accessibles de cette formule sont v *et* x. *On a donc* $X_{acc} = \{x\}$, $X_{inacc} = \{w, y, z\}$ *et* $Repr = \{v, w, y\}$. *Du fait que* $w \succ y \succ z \succ x$ *cette formule est décomposée en*

$$\begin{bmatrix} \exists x\, v = fvx \wedge arbre\, v \wedge num\, x \wedge \\ \begin{bmatrix} \exists z\, z < 1 \wedge 3z + 2x < 0 \wedge num\, z \wedge numx \wedge arbre\, v \wedge \\ \begin{bmatrix} \exists wy\, w + 2x + (-2)z = 1 \wedge y + 3z = 0 \wedge \\ num\, w \wedge num\, x \wedge num\, y \wedge num\, z) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Remarquons que les éléments de A^1 *n'acceptent pas d'élimination de quantificateurs, du fait que les variables de* $\bar{x}^1$ *sont accessibles dans* $\exists\bar{x}^1\,\alpha^1$. *En effet, dans la formule* $\exists x\, v = fvx$ *la quantification* $\exists x$ *ne peut être éliminée dans* T. *Une propriété similaire a été montrée par M.J. Maher dans la théorie des arbres finis ou infinis [16].*

Dans ce qui suit nous utiliserons les notations $\bar{x}^1$, $\bar{x}^2$, $\bar{x}^3$, α^1, α^2,α^3 en nous référant à la décomposition de la formule $\exists\bar{x}\alpha$.

4 Résolution de contraintes du premier ordre dans T

4.1 Formules de travail et formules résolues

Définition 4.1.1 *Une formule normalisée* φ *de profondeur* $d \geq 1$ *est une formule de la forme*

$$\neg(\exists\bar{x}\,\alpha \wedge \textstyle\bigwedge_{i\in I}\varphi_i), \qquad (2)$$

avec I *un ensemble fini éventuellement vide,* α *une formule basique et les* φ_i *des formules normalisées de profondeur* d_i *et* $d = 1 + \max\{0, d_1, ..., d_n\}$.

Il est simple de transformer une formule du premier ordre en formule normalisée équivalente dans T. Il suffit par exemple : (1) d'introduire des équations supplémentaires et des nouvelles variables quantifiées pour mettre les conjonctions d'équations en conjonctions de formules atomiques à plat, (2) de représenter tous les quantificateurs, constantes et connecteurs logiques par $\neg$, $\wedge$ et $\exists$, (3) d'enlever les doubles négations, $\neg\neg\varphi$ devient φ, (4) de remplacer les occurrences de $\neg num\, x$ par $arbre\, x$ et $\neg arbre\, x$ par $num\, x$, (5) si la formule φ ne commence pas par $\neg$, la remplacer par $\neg\neg\varphi$, (6) de renommer les variables quantifiées par des noms aussi différents que possible, et différents de ceux des variables libres, (7) de mettre le quantificateur au début des conjonctions, $\varphi \wedge (\exists\bar{x}\,\psi)$ devient $(\exists\bar{x}\,\varphi \wedge \psi)$ (car les variables libres de φ sont différentes de celles de $\bar{x}$).

Définition 4.1.2 *Une formule de travail est une formule normalisée dont toutes les occurrences de* $\neg$ *sont remplacées par* $\neg^k$ *avec* $k \in \{0, ..., 9\}$ *et telle que chaque occurrence d'une sous-formule de la forme*

$$\phi = \neg^k(\exists\bar{x}\,\alpha^c \wedge \alpha^p \wedge \textstyle\bigwedge_{i\in I}\varphi_i), \qquad (3)$$

est telle que $\alpha^p = vrai$ *si* $k = 0$ *et* **satisfait les** k **premières conditions** *de la liste suivante de conditions si* $k > 0$. *Ici* α^p *est un bloc résolu appelé section de contraintes propagées (propagated section),* α^c *est une formule basique appelé section de contraintes de noyaux (the core section), les* φ_i *sont des formules de travail. Dans les conditions ci-dessous,* $\beta^p \wedge \beta^c$ *est la conjonction des équations et des relations de la surformule immédiate* ψ *de* ϕ *si elle existe, c'est-à-dire* $\psi = \neg^k(\exists \bar{y} \beta^c \wedge \beta^p \wedge \phi \wedge \bigwedge_{j \in J} \phi_j)$ *avec les* ϕ_j *des formules de travail.*

1. *si* ψ *existe alors* $T \models \alpha^p \wedge \alpha^c \rightarrow \beta^p \wedge \beta^c$, *et les sections arborescentes de* α^p *et de* $\beta^c \wedge \beta^p$ *ont le même ensemble de membres gauches d'équations,*
2. *la section arborescente de* $\alpha^p \wedge \alpha^c$ *est formatée et la formule* $\alpha^p \wedge \alpha^c$ *ne contient pas arbre* $x \wedge num\, x$ *pour toute variable* x,
3. $\alpha^p \wedge \alpha^c$ *est un bloc,*
4. *la section numérique de* $\alpha^p \wedge \alpha^c$ *est consistante, et on a* $u \succ v$ *pour chaque variable inaccessible* u *de* $\bar{x}$ *et chaque variable accessible* v *de* $\bar{x}$,
5. $\alpha^p \wedge \alpha^c$ *est un bloc résolu,*
6. α^p *est la formule* $\beta^c \wedge \beta^p$ *si* ψ *existe, et la formule vrai sinon. La formule* α^c *est un bloc résolu et pour chaque relation* $num\, x$ *(ou* $arbre\, x$*) dans* α^p, *si* x *n'apparaît pas dans une équation ou une inéquation de* α^c *alors* $num\, x$ *(resp.* $arbre\, x$*) n'apparaît pas dans* α^c,
7. $(\exists \bar{x}\, \alpha^c)$ *est décomposée en* $(\exists \bar{x}^1\, \alpha^{c1} \wedge (\exists \bar{x}^2\, \alpha^{c2} \wedge (\exists \varepsilon\, vrai)))$,
8. $(\exists \bar{x}\, \alpha^c)$ *est décomposée en* $(\exists \bar{x}^1\, \alpha^{c1} \wedge (\exists \varepsilon\, \alpha^{c2} \wedge (\exists \varepsilon\, vrai)))$,
9. $(\exists \bar{x}\, \alpha^c)$ *est décomposée en* $(\exists \bar{x}^1\, \alpha^{c1} \wedge (\exists \varepsilon\, vrai \wedge (\exists \varepsilon\, vrai)))$.

L'utilisation de $\neg^k$ permet de contrôler l'exécution des règles de réécriture sur des formules de travail. On appelle formule de travail *initiale* une formule de travail de la forme

$$\neg^6(\exists \varepsilon\, vrai \wedge \bigwedge_{i \in I} \varphi_i)$$

avec les φ_i des formules de travail où tout symbole de négation $\neg^k$ est tel que $k = 0$ et toute section de contraintes propagées est réduite à la formule *vrai*. On appelle formule de travail *finale* une formule de travail de la forme

$$\neg^7(\exists \varepsilon\, vrai \wedge \bigwedge_{i \in I} \neg^8(\exists \bar{x}_i\, \alpha_i^c \wedge \alpha_i^p \wedge \bigwedge_{j \in J_i} \neg^9(\exists \bar{y}_{ij}\, \beta_{ij}^c \wedge \beta_{ij}^p))), \tag{4}$$

où les β_{ij}^c sont différents de *vrai*.

Définition 4.1.3 *Une* formule résolue générale *est une formule de la forme*

$$\exists \bar{x}^1\, \alpha^1 \wedge \alpha^2 \wedge \bigwedge_{i \in I} \neg(\exists \bar{y}_i^1\, \beta_i^1), \tag{5}$$

où $\exists \bar{x}^1\, \alpha^1 \in A^1$, $\alpha^2 \in A^2$, $\exists \bar{y}_i^1\, \beta_i^1 \in A^1$, *tous les* $\alpha^1 \wedge \alpha^2 \wedge \beta_i^1$ *sont des blocs résolus et tous les* β_i^1 *sont différents de vrai.*

D'après les propriétés de $\neg^8$ et $\neg^9$, dans la formule de travail finale (4), $\alpha_i^p = vrai$ et $\beta_{ij}^p = \alpha_i^p \wedge \alpha_i^c$. Donc cette formule est équivalente dans T à la disjonction de formules résolues générales

$$\bigvee_{i \in I} (\exists \bar{x}_i\, \alpha_i^c \wedge \bigwedge_{j \in J_i} \neg(\exists \bar{y}_{ij}\, \beta_{ij}^c)). \tag{6}$$

On a donc la propriété suivante :

Propriété 4.1.4 *Toute formule de travail finale de la forme (4) est équivalente dans* T *à la disjonction (6) de formules résolues générales.*

Propriété 4.1.5 *Soit* φ *une formule résolue générale de la forme (5). Si* φ *n'a pas de variable libre alors* φ *est la formule vrai, sinon on n'a ni* $T \models \varphi$ *ni* $T \models \neg\varphi$.

Propriété 4.1.6 *Toute formule résolue générale est équivalente dans* T *à une combinaison booléenne de formules de la forme* $\exists \bar{x}^1 \alpha^1 \wedge \alpha^2$, *avec* $\exists \bar{x}^1 \alpha^1 \in A^1$ *et* $\alpha^2 \in A^2$, *qui n'acceptent pas d'élimination de quantificateurs.*

4.2 Idée principale

L'algorithme de résolution de contraintes du premier ordre dans T utilise un système de règles de réécriture. L'idée principale est de transformer une formule de travail initiale de profondeur d en une formule de travail finale de profondeur inférieure ou égale à 3. La transformation est effectuée en 2 étapes :

1. La première étape est un parcours descendant de structure de formule de travail avec simplification et propagation de contraintes. Dans chaque sous-formule de travail, $\alpha^c \wedge \alpha^p$ est transformé en bloc résolu, puis $\exists \bar{x} \alpha^c$ est décomposé en 3 parties comme présenté dans la sous-section 3.2. La troisième partie est éliminée et ajoutée à la section de contraintes noyaux des sous-formules de travail immédiates, en utilisant une propriété du quantificateur $\exists!$. Les contraintes des deux autres parties sont propagées dans la section de contraintes propagées des sous-formules de travail immédiates. A cette étape, les règles 1 à 24 sont applicables et transforment la formule de travail initiale en une formule de travail dont tout symbole de négation est de la forme $\neg^7$.

2. La seconde étape est un parcours ascendant de structure de formule de travail avec élimination de quantificateur et distribution. Cette étape est effectuée par des règles 25 à 28. Dans chaque sous-formule de travail de profondeur 1 ou 2, la règle 25 élimine les variables quantifiées de la seconde partie de la décomposition (la troisième partie étant éliminée dans la première étape). La règle 26 élimine les contraintes de la seconde partie dans les niveaux les plus profonds. Chaque sous-formule de travail de profondeur 3 est transformée au fur et à mesure en une conjonction de formules de travail de profondeur 2 par la règle 28, en utilisant une propriété du quantificateur $\exists?$. Les transformations de cette étape peuvent créer des nouvelles sous-formules de travail où la première étape nécessite d'être effectuée. A la fin des transformations, on obtient une formule de travail finale de profondeur inférieure ou égale à 3.

4.3 Règles de réécriture

Nous présentons dans la figure 1 les règles de réécriture qui transforment une formule de travail initiale en une formule de travail finale équivalente dans T. L'application d'une règle $p_1 \Longrightarrow p_2$ sur une formule de travail p consiste à remplacer dans p une sous-formule p_1 par la formule p_2, en considérant que le connecteur $\wedge$ est associatif et commutatif.

Dans toutes ces règles, α représente une formule basique, φ et ψ des conjonctions de formules de travail.

Dans les règles 1 à 14, les équations et relations dans α^c et α^p sont mélangées en considérant le connecteur $\wedge$ associatif et commutatif. Dans ces règles, sauf dans la règle 6, toutes les modifications sont effectuées dans α^c, car α^p est un bloc résolu.

Dans la règle 2, f et g sont deux symboles de fonction distincts de F. Dans les règles 4, 6, 7, $x \succ y$. Dans la règle 5, l'équation $x = fz_1...z_n$ n'est pas dans α^p. Dans la règle 6, si l'équation $x = fz_1...z_n$ est dans α^p alors l'équation $x = y \wedge \mathit{arbre}\, y$ obtenue par cette règle est déplacée vers α^p. Dans la règle 7, l'équation $x = z$ n'est pas dans α^p.

Rappelons que la notation 01 dans la règle 8 signifie le terme 0. Dans la règle 9, $a_0 > 0$. Dans les règles 13 et 14 la variable x_k est le représentant de l'équation $\Sigma_i a_i x_i = a_0 1$ et $b_k \neq 0$. De plus l'équation $\Sigma_j b_j x_j = b_0 1$ n'est pas dans α^p. Dans la règle 14, l'inéquation $\Sigma_j b_j x_j < b_0 1$ n'est pas dans α^p et $\lambda = 1$ si $a_k > 0$ et $\lambda = -1$ sinon.

Dans la règle 15, la section arborescente de $\alpha^c \wedge \alpha^p$ est formatée et $\alpha^c \wedge \alpha^p$ n'a pas de sous-formules de la forme $\mathit{num}\, x \wedge \mathit{arbre}\, x$. Dans les règles 16 et 17, la contrainte de typage $\mathit{num}\, z$ (ou $\mathit{arbre}\, z$ pour la règle 17) n'est pas dans $\alpha^c \wedge \alpha^p$ et est une conséquence de $\alpha^c \wedge \alpha^p$. Dans la règle 18, z n'a pas de contrainte de typage dans $\alpha^c \wedge \alpha^p$ et aucune contrainte de typage sur z n'est conséquence de cette formule.

Dans la règle 19, $\alpha^c \wedge \alpha^p$ est un bloc. Dans la règle 20, la section numérique de $\alpha^c \wedge \alpha^p$ est inconsistante. Dans la règle 21, les variables inaccessibles de $\bar{x}$ sont renommées si nécessaire de telle façon que $u \succ v$ pour chaque variable inaccessible u et chaque variable accessible v de $\bar{x}$ et la section numérique de $\alpha^c \wedge \alpha^p$ est consistante. La consistance est vérifiée en utilisant la première phase du Simplex. Dans la règle 22, $\alpha^c \wedge \alpha^p$ est un bloc résolu.

Dans la règle 23, la formule γ^c est obtenue à partir de β^c comme suit. Pour chaque variable $x \in \mathit{var}(\beta^c)$, on ajoute toutes les relations $\mathit{num}\, x$ ou $\mathit{arbre}\, x$ qui sont dans β^p mais pas dans β^c, et pour toute variable y qui n'apparaît pas dans une équation ou une inéquation de β^c on supprime de β^p toutes les relations $\mathit{num}\, y$ ou $\mathit{arbre}\, y$ qui sont à la fois dans β^c et β^p. La formule γ^p est la formule $\alpha^p \wedge \alpha^c$.

Dans la règle 24, la formule $\exists \bar{x} \alpha^c$ est décomposée en $\exists \bar{x}^1 \alpha^{c1} \wedge (\exists \bar{x}^2 \alpha^{c2} \wedge (\exists \bar{x}^3 \alpha^{c3}))$, $\gamma_i^c = \beta_i^c \wedge \alpha^{c3}$ et $\gamma_i^p = \beta_i^p \wedge \alpha^{c1} \wedge \alpha^{c2} \wedge \alpha^p$.

Les quatre règles 25, 26, 27 et 28 ne peuvent pas s'appliquer sur le premier symbole $\neg^7$ de la formule de travail (le symbole du premier niveau). Dans la règle 25, I' est l'ensemble des $i \in I$ tels que β_i^c ne contient pas d'occurrences de variables de $\bar{x}^2$. La formule α^{c2*} est telle que $T \models (\exists \bar{x}^2 \alpha^{c2}) \leftrightarrow \alpha^{c2*}$ et est calculée en utilisant l'élimination de Fourrier. La section de contraintes propagées β_i^{p*} est $\alpha^{c1} \wedge \alpha^{c2*} \wedge \alpha^p$.

Dans la règle 26, φ est tel que $k \geq 6$ pour tout symbole de négation $\neg^k$, φ_0 est obtenu de φ en remplaçant toute occurrence de $\neg^k$ par $\neg^0$ et toute section de contraintes propagées par la formule vrai. Soit β^2 la formule obtenue de β^{c2} en supprimant de β^{c2} les occurrences multiples des contraintes de typage et pour toute variable y n'apparaissant pas dans une inéquation de β^{c2} en supprimant de β^{c2} toutes les relations $\mathit{num}\, y$ ou $\mathit{arbre}\, y$ qui sont à la fois dans β^{c1} et β^{c2}. Si β^2 est la formule vrai alors $I = \emptyset$, sinon les formules β_i^{c2*} avec $i \in I$ sont obtenues de β^2 comme suit : du fait que $\beta^2 \in A^2$, elle est donc de la forme

$$\begin{bmatrix} (\bigwedge_{\ell \in L} \mathit{num}\, z_\ell) \wedge (\bigwedge_{k \in K} \mathit{arbre}\, v_k) \wedge \\ ((\bigwedge_{j \in J} \sum_{i=1}^n a_{ij} x_i < a_{0j}) \wedge \bigwedge_{m=1}^n \mathit{num}\, x_m) \end{bmatrix},$$

$\neg\beta^2$ est donc de la forme

$$\begin{bmatrix} (\bigvee_{\ell \in L} \mathit{arbre}\ z_\ell) \vee (\bigvee_{k \in K} \mathit{num}\, v_k) \vee (\bigvee_{m=1}^n \mathit{arbre}\, x_m) \\ \bigvee_{j \in J} ((\sum_{i=1}^n a_{ij} x_i = a_{0j} 1 \wedge \bigwedge_{m=1}^n \mathit{num}\, x_m) \vee \\ (\sum_{i=1}^n (-a_{ij}) x_i < (-a_{0j}) 1 \wedge \bigwedge_{m=1}^n \mathit{num}\, x_m)) \end{bmatrix}$$

Chaque élément de cette disjonction représente une formule β_i^{c2*}. Bien entendu $T \models (\neg\beta^2) \leftrightarrow \bigvee_i \beta_i^{c2*}$.

FIG. 1 – Les règles de réécriture

$$
\begin{array}{lll}
1 & \neg^1(\exists \bar{u}\, num\, x \wedge arbre\, x \wedge \alpha \wedge \varphi) & \Longrightarrow\ vrai \\
2 & \neg^1(\exists \bar{u}\, x = f\bar{y} \wedge x = g\bar{z} \wedge arbre\, x \wedge \alpha \wedge \varphi) & \Longrightarrow\ vrai \\
3 & \neg^1(\exists \bar{u}\, x = x \wedge \alpha \wedge \varphi) & \Longrightarrow\ \neg^1(\exists \bar{u}\, \alpha \wedge \varphi) \\
4 & \neg^1(\exists \bar{u}\, y = x \wedge arbre\, x \wedge \alpha \wedge \varphi) & \Longrightarrow\ \neg^1(\exists \bar{u}\, x = y \wedge arbre\, x \wedge \alpha \wedge \varphi) \\
5 & \neg^1 \left[\begin{array}{l} \exists \bar{u}\, x = fy_1 ... y_n \wedge x = fz_1 ... z_n \wedge \\ arbre\, x \wedge \alpha \wedge \varphi \end{array} \right] & \Longrightarrow\ \neg^1 \left[\begin{array}{l} \exists \bar{u}\, x = fy_1 ... y_n \wedge \bigwedge_i y_i = z_i \wedge \\ arbre\, x \wedge \alpha \wedge \varphi \end{array} \right] \\
6 & \neg^1 \left[\begin{array}{l} \exists \bar{u}\, x = y \wedge x = fz_1 ... z_n \wedge \\ arbre\, x \wedge arbre\, y \wedge \alpha \wedge \varphi \end{array} \right] & \Longrightarrow\ \neg^1 \left[\begin{array}{l} \exists \bar{u}\, x = y \wedge y = fz_1 ... z_n \wedge \\ arbre\, x \wedge arbre\, y \wedge \alpha \wedge \varphi \end{array} \right] \\
7 & \neg^1(\exists \bar{u}\, x = y \wedge x = z \wedge arbre\, x \wedge \alpha \wedge \varphi) & \Longrightarrow\ \neg^1(\exists \bar{u}\, x = y \wedge y = z \wedge arbre\, x \wedge \alpha \wedge \varphi) \\
8 & \neg^4(\exists \bar{u}\, 0 = 01 \wedge \alpha \wedge \varphi) & \Longrightarrow\ \neg^4(\exists \bar{u}\, \alpha \wedge \varphi) \\
9 & \neg^4(\exists \bar{u}\, 0 < a_0 1 \wedge \alpha \wedge \varphi) & \Longrightarrow\ \neg^4(\exists \bar{u}\, \alpha \wedge \varphi) \\
10 & \neg^4 \left[\begin{array}{l} \exists \bar{u}\, x = y \wedge \\ num\, x \wedge num\, y \wedge \alpha \wedge \varphi \end{array} \right] & \Longrightarrow\ \neg^4 \left[\begin{array}{l} \exists \bar{u}\, x + (-1y) = 0 \wedge \\ num\, x \wedge num\, y \wedge \alpha \wedge \varphi \end{array} \right] \\
11 & \neg^4 \left[\begin{array}{l} \exists \bar{u}\, x = -y \wedge \\ num\, x \wedge num\, y \wedge \alpha \wedge \varphi \end{array} \right] & \Longrightarrow\ \neg^4 \left[\begin{array}{l} \exists \bar{u}\, x + y = 0 \wedge \\ num\, x \wedge num\, y \wedge \alpha \wedge \varphi \end{array} \right] \\
12 & \neg^4 \left[\begin{array}{l} \exists \bar{u}\, x = y + z \wedge num\, x \wedge \\ num\, y \wedge num\, z \wedge \alpha \wedge \varphi \end{array} \right] & \Longrightarrow\ \neg^4 \left[\begin{array}{l} \exists \bar{u}\, x + (-1y) + (-1z) = 0 \wedge \\ num\, x \wedge num\, y \wedge num\, z \wedge \alpha \wedge \varphi \end{array} \right] \\
13 & \neg^4 \left[\begin{array}{l} \exists \bar{u}\, \Sigma_{i=1}^n a_i x_i = a_0 1 \wedge \\ \Sigma_{j=1}^n b_j x_j = b_0 1 \wedge \\ \bigwedge num\, x \wedge \alpha \wedge \varphi \end{array} \right] & \Longrightarrow\ \neg^4 \left[\begin{array}{l} \exists \bar{u}\, \Sigma_{i=1}^n a_i x_i = a_0 1 \wedge \\ \Sigma_{j=1}^n (b_k a_i - a_k b_i) x_i = (b_k a_i - a_k b_0) 1 \wedge \\ \bigwedge num\, x \wedge \alpha \wedge \varphi \end{array} \right] \\
14 & \neg^4 \left[\begin{array}{l} \exists \bar{u}\, \Sigma_{i=1}^n a_i x_i = a_0 1 \wedge \\ \Sigma_{j=1}^n b_j x_j < b_0 1 \wedge \\ \bigwedge num\, x \wedge \alpha \wedge \varphi \end{array} \right] & \Longrightarrow\ \neg^4 \left[\begin{array}{l} \exists \bar{u}\, \Sigma_{i=1}^n a_i x_i = a_0 1 \wedge \\ \Sigma_{j=1}^n \lambda (b_k a_i - a_k b_i) x_i < (b_k a_i - a_k b_0) 1 \wedge \\ \bigwedge num\, x \wedge \alpha \wedge \varphi \end{array} \right] \\
15 & \neg^1(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ \neg^2(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) \\
16 & \neg^2(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ \neg^1(\exists \bar{x}\, num\, z \wedge \alpha^c \wedge \alpha^p \wedge \varphi) \\
17 & \neg^2(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ \neg^1(\exists \bar{x}\, arbre\, z \wedge \alpha^c \wedge \alpha^p \wedge \varphi) \\
18 & \neg^2(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ \left[\begin{array}{l} \neg^1(\exists \bar{x}\, num\, z \wedge \alpha^c \wedge \alpha^p \wedge \varphi) \wedge \\ \neg^1(\exists \bar{x}\, arbre\, z \wedge \alpha^c \wedge \alpha^p \wedge \varphi) \end{array} \right] \\
19 & \neg^2(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ \neg^3(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) \\
20 & \neg^3(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ vrai \\
21 & \neg^3(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ \neg^4(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) \\
22 & \neg^4(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) & \Longrightarrow\ \neg^5(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi) \\
23 & \neg^7 \left[\begin{array}{l} \exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi \wedge \\ \neg^5(\exists \bar{y}\, \beta^c \wedge \beta^p \wedge \psi) \end{array} \right] & \Longrightarrow\ \neg^7 \left[\begin{array}{l} \exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi \wedge \\ \neg^6(\exists \bar{y}\, \gamma^c \wedge \gamma^p \wedge \psi) \end{array} \right] \\
24 & \neg^6 \left[\begin{array}{l} \exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \\ \bigwedge_i \neg^0(\exists \bar{y}_i \beta_i^c \wedge \beta_i^p \wedge \varphi_i) \end{array} \right] & \Longrightarrow\ \neg^7 \left[\begin{array}{l} \exists \bar{x}^1 \bar{x}^2\, \alpha^{c1} \wedge \alpha^{c2} \wedge \alpha^p \wedge \\ \bigwedge_i \neg^1(\exists \bar{y}_i \bar{x}^3 \gamma_i^c \wedge \gamma_i^p \wedge \varphi_i) \end{array} \right] \\
25 & \neg^7 \left[\begin{array}{l} \exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \\ \bigwedge_{i \in I} \neg^9(\exists \bar{y}_i \beta_i^c \wedge \beta_i^p) \end{array} \right] & \Longrightarrow\ \neg^8 \left[\begin{array}{l} \exists \bar{x}^1 \alpha^{c1} \wedge \alpha^{c2*} \wedge \alpha^p \wedge \\ \bigwedge_{i \in I'} \neg^9(\exists \bar{y}_i \beta_i^c \wedge \beta_i^{p*}) \end{array} \right] \\
26 & \neg^7 \left[\begin{array}{l} \exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi \wedge \\ \neg^8(\exists \bar{y} \beta^c \wedge \beta^p) \end{array} \right] & \Longrightarrow\ \left[\begin{array}{l} \neg^7(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi \wedge \neg^9(\exists \bar{y} \beta^{c1} \wedge \beta^p)) \wedge \\ \bigwedge_{i \in I} \neg^1(\exists \bar{x} \bar{y} \beta^p \wedge \beta^{c1} \wedge \beta_i^{c2*} \wedge \varphi_0) \end{array} \right] \\
27 & \neg^7 \left[\begin{array}{l} \exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi \wedge \\ \neg^9(\exists \varepsilon\, vrai \wedge \beta^p) \end{array} \right] & \Longrightarrow\ vrai \\
28 & \neg^7 \left[\begin{array}{l} \exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi \wedge \\ \neg^8 \left[\begin{array}{l} \exists \bar{y}\, \beta^c \wedge \beta^p \wedge \\ \bigwedge_{i \in I} \neg^9(\exists \bar{z}_i\, \gamma_i^c \wedge \gamma_i^p) \end{array} \right] \end{array} \right] & \Longrightarrow\ \left[\begin{array}{l} \neg^7(\exists \bar{x}\, \alpha^c \wedge \alpha^p \wedge \varphi \wedge \neg^8(\exists \bar{y}\, \beta^c \wedge \beta^p)) \wedge \\ \bigwedge_{i \in I} \neg^6(\exists \bar{x} \bar{y} \bar{z}_i\, \delta_i^c \wedge \delta_i^p \wedge \varphi_0) \end{array} \right]
\end{array}
$$

Dans la règle 28, $I \neq \emptyset$, φ est telle que $k \geq 6$ pour tout symbole de négation $\neg^k$, φ_0 est obtenue de φ en remplaçant toute occurrence de $\neg^k$ par $\neg^0$ et toute section de contraintes propagées par *vrai*. La formule δ_i^p est α^p et la formule δ_i^c est $\gamma_i^c \wedge \beta^c \wedge \alpha^c$.

Propriété 4.3.1 *Toute application répétée des règles de réécriture précédentes sur une formule de travail initiale se termine et produit une formule de travail finale qui est équivalente dans T et qui ne contient pas d'autres variables libres que celles de la formule initiale.*

4.4 Algorithme de résolution

La résolution d'une contrainte φ dans T se fait de la façon suivante :

1. Transformer φ en une formule normalisée équivalente, puis en une formule de travail initiale ϕ équivalente dans T.
2. Transformer ϕ en une formule de travail finale ψ en utilisant les règles de réécriture définies dans la sous-section 4.3.
3. Extraire de ψ la disjonction de formules résolues générales équivalente à ψ dans T en utilisant la propriété 4.1.4. Si la disjonction contient la formule résolue générale *vrai*, elle se réduit alors à *vrai*.

Exemple 4.4.1 *Etudions dans cet exemple la résolution de la contrainte suivante ayant les variables i, j comme variables libres :*

$$\exists x\, x = fij \wedge i > 0 \wedge arbre\, x \wedge num\, i \wedge num\, j \wedge \neg(\exists k\, j = 2k \wedge num\, k) \quad (7)$$

En examinant cette contrainte dans la théorie T, on peut remarquer que $num\, j \wedge \neg(\exists k\, j = 2k \wedge num\, k)$ est toujours faux car d'après l'axiome 15_n avec $n = 2$, pour chaque j tel que $num\, j$, il existe toujours un unique k tel que $num\, k$ et $j = 2k$. Donc bien que la formule (7) contienne deux variables libres, elle est toujours équivalente à faux dans T. Utilisons maintenant notre algorithme pour résoudre cette contrainte. La contrainte est d'abord transformée en la formule initiale suivante (les sections de contraintes propagées sont soulignées) :

$$\neg^6\neg^0 \left[\begin{array}{l} \exists x\, x = fij \wedge i > 0 \wedge arbre\, x \wedge num\, j \wedge \underline{vrai} \wedge \\ \neg^0(\exists k\, j = 2k \wedge num\, k \wedge \underline{vrai}) \end{array} \right]$$

Après avoir appliqué les règles 24, 15, 16, 15, 19, 21, 22, 23 dans cet ordre, on obtient :

$$\neg^7\neg^6 \left[\begin{array}{l} \exists x\, x = fij \wedge i > 0 \wedge arbre\, x \wedge num\, i \wedge num\, j \wedge \underline{vrai} \wedge \\ \neg^0(\exists k\, j = 2k \wedge num\, k \wedge \underline{vrai}) \end{array} \right]$$

Seule la règle 24 peut s'appliquer, donnant :

$$\neg^7\neg^7 \left[\begin{array}{l} i > 0 \wedge num\, i \wedge num\, j \wedge \underline{vrai} \wedge \\ \neg^1 \left[\begin{array}{l} \exists xk\, x = fij \wedge j = 2k \wedge num\, k \wedge arbre\, x \wedge \\ \underline{i > 0 \wedge num\, i \wedge num\, j} \end{array} \right] \end{array} \right]$$

Les règles 15, 19, 21, 12, 22, 23 peuvent s'appliquer dans l'ordre sur la sous-formule de travail $\neg^1(...)$, l'application donne

$$\neg^7\neg^7 \left[\begin{array}{l} i > 0 \wedge num\, i \wedge num\, j \wedge \underline{vrai} \wedge \\ \neg^6 \left[\begin{array}{l} \exists xk\, x = fij \wedge j - 2k = 0 \wedge num\, k \wedge arbre\, x \wedge \\ \underline{i > 0 \wedge num\, i \wedge num\, j} \end{array} \right] \end{array} \right]$$

Seule la règle 24 peut s'appliquer et on obtient :

$$\neg^7\neg^7 \left[\begin{array}{l} i > 0 \wedge num\, i \wedge num\, j \wedge \underline{vrai} \wedge \\ \neg^7(vrai \wedge \underline{i > 0 \wedge num\, i \wedge num\, j}) \end{array} \right]$$

Les règles 25, 26 s'appliquant dans l'ordre, donnent :

$$\neg^7\neg^7 \left[\begin{array}{l} i > 0 \wedge num\, i \wedge num\, j \wedge \underline{vrai} \wedge \\ \neg^9(vrai \wedge \underline{i > 0 \wedge num\, i \wedge num\, j}) \end{array} \right]$$

Finalement, seule la règle 27 peut s'appliquer et on obtient la formule de travail finale $\neg^7 vrai$, qui d'après la propriété 4.1.4 (avec $I = \emptyset$) est équivalente à la disjonction vide de formules résolues générales, c'est-à-dire à la formule faux.

Du fait que T a au moins un modèle [12], T est consistante. D'après les propriétés 4.3.1, 4.1.4 et 4.1.5, on obtient le corollaire suivant, qui est une autre preuve de la complétude de la théorie T :

Corollaire 4.4.2 *Chaque formule est équivalente dans T soit à vrai, soit à faux, soit à une disjonction de formules résolues générales, qui a au moins une variable libre et qui n'est équivalente ni à vrai ni à faux dans T.*

5 Conclusion

Nous avons présenté dans ce papier un algorithme de résolution de contraintes du premier ordre dans la théorie T des arbres évalués. L'algorithme est présenté sous forme de 28 règles de réécriture et sa correction est une autre preuve de la complétude de T. L'algorithme transforme une formule φ contenant éventuellement des variables libres en une disjonction de formules résolues ϕ équivalente dans T, ne contenant pas d'autres variables libres que celles de φ, et telle que ϕ est soit la formule *vrai*, soit la formule *faux*, soit une formule contenant au moins une variable libre et n'étant équivalente ni à *vrai* ni à *faux* dans T. Notre objectif dans ce travail n'était pas seulement de décider la vérité des propositions, mais de simplifier des contraintes générales avec des variables libres et de présenter les solutions d'une façon explicite. Cet algorithme est capable de détecter des formules ayant des variables libres mais étant toujours vraies ou toujours fausses dans T, en donnant comme résultat *vrai* ou *faux*.

S. Vorobyov [22] a montré que le problème de décision de propositions dans la théorie des arbres est non-élémentaire, c'est-à-dire que la complexité de tout algorithme de décision ne peut pas être bornée par une tour d'exponentielles de 2 (avec une évaluation de haut en bas) d'une hauteur fixée. A. Colmerauer et B. Dao [6] ont aussi donné une preuve de complexité non-élémentaire de résolution de contraintes dans cette théorie. Notre algorithme ne doit pas donc échapper à cette complexité dans le pire des cas. Par conséquent

nous utilisons deux stratégies dans cet algorithme : une résolution locale et propagation de contraintes en descendant la structure de formule, et une élimination de quantificateurs et distribution en remontant la structure de formule. La résolution locale permet de détecter plus rapidement les sous-formules équivalentes à faux. La propagation de contraintes quant à elle nous évite de résoudre des sous-formules qui contredisent leur sur ou sous formules en réduisant le tout à *faux*. Nous avons implanté un algorithme similaire dans le cas de la théorie des arbres finis ou infinis et résolu des formules de positions gagnantes de jeux à deux joueurs, contenant jusqu'à 160 quantificateurs imbriqués [11].

Relativement à ce travail, nous essayons de formaliser une axiomatisation automatique de la combinaison d'une théorie T avec la théorie des arbres finis ou infinis. Nous travaillons également sur un algorithme général de résolution de contraintes du premier ordre dans la nouvelle théorie hybride. Nous étudions aussi l'expressivité de contraintes du premier ordre dans ces théories pour trouver des classes de problèmes avec des meilleures complexités de résolution.

Remerciement Nous remercions Alain Colmerauer pour nos nombreuses discussions. Nous lui dédions ce papier en lui souhaitant un rétablissement rapide.

Références

[1] Benhamou F, Bouvier P, Colmerauer A, Garetta H, Giletta B, Massat J, Narboni G, N'dong S, Pasero R, Pique J, Touraivane, Van caneghem M, Vetillard E. Le manuel de Prolog IV , PrologIA, Marseille, France, 1996.

[2] Bürckert H. Solving disequations in equational theories. In Proc. 9th Conf. on Automated Deduction, LNCS 310, pages 517-526. 1988.

[3] Colmerauer A. An introduction to Prolog III. *Communication of the ACM*, 33(7) :68–90,1990.

[4] Colmerauer A. Equations and disequations on finite and infinite trees. Proceeding of the International conference on the fifth generation of computer systems Tokyo, 1984. P. 85–99.

[5] Colmerauer A. Prolog and infinite trees. In K.L. Clark and S-A. Tarnlund, editors, Logic Programming, pages 231-251, 1982. Academic Press.

[6] Colmerauer A., Dao. TBH., Expressiveness of full first-order constraints in the algebra of finite or infinite trees, Constraints, Vol. 8, No. 3, 2003, pages 283-302.

[7] Comon H. Unification et disunification : Théorie et applications. PhD thesis, Institut National Polytechnique de Grenoble, 1988.

[8] Comon H. Résolution de contraintes dans des algèbres de termes. Rapport d'Habilitation, Université de Paris Sud, 1992.

[9] Courcelle B. Equivalences and Transformations of Regular Systems applications to Program Schemes and Grammars, TCS vol. 42, 1986, p. 1–122.

[10] Courcelle B. Fundamental Properties of Infinite Trees, TCS vol. 25, no 2, 1983, pp. 95–169.

[11] Dao T. Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis. Thèse d'informatique, Université de la Méditerranée, 2000.

[12] Djelloul K. About the combination of trees and rational numbers in a complete first-order theory. 5th Int. Conf. FroCoS 2005, LNAI, vol 3717, P. 106–122.

[13] Huet G. Résolution d'équations dans les langages d'ordre 1, 2,...ω. Thèse d'Etat, Université Paris 7. France,1976.

[14] Jaffar J. Efficient unification over infinite terms. New Generation Computing, 2(3) :207-219, 1984.

[15] Kunen K. Negation in logic programming. Journal of Logic Programming, 4 :289ñ308, 1987.

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

[17] Malcev A. Axiomatizable classes of locally free algebras of various types. In B.Wells III, editor, The Metamathematics of Algebraic Systems. Anatolii Ivanovic Malcev. Collected Papers : 1936-1967, volume 66, chapter 23, pages 262-281. 1971.

[18] Matelli A. and Montanari U. An efficient unification algorithm. ACM Trans. on Languages and Systems, 4(2) :258-282, 1982.

[19] Paterson M and Wegman N. Linear unification. Journal of Computer and Systems Science, 16 :158-167, 1978.

[20] Ramachandran V. and Van Hentenryck P. Incremental algorithms for constraint solving and entailment over rational trees. In Proc. of the 13th Conf. Foundations of Software Technology and Theoretical Computer Science, LNCS 761, 205-217, 1993.

[21] Robinson J.A. A machine-oriented logic based on the resolution principle. JACM, 12(1) :23-41, 1965.

[22] Vorobyov S. An Improved Lower Bound for the Elementary Theories of Trees, Proceeding of the 13th International Conference on Automated Deduction (CADE'96). LNAI 1104, pp. 275- 287, 1996.