Thi-Bich-Hanh Dao

Contraintes du premier ordre dans l'algèbre des arbres finis ou infinis

Volume 5, nº 2-3 (2024), p. 121-138.

https://doi.org/10.5802/roia.75

© Les auteurs, 2024.

Cet article est diffusé sous la licence
Creative Commons Attribution 4.0 International License.
http://creativecommons.org/licenses/by/4.0/

La Revue Ouverte d'Intelligence Artificielle est membre du Centre Mersenne pour l'édition scientifique ouverte
www.centre-mersenne.org
e-ISSN : 2967-9672

Revue Ouverte d'Intelligence Artificielle
Volume 5, nº 2-3, 2024, 121-138

Contraintes du premier ordre dans l'algèbre des arbres finis ou infinis

Thi-Bich-Hanh Dao[a]

[a] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067, Orléans (France)
E-mail : thi-bich-hanh.dao@univ-orleans.fr.

Résumé. — L'algèbre des arbres finis ou infinis joue un rôle fondamental en informatique. Entre autres, le fonctionnement de Prolog II, III et IV a été modélisé par Alain Colmerauer comme la résolution de contraintes sous la forme d'équations et de diséquations dans cette algèbre. Dans un travail publié à la revue Constraints avec Alain Colmerauer [6], nous avons étudié l'expressivité des contraintes plus générales dans cette algèbre, qui sont des formules logiques du premier ordre, construites à partir des variables, désignant des arbres, d'opérations de construction, de la relation de l'égalité, des connecteurs logiques et des quantificateurs. Nous avons présenté la construction d'exemples pour montrer un pouvoir d'expressivité immensément grand de contraintes aussi générales. Cet article présente une version réduite de ce travail sur le pouvoir d'expressivité de ces contraintes [6]. De plus, nous décrivons la méthode que nous avons développée pour résoudre ces contraintes.

Mots-clés. — Algèbre des arbres, contraintes, expressivité de contraintes, résolution de contraintes.

1. Introduction

L'algèbre des arbres finis ou infinis joue un rôle fondamental en informatique. Elle modélise aussi bien des structures de données que des programmes ou des exécutions de programme. L'unification de termes finis, c'est-à-dire la résolution de conjonction d'équations portant sur des arbres finis, a d'abord été étudiée par J.A. Robinson [22]. L'unification de termes infinis, donc la résolution de conjonction d'équations dans les arbres infinis, a été étudiée par G. Huet [14], A. Colmerauer [3, 7] et J. Jaffar [15]. Pour une synthèse sur ce sujet, nous renvoyons à l'article de J.-P. Jouannaud et C. Kirchner [16].

Le fonctionnement de Prolog II, III et IV a été modélisé par A. Colmerauer comme la résolution d'équations (relation =) et de diséquations (relation ≠) dans les arbres (éventuellement) infinis [1, 4, 5, 7]. L'exécution de programmes Prolog est donc devenue la résolution de contraintes dans l'algèbre des arbres, contraintes qui sont sous la forme de combinaison par la disjonction et la conjonction d'équations et de diséquations. La résolution de ce type de contraintes a été ensuite étudiée dans d'autres travaux [2, 21, 23].

Il a été alors naturel de nous intéresser à des contraintes plus générales dans cette algèbre, qui sont des formules logiques du premier ordre, construites à partir des variables, désignant des arbres, des opérations de construction, de la relation d'égalité, des connecteurs logiques et des quantificateurs. Des algorithmes d'élimination de quantificateurs pour transformer des formules logiques du premier ordre en des combinaisons booléennes de formules plus simples ont été proposés. Pour le cas des arbres finis, nous pouvons citer les travaux de A. Malcev [19], de K. Kunen [17], de M. Maher [18] ou de H. Comon [9, 10]. Pour une synthèse sur ce sujet nous renvoyons à l'article de H. Comon [8]. Une théorie complète de l'algèbre des arbres finis ou infinis a été proposée par M. Maher [18]. Il a, entre autres, montré que dans cette théorie, et donc dans l'algèbre des arbres, toute formule du premier ordre était équivalente à une combinaison booléenne de conjonctions d'équations où toutes les variables ou une partie des variables sont quantifiées existentiellement.

Dans cet article, nous considérons des contraintes, qui sont des formules logiques du premier ordre avec des variables libres, dans l'algèbre des arbres. Dans un travail publié à la revue Constraints [6] avec A. Colmerauer, nous avons étudié l'expressivité de ces contraintes, où nous avons présenté des exemples afin de montrer un pouvoir d'expressivité quasi universel de contraintes aussi générales. Nous présentons dans cet article des résultats principaux de ce travail [6]. De plus, nous décrivons la méthode que nous avons développée pour trouver les solutions de ces contraintes.

L'article est organisé de la manière suivante. La section 2 présente les notions d'algèbre des arbres finis ou infinis et de contraintes du premier ordre dans cette algèbre. La section 3 considère un jeu à deux joueurs et présente une modélisation des positions k-gagnantes par des contraintes d'arbres avec des quantificateurs imbriqués. Cette modélisation permet de définir une position gagnante du jeu en jouant au plus k coups par une formule dont la taille est linéaire en k. Dans le but d'illustrer des modélisations plus compactes, la section 4 présente la construction d'une contrainte de taille linéaire en k et qui permet d'itérer $\alpha(k)$ fois une fonction de transition φ, où $\alpha(k)$ est une tour d'exponentielle de hauteur k. La section 5 présente deux applications de cette contrainte. La première illustre une contrainte de taille linéaire en k dont l'unique solution est un arbre fini de taille $\alpha(k)$. La seconde application considère une machine de Turing dont chaque configuration est représentée par un arbre infini et dont les instructions sont exprimées par la fonction de transition φ, pour montrer que tout problème de décision pouvant être résolu par une machine de Turing en exécutant au plus $\alpha(k)$ transitions peut être encodé par une contrainte d'arbres de taille linéaire en k, établissant ainsi un pouvoir d'expressivité « quasi-universel » des contraintes d'arbres. La section 6 décrit une méthode de résolution de ces contraintes et la section 7 conclut.

2. Préliminaires

2.1. L'algèbre des arbres finis ou infinis

Comme habituellement, un *symbole de fonction* est un symbole auquel est associé un entier non négatif, son *arité*. Les arbres, avec des nœuds étiquetés par des symboles

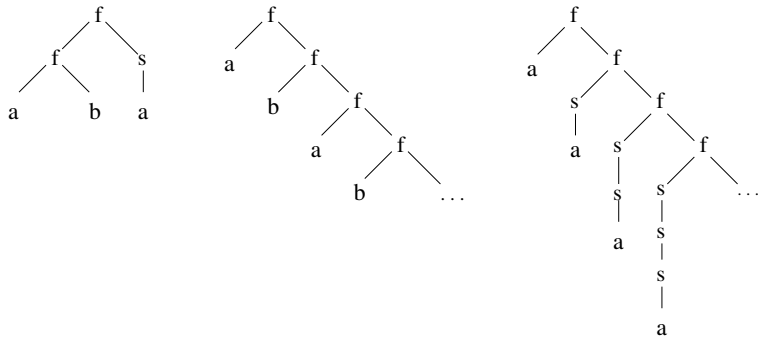

FIGURE 2.1. Exemples d'arbres, avec les symboles de fonction a, b, s, f d'arités respectives 0, 0, 1 et 2.

de fonction, sont bien connus en informatique. La figure 2.1 présente quelques exemples d'arbres. On remarque que seul le premier arbre a un ensemble fini de nœuds mais que le deuxième a quand même un ensemble fini (de formes) de sous-arbres. Le premier arbre est un arbre fini, les deux autres sont des arbres infinis. Le deuxième arbre est un arbre infini rationnel car le nombre de formes de sous-arbres est fini. Le troisième arbre est infini non rationnel.

Nous nous donnons un ensemble F infini de symboles de fonction et désignerons par $\mathbf{A}$ l'ensemble de tous les arbres construits sur l'ensemble infini F. L'ensemble $\mathbf{A}$ est muni d'un ensemble d'opérations de construction, une pour chaque élément $f \in F$. Elle consiste en l'application $(a_1, \ldots, a_n) \mapsto a$, où n est l'arité de f et a l'arbre dont le nœud initial est étiqueté f et dont la suite de fils est $a_1, \ldots, a_n$ (voir la figure 2.2). On obtient ainsi l'*algèbre des arbres finis ou infinis* construits sur F, que l'on note $(\mathbf{A}, F)$.

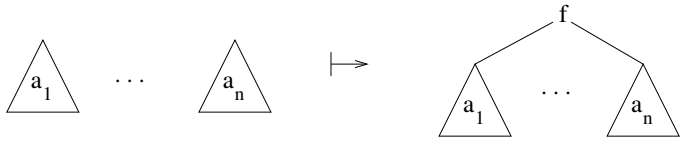

FIGURE 2.2. L'opération de construction associée à chaque symbole de fonction $f \in F$.

2.2. Les contraintes d'arbres

Nous nous intéressons aux contraintes représentées par des formules du premier ordre avec pour seul symbole de relation l'égalité et pour symboles de fonction les éléments d'un ensemble infini F. Ces *contraintes d'arbres* seront de l'une des dix formes :

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

où φ et ψ sont des contraintes d'arbres plus courtes, x une variable prise dans un ensemble infini et s, t des termes, c'est-à-dire des expressions de l'une des formes :

$$x, \quad f(t_1, \ldots, t_n),$$

avec $n \geqslant 0$, $f \in F$ et d'arité n et les t_i des termes plus courts.

Les variables représenteront des éléments de l'ensemble $\mathbf{A}$ des arbres construits sur F, le symbole d'égalité sera interprété par l'égalité et les symboles de fonction f seront interprétés comme des opérations de constructions dans l'algèbre des arbres infinis $(\mathbf{A}, F)$. Un terme x sera donc interprété par l'arbre que représente x et un terme $f(t_1, \ldots, t_n)$ interprété par l'arbre résultat de la construction associée à f sur les arbres interprétant $t_1, \ldots, t_n$.

Une contrainte sans variable libre sera donc vraie ou fausse et une contrainte $\varphi(x_1, \ldots, x_n)$ faisant intervenir n variables libres $x_1, \ldots, x_n$ établira une relation n-aire dans l'ensemble des arbres. Par exemple la contrainte d'arbres

$$\varphi(u, v, w, x, y) \stackrel{\text{def}}{=} \exists z \begin{bmatrix} z = f(x, f(y, z)) \ \wedge \\ z = f(u, f(v, f(w, z))) \end{bmatrix}$$

est équivalente à

$$x = u \ \wedge \ y = v \ \wedge \ x = w \ \wedge \ y = u \ \wedge \ x = v \ \wedge \ y = w.$$

Elle exprime donc que les arbres u, v, w, x, y sont tous égaux.

3. Positions gagnantes dans un jeu à deux adversaires

Nous considérons maintenant un jeu à deux adversaires et nous intéressons à caractériser les positions à partir desquelles il est toujours possible de gagner en jouant au plus k coup. On se donne un graphe orienté (V, E) constitué d'un ensemble V de sommets et d'un ensemble $E \subseteq V \times V$ d'arêtes. On permet que V et E soient infinis et on appelle aussi *positions* les éléments de V. On considère le jeu à deux adversaires qui, étant donnée une position initiale x_0, consiste à tour de rôle à choisir une position x_1 tel que $(x_0, x_1) \in E$, puis une position x_2 tel que $(x_1, x_2) \in E$, puis une position x_3 tel que $(x_2, x_3) \in E$ et ainsi de suite. Le premier qui ne peut plus jouer a perdu et l'autre a gagné. Par exemple, on se donne un couple (i, j) d'entiers positifs ou nuls et à tour de rôle on choisit l'un des deux entiers i, j. Suivant que l'entier choisi u est impair ou pair on augmente ou on diminue l'autre entier v de 1 sans jamais le rendre strictement négatif. La première personne qui ne peut plus jouer a perdu. Ce jeu peut être représenté par le graphe orienté dans la figure 3.1, où les arêtes représentent les coups possibles.

3.1. Positions k-gagnantes

Soit $x \in V$ un sommet quelconque du graphe orienté (V, E) et supposons que ce soit au tour de la personne A de jouer. La position x est dite *k-gagnante* si, quelle que soit la façon de jouer de l'autre personne B, il est toujours possible que A gagne en jouant au plus k coups. Elle est dite *k-perdante* si, quelle que soit la façon de jouer de A, il existe toujours une façon de jouer pour B qui a pour effet que A perde et joue au plus k coups.

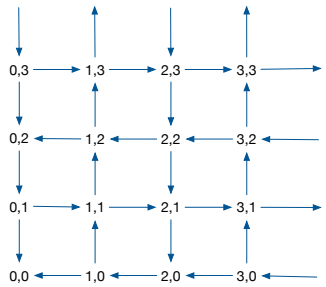

FIGURE 3.1. Un exemple de jeu à deux joueurs.

Soit $coup\,(a, b)$ une contrainte représentant un mouvement possible de $a \in V$ vers $b \in V$. La contrainte $gagnant_k(x)$, avec $k \geqslant 0$, exprimant x une position k-gagnante est alors définie par :

$$\begin{array}{lcl} gagnant_0(x) & \leftrightarrow & faux, \\ gagnant_{k+1}(x) & \leftrightarrow & \exists y\ coup\,(x, y) \wedge perdant_k(y), \end{array} \tag{3.1}$$

avec la contrainte $perdant_k(x)$ définie par :

$$perdant_k(x) \quad \leftrightarrow \quad \forall y\ coup\,(x, y) \rightarrow gagnant_k(y). \tag{3.2}$$

En se restreignant aux quantifications existentielles et en déroulant les équivalences, on constate alors qu'on peut remplacer (3.1) par

$$gagnant_k(x) \ \leftrightarrow \ \boxed{\begin{array}{l} \exists y\ coup\,(x, y) \wedge \neg(\\ \exists x\ coup\,(y, x) \wedge \neg(\\ \exists y\ coup\,(x, y) \wedge \neg(\\ \exists x\ coup\,(y, x) \wedge \neg(\\ \cdots \\ \exists y\ coup\,(x, y) \wedge \neg(\\ \exists x\ coup\,(y, x) \wedge \neg(\\ faux \qquad\qquad \underbrace{)\ldots)}_{2k} \end{array}} \tag{3.3}$$

Les équivalences (3.3) et (3.2) donnent alors un moyen explicite pour construire les contraintes désirées. On remarque qu'en descendant les négations dans (3.3), on obtient une imbrication de $2k$ quantificateurs alternés.

3.2. CODAGE PAR LES ARBRES

Prenons comme structure l'algèbre $(\mathbf{A}, F)$ des arbres construits sur un ensemble F de symboles fonctionnels comprenant notamment les symboles $0, f, g, c$, d'arités respectives 0, 1, 1 et 2. Codons les sommets (i, j) du graphe du jeu par les arbres $c(\bar{i}, \bar{j})$

avec $\bar{i} = (fg)^{\frac{i}{2}}(0)$, si i est pair, et $\bar{i} = g(\overline{i-1})$, si i est impair [(1)]. Par exemple l'arbre dans la figure 3.2 représente le couple $(2, 3)$.

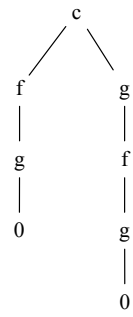

FIGURE 3.2. Arbre représentant le couple $(2, 3)$.

Le lecteur perspicace se convaincra que pour la contrainte $coup\,(x, y)$ généralisée on peut prendre :

$$coup\,(x, y) \stackrel{\text{def}}{=} transition(x, y) \vee (\neg(\exists u\, \exists v\, x = c(u, v)) \wedge x = y)$$

avec

$$transition(x, y) \stackrel{\text{def}}{=} \begin{bmatrix} \exists u\, \exists v\, \exists w \\ \begin{bmatrix} (x = c(u, v) \wedge y = c(u, w)) \vee \\ (x = c(v, u) \wedge y = c(w, u)) \end{bmatrix} \\ \wedge \\ \begin{bmatrix} (\exists i\, u = g(i) \wedge succ\,(v, w)) \vee \\ (\neg(\exists i\, u = g(i)) \wedge pred\,(v, w)) \end{bmatrix} \end{bmatrix}$$

$$succ\,(v, w) \stackrel{\text{def}}{=} \begin{bmatrix} ((\exists j\, v = g(j)) \wedge w = f(v)) \vee \\ (\neg(\exists j\, v = g(j)) \wedge w = g(v)) \end{bmatrix}$$

$$pred\,(v, w) \stackrel{\text{def}}{=} \begin{bmatrix} (\exists j\, v = f(j) \wedge \begin{bmatrix} (\exists k\, j = g(k) \wedge w = j) \vee \\ (\neg(\exists k\, j = g(k)) \wedge w = v) \end{bmatrix}) \vee \\ (\exists j\, v = g(j) \wedge \begin{bmatrix} (\exists k\, j = g(k) \wedge w = v) \vee \\ (\neg(\exists k\, j = g(k)) \wedge w = j) \end{bmatrix}) \vee \\ (\neg(\exists j\, v = f(j)) \wedge \neg(\exists j\, v = g(j)) \wedge \\ \neg(v = 0) \wedge w = v) \end{bmatrix}$$

Remarquons que la contrainte $coup\,(x, y)$ est définie sur les variables x et y représentant des arbres quelconques. Donc si l'arbre x encode bien une position du jeu, la

(1) Bien entendu, $(fg)^0(x) = x$ et $(fg)^{i+1}(x) = f(g((fg)^i(x)))$.

première partie de cette contrainte (i.e. $transition(x, y)$) représente les mouvements possibles à partir de cette position. Dans le cas contraire, si l'arbre x n'est pas de la forme $c(u, v)$, la deuxième partie de la contrainte $coup\,(x, y)$ indique que la position suivante sera identique à la position actuelle ($x = y$). Aussi, si le sous-arbre choisi de x n'a pas de racine étiquetée par f ou g (contrainte $pred\,(v, w)$), la position suivante sera identique à la position actuelle ($w = v$).

L'ensemble des positions k-gagnantes du jeu est l'ensemble des solutions en x de la contrainte $gagnant_k(x)$ définie en (3.3). Par exemple, pour $k = 1$ la contrainte $gagnant_k(x)$ est équivalente à

$$x=c(g(0),0) \vee x=c(0,g(0))) \tag{3.4}$$

et, pour $k = 2$, à

$$\begin{bmatrix} x=c(0,g(0)) \vee x=c(g(0),0) \vee x=c(0,g(f(g(0)))) \vee \\ x=c(g(0),f(g(0))) \vee x=c(f(g(0)),g(0)) \vee x=c(g(f(g(0))),0) \end{bmatrix} \tag{3.5}$$

4. Itération de contraintes

Dans cette section, nous allons présenter la construction d'une contrainte de taille linéaire en k qui permet d'itérer $\alpha(k)$ fois une fonction de transition, où $\alpha(k)$ est une tour d'exponentielle de hauteur k. Introduisons la fonction $\alpha(k)$ définie pour tout entier $k \geqslant 0$ par :

$$\alpha(0) = 1, \qquad \alpha(k+1) = 2^{\alpha(k)}.$$

On a donc :

$$\alpha(n) = \underbrace{2^{\cdots^{\left(2^{(2^2)}\right)}}}_{n}.$$

La fonction α croit d'une façon stupéfiante, puisque $\alpha(0) = 1$, $\alpha(1) = 2$, $\alpha(2) = 4$, $\alpha(3) = 16$, $\alpha(4) = 65536$ et $\alpha(5) = 2^{65536}$. L'entier $\alpha(5)$ est donc supérieur à 10^{20000}, un nombre probablement bien supérieur au nombre d'atomes constituant l'univers et au nombre de nanosecondes qui se sont écoulées depuis sa création !

On convient que la taille $|\varphi|$ d'une contrainte φ est le nombre d'occurrences de tous les symboles, à l'exception des parenthèses et des virgules (car les contraintes peuvent être écrites en notation infixée). Si x, y sont des variables et $\varphi(x, y)$ une contrainte alors, pour tout $n \geqslant 0$, la *n-itérée* de $\varphi(x, y)$ est la contrainte notée et définie par [(2)] :

$$\varphi^n(x,y) \stackrel{\text{def}}{=} \exists u_0 \cdots \exists u_n\ x=u_0 \wedge \varphi(u_0,u_1) \wedge \varphi(u_1,u_2) \wedge \cdots \wedge \varphi(u_{n-1},u_n) \wedge u_n=y \tag{4.1}$$

[(2)] Bien entendu si $\varphi(x_1, \ldots, x_n)$ est une contrainte quelconque, les x_i des variables et les t_i des termes, $\varphi(t_1, \ldots, t_n)$ désigne cette même contrainte avec les t_i substitués aux occurrences libres des x_i correspondants.

4.1. La contrainte d'itération de formules

On suppose que l'ensemble d'arbres **A** est construit sur un ensemble F de symboles fonctionnels comprenant notamment les symboles 1, 2, 3, d'arités nulles, et le symbole f, d'arité quatre. On se donne une contrainte quelconque $\varphi(x, y)$ et pour $k \geqslant 0$ on introduit la contrainte :

$$\boxed{\mathit{superitérée}_k[\varphi](x, y) \stackrel{\text{def}}{=} \exists z \ \mathit{triangle}_k(3, x, z, y)} \tag{4.2}$$

avec,

$$\boxed{\begin{array}{l} \mathit{triangle}_0(t, x, z, y) \stackrel{\text{def}}{=} z = x \wedge z = y \\ \mathit{triangle}_{k+1}(t, x, z, y) \stackrel{\text{def}}{=} \left[\begin{array}{l} [\exists u_1 \exists u_2 \, z = f(x, u_1, u_2, y)] \\ \wedge \\ \left[\begin{array}{l} \forall t' \forall y' \forall z' \\ \left[\begin{array}{l} (t' = 1 \vee t' = 2) \wedge \\ \mathit{triangle}_k(t', z, y', z') \end{array}\right] \rightarrow \\ \left[\begin{array}{l} (t' = 1 \wedge \mathit{forme1}(z')) \vee \\ (t' = 2 \wedge \left[\begin{array}{l} \exists u \exists v \, \mathit{forme2}(u, z', v) \wedge \\ (t = 1 \rightarrow \mathit{fils}(u, v)) \wedge \\ (t = 2 \rightarrow \mathit{fils}(u, v) \vee u = v) \wedge \\ (t = 3 \rightarrow \varphi(u, v)) \end{array}\right] \end{array}\right] \end{array}\right] \end{array}\right] \end{array}} \tag{4.3}$$

et

$$\boxed{\begin{array}{lll} \mathit{forme1}(x) & \stackrel{\text{def}}{=} & \exists u_1 \ldots \exists u_4 \, x = f(u_1, f(u_2, u_2, u_2, u_2), f(u_3, u_3, u_3, u_3), u_4) \\ \mathit{forme2}(x, z, y) & \stackrel{\text{def}}{=} & \exists u_1 \ldots \exists u_6 \, z = f(u_1, f(u_1, u_2, u_3, x), f(y, u_4, u_5, u_6), u_6) \\ \mathit{fils}(x, y) & \stackrel{\text{def}}{=} & \exists u_1 \ldots \exists u_4 \, x = f(u_1, u_2, u_3, u_4) \wedge (y = u_2 \vee y = u_3) \end{array}} \tag{4.4}$$

La forme des arbres satisfaisant les contraintes $\mathit{forme1}(x)$ et $\mathit{forme2}(x, z, y)$ est donnée dans la figure 4.1. La contrainte $\mathit{fils}(x, y)$ impose que x soit un arbre dont la racine est étiquetée par f, qu'il ait quatre sous-arbres, et que le deuxième ou le troisième sous-arbre de x soit identique à l'arbre y.

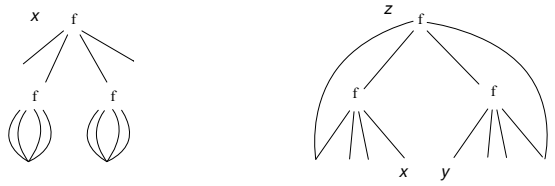

Figure 4.1. La forme des arbres satisfaisant les contraintes $\mathit{forme1}(x)$ et $\mathit{forme2}(x, z, y)$.

Quant à la contrainte $\mathit{triangle}_k(t,x,z,y)$ définie en (4.3), des explications seront présentées dans la section 4.2 afin de montrer que :

$$\begin{array}{lcl}(\exists z\ \mathit{triangle}_k(1,x,z,y)) & \leftrightarrow & \mathit{fils}^{\alpha(k)-1}(x,y),\\ (\exists z\ \mathit{triangle}_k(2,x,z,y)) & \leftrightarrow & \bigvee_{i=0}^{i=\alpha(k)-1}\mathit{fils}^{i}(x,y),\\ (\exists z\ \mathit{triangle}_k(3,x,z,y)) & \leftrightarrow & \varphi^{\alpha(k)-1}(x,y).\end{array} \tag{4.5}$$

Remarquons que la taille de la contrainte $\mathit{superitérée}_k[\varphi](x,y)$ définie en (4.2) dépend linéairement de k. Plus précisément :

Propriété 4.1. — $\boxed{|\ \text{superitérée}_k[\varphi](x,y)| = 9 + k(155 + |\varphi(x,y)|).}$

D'après les formules (4.2) et (4.5), le résultat essentiel est :

Théorème 4.2. — $\boxed{\text{superitérée}_k[\varphi](x,y) \ \leftrightarrow\ \varphi^{\alpha(k)-1}(x,y).}$

4.2. Explication de la contrainte d'itération

La preuve formelle du théorème 4.2 se trouve dans [6]. Nous présentons, pour expliquer l'idée ici, un autre enchaînement parmi les nombreuses versions que nous avons élaborées avec A. Colmerauer. L'élément essentiel est la construction des arbres, appelés *k-oignons*, dont les parties supérieures de profondeur inférieure ou égale à k sont essentiellement des arbres binaires complétés par deux branches (la première et la quatrième) permettant d'avoir des accès directs à la profondeur $k+1$. Ce type d'arbres a été inspiré d'un travail de P. Mielniczuk [20] sur la complexité des algorithmes de décision dans la théorie des arbres avec traits. Pour $k = 3$ de tels arbres sont de la forme donnée dans la figure 4.2.

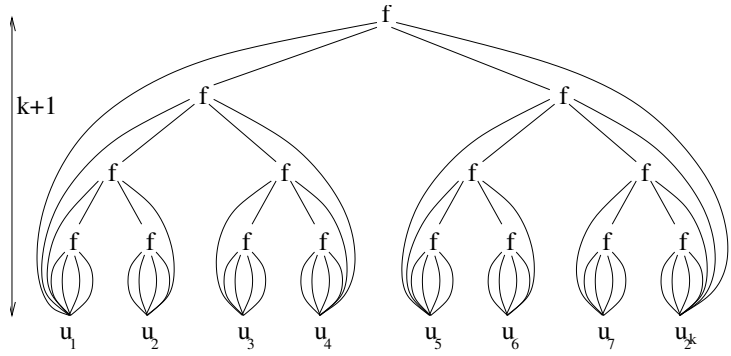

Figure 4.2. La forme des arbres satisfaisant la contrainte $\mathit{oignon}_k(z)$ avec $k = 3$. Les u_i sont des arbres quelconques.

Plus généralement, pour $k \geqslant 0$, un *k-oignon* est un arbre z qui satisfait à la contrainte $oignon_k(z)$, avec :

$$\begin{aligned} oignon_0(z) &\stackrel{\text{def}}{=} \exists u\, z = f(u,u,u,u) \\ oignon_{k+1}(z) &\stackrel{\text{def}}{=} \begin{bmatrix} \exists u\, \exists u_1 \ldots \exists u_4\, \exists v\, \exists v_1 \ldots \exists v_4 \\ u = f(u_1,u_2,u_3,u_4) \wedge oignon_k(u) \wedge \\ v = f(v_1,v_2,v_3,v_4) \wedge oignon_k(v) \wedge \\ z = f(u_1,u,v,v_4) \end{bmatrix} \end{aligned} \tag{4.6}$$

Rappelons qu'un arbre x satisfaisant $forme1\,(x)$ et des arbres x, y, z satisfaisant $forme2\,(x,z,y)$ sont comme dans la figure 4.1. Avec la notation $fils^{\,i}$ définie par (4.1), nous avons l'équivalence suivante (une preuve par induction se trouve dans [6]). Pour tout arbre z et tout $k \geqslant 1$:

$$oignon_k(z) \;\leftrightarrow\; \begin{bmatrix} \begin{bmatrix} \forall z' \\ fils^{\,k-1}(z,z') \rightarrow forme1\,(z') \end{bmatrix} \\ \wedge \\ \begin{bmatrix} \forall z' \\ \left[\bigvee_{i=0}^{k-1} fils^{\,i}(z,z')\right] \rightarrow \left[\exists u \exists v\, forme2\,(u,z',v)\right] \end{bmatrix} \end{bmatrix} \tag{4.7}$$

Considérons un arbre z tel que $oignon_k(z)$. D'après la définition de *fils*, avec $0 \leqslant i \leqslant k-1$, il y a 2^i sous-arbres z' qui satisfont $fils^{\,i}(z,z')$. Nous désignons ces sous-arbres de gauche à droite par $w_1^i, \ldots, w_{2^i}^i$. L'arbre z a aussi 2^k sous-arbres à la profondeur $k+1$, nous les désignons de gauche à droite par $u_1, \ldots, u_{2^k}$.

Pour les sous-arbres w_j^{k-1}, avec $1 \leqslant j \leqslant 2^{k-1}$, les contraintes $forme1\,(w_j^{k-1})$ et $\exists u \exists v\ forme2\,(u, w_j^{k-1}, v)$ sont satisfaites. Ces sous-arbres sont alors de la forme donnée dans la figure 4.3.

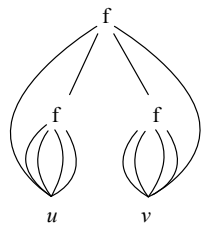

FIGURE 4.3. La forme d'un arbre x satisfaisant les contraintes $forme1\,(x)$ et $\exists u \exists v\, forme2\,(u,x,v)$.

Pour les sous-arbres w_j^i, avec $0 \leqslant i \leqslant k-1$ et $1 \leqslant j \leqslant 2^i$, les contraintes $\exists u \exists v\, forme2\,(u, w_j^i, v)$ sont satisfaites. La contrainte $forme2\,(x,z,y)$ indique que

- le fils le plus à gauche de l'arbre z et le fils le plus à gauche du second fils de z sont identiques, et
- le fils le plus à droite de l'arbre z et le fils le plus à droite du troisième fils de z sont identiques.

On se convainc donc que la contrainte $\exists u \exists v\, forme2\,(u, w^i_j, v)$ entraine que u est le sous-arbre $u_{2^{k-i-1}j}$ et v est le sous-arbre $u_{2^{k-i-1}j+1}$.

Par conséquent, si pour chaque sous-arbre w^i_j, avec $0 \leqslant i \leqslant k-1$ et $1 \leqslant j \leqslant 2^i$, et pour les arbres u et v tels que $forme2\,(u, w^i_j, v)$, il y a une relation $\varphi(u, v)$, alors entre u_1 et u_{2^k} il y a la relation $\varphi^{2^k-1}(u_1, u_{2^k})$.

Nous avons donc l'équivalence suivante pour toute contrainte d'arbre $\varphi(x, y)$ et tout $k \geqslant 0$ [3] :

$$\varphi^{2^k-1}(x,y) \;\leftrightarrow\; \exists z \left[\begin{array}{l} oignon_{\,k}(z) \\ \wedge \\ [\exists v_2 \exists v_3\; z = f(x, v_2, v_3, y)] \\ \wedge \\ \left[\begin{array}{l} \forall z' \\ \left[\bigvee_{i=0}^{k-1} fils^{\,i}(z, z')\right] \rightarrow \\ \left[\begin{array}{l} \exists u \exists v\, forme2\,(u, z', v) \wedge \\ \varphi(u, v) \end{array}\right] \end{array}\right] \end{array}\right] \tag{4.8}$$

Remarquons qu'à la profondeur $k+1$, un arbre z de type k-*oignon* a 2^k sous-arbres $u_1, \ldots, u_{2^k}$, où les u_i sont des arbres quelconques. Et si un autre arbre k-*oignon* z' est construit en mettant les arbres $u_1, \ldots, u_{2^k}$ sur la relation $fils^{\,2^k}$, l'arbre z' aura $2^{(2^k)}$ sous-arbres à la profondeur $2^k + 1$.

Revenons aux équivalences de la formule (4.5), la preuve de ces équivalences peut se faire par induction sur k. Pour rappel, ces équivalences sont :

$$\begin{array}{lcl} (\exists z\;\; triangle_{\,k}(1, x, z, y)) & \leftrightarrow & fils^{\,\alpha(k)-1}(x, y), \\ (\exists z\;\; triangle_{\,k}(2, x, z, y)) & \leftrightarrow & \bigvee_{i=0}^{i=\alpha(k)-1} fils^{\,i}(x, y), \\ (\exists z\;\; triangle_{\,k}(3, x, z, y)) & \leftrightarrow & \varphi^{\,\alpha(k)-1}(x, y). \end{array}$$

Elles sont vraies pour $k = 0$. Supposons qu'elles soient vraies pour un certain $k \geqslant 0$ et montrons qu'elles sont vraies pour $k+1$.

De (4.3), en quantifiant existentiellement z de part et d'autre de la double flèche et en dissociant le cas où $t' = 1$ du cas où $t' = 2$, on déduit l'équivalence :

$$\exists z\;\; triangle_{\,k+1}(t, x, z, y) \;\leftrightarrow\; \exists z \left[\begin{array}{l} [\exists u_1\, \exists u_2\; z = f(x, u_1, u_2, y)] \\ \wedge \\ \left[\begin{array}{l} \forall z' \\ (\exists y'\;\; triangle_{\,k}(1, z, y', z')) \rightarrow \\ forme1\,(z') \end{array}\right] \\ \wedge \\ \left[\begin{array}{l} \forall z' \\ (\exists y'\;\; triangle_{\,k}(2, z, y', z')) \rightarrow \\ \left[\begin{array}{l} \exists u\, \exists v\, forme2\,(u, z', v) \wedge \\ \psi(u, v) \end{array}\right] \end{array}\right] \end{array}\right]$$

[3] Pour $k = 0$ on considère que $\bigvee_{i=0}^{k-1} fils^{\,i}(z, z')$ est la constante logique *faux*

avec :

$$\psi(u,v) \stackrel{\text{def}}{=} \begin{bmatrix} (t=1 \rightarrow \mathit{fils}\,(u,v)) \wedge \\ (t=2 \rightarrow \mathit{fils}\,(u,v) \vee u=v) \wedge \\ (t=3 \rightarrow \varphi(u,v)) \end{bmatrix}$$

Du fait qu'on a supposé les équivalences (4.5) vraies pour k, il s'ensuit que :

$$\exists z\ \mathit{triangle}_{k+1}(t,x,z,y) \leftrightarrow \exists z \begin{bmatrix} [\exists u_2 \exists u_3\ z = f(x,u_2,u_3,y)] \\ \wedge \\ \begin{bmatrix} \forall z' \\ \mathit{fils}^{\alpha(k)-1}(z,z') \rightarrow \\ \mathit{forme1}\,(z') \end{bmatrix} \\ \wedge \\ \begin{bmatrix} \forall z' \\ [\bigvee_{i=0}^{\alpha(k)-1} \mathit{fils}^{i}(z,z')] \rightarrow \\ \begin{bmatrix} \exists u\, \exists v\, \mathit{forme2}\,(u,z',v) \wedge \\ \psi(u,v) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

c'est-à-dire, en se reportant à la formule (4.7),

$$\exists z\ \mathit{triangle}_{k+1}(t,x,z,y) \leftrightarrow \exists z \begin{bmatrix} \mathit{oignon}_{\alpha(k)}(z) \\ \wedge \\ [\exists u_2 \exists u_3\ z = f(x,u_2,u_3,y)] \\ \wedge \\ \begin{bmatrix} \forall z' \\ \left[\bigvee_{i=0}^{\alpha(k)-1} \mathit{fils}^{i}(z,z')\right] \rightarrow \\ \begin{bmatrix} \exists u \exists v\, \mathit{forme2}\,(u,z',v) \wedge \\ \psi(u,v) \end{bmatrix} \end{bmatrix} \end{bmatrix} \tag{4.9}$$

De (4.9), en utilisant la formule (4.8) avec ψ à la place de φ, on déduit :

$$\exists z\ \mathit{triangle}_{k+1}(t,x,y,z) \leftrightarrow \psi^{\alpha(k+1)-1}(u,v)$$

et en donnant successivement à t les valeurs $1, 2, 3$ et en se reportant à la définition de ψ, on obtient (4.5). Ceci, avec la formule (4.2), permet de prouver le théorème 4.2.

5. Quasi universalité des contraintes d'arbres

Ainsi que nous allons le voir, la contrainte d'itération est un bon point de départ pour définir des contraintes expressives.

5.1. Un arbre fini énorme

On peut tout d'abord définir un arbre fini énorme, de $\alpha(k)$ nœuds, par une petite contrainte, de taille dépendant linéairement de k. On suppose que le symbole 0, d'arité nulle, et le symbole s, d'arité un, figure dans F et on introduit la contrainte

$$\boxed{\mathit{énorme}_k(x) \stackrel{\text{def}}{=} \mathit{superitérée}_k[\varphi](x,0),}$$

avec

$$\boxed{\varphi(x,y) \stackrel{\text{def}}{=} x = s(y).}$$

D'après la propriété 4.1 et le théorème 4.2, nous avons la taille de la contrainte :

$$|\mathit{énorme}_k(x)| \;=\; 9{+}159k$$

et la solution qu'elle représente est un arbre fini (sous forme d'une chaîne) énorme de $\alpha(k)$ nœuds :

$$\mathit{énorme}_k(x) \;\leftrightarrow\; x = s^{\alpha(k)-1}(0)$$

5.2. Quasi universalité

Considérons une machine de Turing M et représentons une configuration de la machine par un arbre infini comme dans la figure 5.1.

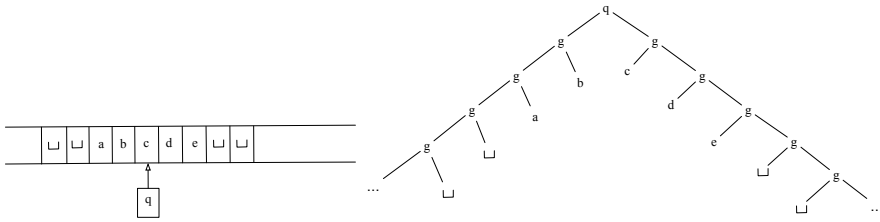

Figure 5.1. Représentation d'une configuration d'une machine de Turing par un arbre fini.

Exprimons par $\varphi(x,y)$ sa contrainte de transition, c'est-à-dire le fait que M puisse passer de la configuration x à la configuration y en exécutant une instruction ou le fait que $x = y$. Par exemple considérons la machine formée de l'ensemble d'instructions[(4)] :

$$\begin{Bmatrix} (q_0,1,1,q_1,D), & (q_0,\sqcup,\sqcup,q_2,G), \\ (q_1,1,\sqcup,q_0,D), & (q_1,\sqcup,1,q_2,G) \end{Bmatrix} \tag{5.1}$$

dont les états sont q_0, q_1, q_2 et l'alphabet $\{\sqcup, 1\}$. La contrainte de transition $\varphi(x,y)$ de cette machine est :

$$\varphi(x,y) \stackrel{\text{def}}{=} \boxed{\begin{array}{l} x = y \;\vee \\ \begin{bmatrix} \exists u\,\exists v\,\exists w \\ x = q_0(u, g(1, g(v,w))) \;\wedge \\ y = q_1(g(u,1), g(v,w)) \end{bmatrix} \;\vee\; \begin{bmatrix} \exists u\,\exists v\,\exists w \\ x = q_0(g(u,v), g(\sqcup,w)) \;\wedge \\ y = q_2(u, g(v, g(\sqcup,w))) \end{bmatrix} \;\vee \\ \begin{bmatrix} \exists u\,\exists v\,\exists w \\ x = q_1(u, g(1, g(v,w))) \;\wedge \\ y = q_0(g(u,\sqcup), g(v,w)) \end{bmatrix} \;\vee\; \begin{bmatrix} \exists u\,\exists v\,\exists w \\ x = q_1(g(u,v), g(\sqcup,w)) \;\wedge \\ y = q_2(u, g(v, g(1,w))) \end{bmatrix} \end{array}}$$

[(4)] L'instruction (q,s,s',q',D) signifie : si la machine est dans l'état q et si sa tête de lecture-écriture est positionnée sur une case contenant le symbole s alors elle remplace s par s', déplace sa tête de lecture-écriture, d'une case à droite et passe à l'état q'. L'instruction (q,s,s',q',G) signifie la même chose, mais avec un déplacement à gauche.

en considérant que $\sqcup$, 1, g, q_0, q_1 et q_2 sont des symboles d'arités respectives 0, 0, 2, 2, 2 et 2.

Supposons d'une façon générale que la machine de Turing M ait pour alphabet $\Sigma \cup \{\sqcup\}$, que son état initial soit q_0, son état final q_h, et que sa contrainte de transition soit $\varphi(x, y)$. Pour tout mot $a = a_1 a_2 \dots a_m$ de $\Sigma^\star$, désignons par $\overline{M}(a)$ l'élément de $\Sigma^\star$ calculé par M, s'il existe [5]. Introduisons les contraintes d'arbres :

$$\begin{array}{lcl}
\mathit{sortie}_k[\varphi, a](v) & \stackrel{\text{def}}{=} & \left[\begin{array}{l}\exists u \exists x \exists y \\ \mathit{est}[a](u) \wedge \mathit{initial}(u, x) \wedge \mathit{final}(y, v) \wedge \\ \mathit{superitérée}_k[\varphi](x, y)\end{array}\right] \\
\mathit{est}[a](u) & \stackrel{\text{def}}{=} & \exists v\, u = g(a_1, g(a_2, \dots, g(a_m, v)\dots)) \wedge v = g(\sqcup, v), \\
\mathit{initial}(u, x) & \stackrel{\text{def}}{=} & \exists v\, x = q_0(v, u) \wedge v = g(v, \sqcup), \\
\mathit{final}(x, u) & \stackrel{\text{def}}{=} & \exists v\, x = q_h(v, u)
\end{array}$$

De la propriété 4.1 et du théorème 4.2, on conclut que :

Propriété 5.1. — $\boxed{|\ \text{sortie}_k[\varphi, a](v)| = 40 + 2|a| + k(155 + |\varphi(x, y)|).}$

Propriété 5.2. — $\boxed{\text{sortie}_k[\varphi, a](v) \leftrightarrow\ \text{est}[\overline{M}(a)](v),}$ *sous réserve que la machine M exécute moins de $\alpha(k)$ instructions.*

Considérons qu'un arbre v qui satisfait à la contrainte $\mathit{est}[b](v)$ est un *codage explicite* du mot $b = b_1 \dots b_n$ et désignons par $|M|$ le nombre d'instructions de la machine M. Des deux propriétés précédentes on déduit alors :

Corollaire 5.3 (Quasi universalité des contraintes d'arbres). — *Pour tout mot b, qu'une machine de Turing M peut calculer en exécutant moins de $\alpha(k)$ instructions, à partir d'un mot a, il existe une contrainte d'arbre $\psi(v)$ de taille $O(|M|k + |a|)$, dont l'unique solution en v est un arbre codant explicitement b.*

6. Résolution de contraintes d'arbres

Les propriétés de l'algèbre des arbres finis ou infinis ont été axiomatisées en une théorie du premier ordre par M. Maher [18]. Cette théorie T est définie sur un ensemble infini de symboles de fonction F et l'unique symbole de relation de l'égalité =. La théorie est l'ensemble de propositions de l'une des trois formes suivantes :

$$\begin{array}{ll}
\forall x_1 \dots \forall x_n \forall y_1 \dots \forall y_m & \neg(f(x_1, \dots, x_n) = g(y_1, \dots, y_m)) \\
\forall x_1 \dots \forall x_n \forall y_1 \dots \forall y_n & f(x_1, \dots, x_n) = f(y_1, \dots, y_n) \ \rightarrow \bigwedge_{i=1}^{n} x_i = y_i \\
\forall x_1 \dots \forall x_n \exists! y_1 \dots \exists! y_m & \bigwedge_{i=1}^{m} y_i = t_i[x_1, \dots, x_n, y_1, \dots, y_m]
\end{array} \qquad (6.1)$$

[5] Au départ la machine est dans l'état initial q_0 et le ruban biinfini est rempli de symboles $\sqcup$, sauf une portion commençant à la position de la tête de lecture écriture, qui contient le mot $a \in \Sigma^\star$. Ensuite la machine, qui est supposée être déterministe, exécute les instructions jusqu'à atteindre l'état final q_h. Le mot calculé est l'élément de $\Sigma^\star$ le plus long qui commence à la position finale de la tête de lecture-écriture. Par exemple, avec l'état initial q_0, avec l'état final q_2, avec n 1 en entrée, la machine (5.1) sort le mot vide, si n est pair, et sort 11, si n est impair.

où $f, g \in F$ avec $f \neq g$, et $t_i[x_1, \ldots, x_n, y_1, \ldots, y_m]$ est un terme formé par un élément de F et par des variables prises de l'ensemble $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$. M. Maher a montré que cette théorie est complète, i.e. pour toute formule p sans variable libre, soit $T \models p$, soit $T \models \neg p$. Pour cela, il a proposé une méthode d'élimination de quantificateurs, qui, en s'appliquant à une formule p sans variables libres, transforme la formule soit en *vrai* (on a donc $T \models p$), soit en *faux* (donc $T \models \neg p$). Cette théorie est appelée la théorie des arbres, car l'algèbre des arbres finis ou infinis est un modèle de cette théorie.

La section 5 a montré que des contraintes d'arbres en logique du premier ordre ont un pouvoir d'expressivité quasi-universel au sens du Corollaire 5.3. La complexité des algorithmes pour les résoudre ne peut donc qu'être très élevée. En effet les propriétés 5.1 et 5.2 permettent de redémontrer le résultat de S. Vorobyov [24], que la complexité en temps d'un algorithme qui décide si une formule sans variable libre est vraie ne peut être majorée par une fonction élémentaire.

Les contrainte d'arbres, quant à elles, sont en général définies avec des variables libres, par exemple $gagnant_k(x)$ indique que l'arbre x représente une position k-gagnante. La résolution d'une contrainte d'arbre sur les variables libres $x_1, \ldots, x_n$ consiste à expliciter les affectations à chaque variable libre x_i un arbre tel que la contrainte soit satisfaite. Dans [11], nous avons développé une méthode pour résoudre des contraintes d'arbres. Une forme résolue des contraintes d'arbres a été définie, où les solutions sont exprimées explicitement. Une formule résolue $f(\overline{x})$ définie sur un vecteur de variables libres $\overline{x}$ est de la forme

$$\bigvee_{i=1}^{k} (\exists \overline{y_i} a_i \wedge \bigwedge_{j=1}^{k_i} \neg(\exists \overline{z_{ij}} b_{ij})) \tag{6.2}$$

où les y_i et z_{ij} sont des vecteurs de variables et a_i et b_{ij} sont des conjonctions d'équations satisfaisant des conditions particulières. Par exemple la formule suivante sur une variable libre x est sous forme résolue, et elle représente les deux arbres de x définis dans (3.4) :

$$\begin{bmatrix} \exists y_1 y_2 (x = c(y_1, y_2) \wedge y_1 = g(y2) \wedge y_2 = 0) & \vee \\ \exists y_3 y_4 (x = c(y_3, y_4) \wedge y_3 = 0 \wedge y_4 = g(y3)) & \end{bmatrix}$$

Remarquons qu'une formule résolue est définie avec au plus deux niveaux d'imbrication de quantificateurs. Chaque élément de la disjonction (6.2) représente une solution possible. Une contrainte d'arbres, comme par exemple (3.3), peut en général être définie avec plusieurs niveaux d'imbrication de quantificateurs.

Dans [11], une méthode d'élimination de quantificateurs a été développée, permettant de transformer une formule du premier ordre en une formule équivalente. La résolution de contraintes est représentée comme une suite de transformations, qui transforme la contrainte de départ C en une formule finale équivalente, qui est soit *faux* (si C n'a pas de solution), soit une formule sous forme résolue (les solutions sont explicites). Les transformations possibles sont exprimées par des règles de réécriture de sous-formules, et un ensemble de 11 règles de réécriture a été proposé pour représenter

toutes ces transformations. Chaque règle de réécriture transforme une sous-formule en une formule équivalente. Pour résoudre une contrainte, les règles de réécriture s'applique autant de fois que possible et peuvent s'appliquer sur différentes sous-formules d'une façon indépendante. Une preuve de terminaison a été présentée, qui montre que l'application des règles se termine toujours, et que la contrainte finale est soit *faux*, soit sous la forme résolue. Un solveur a été implanté en C++ et a été capable de résoudre des contraintes complexes, par exemple celles de positions gagnantes, faisant intervenir jusqu'à 160 imbrications de quantificateurs. Pour plus de détails, nous renvoyons le lecteur vers [11, 12].

Les arbres de l'algèbre peuvent être finis ou infinis. Afin de pouvoir restreindre explicitement certaines conditions sur les arbres finis, nous avons étendu la théorie des arbres avec la relation $\mathit{fini}(t)$, indiquant que t est un arbre fini. Exploitant les règles de réécriture, nous avons développé un algorithme pour la résolution de contraintes dans cette théorie et l'avons implanté en C++ avec le formalisme du CHR [13].

7. Conclusion

Nous avons présenté dans cet article des résultats principaux de l'article écrit avec A. Colmerauer [6] sur le pouvoir d'expression des contraintes représentées par des formules générales du premier ordre, avec pour seul symbole de relation l'égalité et pour symboles de fonction les éléments d'un ensemble infini F. Le domaine choisi est l'ensemble des arbres, dont les nœuds, en nombre éventuellement infini, sont étiquetés par des éléments de F. Des contraintes faisant intervenir des suites alternées de quantificateurs $\exists\forall\exists\forall\ldots$ permettent d'exprimer les positions gagnantes d'un jeu à deux adversaires. Des contraintes d'arbres ont également le pouvoir de décrire brièvement des arbres que l'ordinateur le plus puissant n'aura jamais le temps de calculer. Pour plus de détails et de contenu, nous renvoyons le lecteur vers le papier original. La complexité pour résoudre ces contraintes ne peut être majorée par une fonction élémentaire. Nous avons dévelopé une méthode de résolution de ces contraintes par des règles de réécriture de sous-formules, pour transformer la contrainte de départ en une formule équivalente, qui est soit *faux* si la contrainte n'a pas de solution, soit une forme résolue finale où les solutions sont explicites.

Bibliographie

[1] F. Benhamou, P. Bouvier, A. Colmerauer, H. Garetta, B. Giletta, J. Massat, G. Narboni, S. N'Dong, R. Pasero, J. Pique, Touraïvane, M. Van Caneghem & E. Vétillard, *Le manuel de Prolog IV*, PrologIA, Marseille, Juin 1996.

[2] H.-J. Bürckert, « Solving disequations in equational theories », in *Proc. 9th Conf. on Automated Deduction*, LNCS 310, Springer-Verlag, 1988, p. 517-526.

[3] A. Colmerauer, « Prolog and Infinite Trees », in *Logic Programming* (New York) (K. L. Clark & S.-A. Tarnlund, éds.), Academic Press, 1982, p. 231-251.

[4] ———, « Equations and Inequations on Finite and Infinite Trees », in *Proceeding of the International Conference on Fifth Generation Computer Systems (FCGS-84)* (Tokyo), ICOT, 1984, p. 85-99.

[5] ———, « An Introduction to Prolog III », *Communications of the ACM* **33** (1990), n° 7, p. 68-90.

[6] A. Colmerauer & T.-B.-H. Dao, « Expressiveness of Full First-Order Constraints in the Algebra of Finite or Infinite Trees », *Constraints* **8** (2003), n° 3, p. 283-302.

[7] A. Colmerauer, H. Kanoui & M. Van Caneghem, « Prolog, Theoretical Principles and Current Trends », *Technology and Science of Informatics* **2** (1983), n° 4, p. 255-292, Version anglaise de la revue *TSI*, AFCET-Bordas.

[8] H. Comon, « Disunification: a Survey », in *Computational Logic: Essays in Honor of Alan Robinson* (J. Lassez & G. Plotkin, éds.), MIT Press, 1991.

[9] ———, « Résolution de contraintes dans des algèbres de termes », Rapport d'habilitation, Université de Paris Sud, 1992.

[10] H. Comon & P. Lescanne, « Equational Problems and Disunification », *Journal of Symbolic Computation* **7** (1989), n° 3-4, p. 371-425.

[11] T.-B.-H. Dao, « Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis », Thèse d'informatique, Université Aix-Marseille 2, 2000.

[12] ———, « Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis », in *Programmation en logique avec contraintes, JFPLC 2000, 28-30 Juin 2000, Marseille, France* (Touraïvane, éd.), Hermes, 2000, p. 225-240.

[13] K. Djelloul, T.-B.-H. Dao & T. W. Frühwirth, « Theory of finite or infinite trees revisited », *Theory Pract. Log. Program.* **8** (2008), n° 4, p. 431-489.

[14] G. Huet, « Résolution d'équations dans les langages d'ordre 1, 2, . . . , ω », Thèse d'État, Université Paris 7, 1976.

[15] J. Jaffar, « Efficient unification over infinite terms », *New Generation Computing* **2** (1984), n° 3, p. 207-219.

[16] J.-P. Jouannaud & C. Kirchner, « Solving Equations in Abstract algebras: A Rule-Based survey of unifications », in *Computational Logic: Essays in Honor of Alan Robinson* (J. Lassez & G. Plotkin, éds.), MIT Press, 1991, p. 257-331.

[17] K. Kunen, « Negation in Logic Programming », *Journal of Logic Programming* **4** (1987), p. 289-308.

[18] M. J. Maher, « Complete Axiomatization of the Algebra of Finite, Rational and Infinite Trees », Tech. report, IBM – T.J. Watson Research Center, 1988.

[19] A. Malcev, « Axiomatizable classes of locally free algebras of various types », in *The Metamathematics of Algebraic Systems. Anatolii Ivanovic Malcev. Collected Papers: 1936-1967* (B. W. III, éd.), vol. 66, North Holland, 1971, p. 262-281.

[20] P. Mielniczuk, « Basic theory of feature trees », *ACM Trans. Comput. Log.* **5** (2004), n° 3, p. 385-402.

[21] V. Ramachandran & P. Van Hentenryck, « Incremental algorithms for constraint solving and entailment over rational trees », in *Proc. of the 13th Conference Foundations of Software Technology and Theoretical Computer Science*, LNCS, vol. 761, 1993, p. 205-217.

[22] J. Robinson, « A machine-oriented logic based on the resolution principle », *JACM* **12** (1965), n° 1, p. 23-41.

[23] D. A. Smith, « Constraint operations for CLP($\mathcal{FT}$) », in *Logic Programming: Proceedings of the 8th International Conference* (Paris), 1991, p. 760-774.

[24] S. Vorobyov, « An Improved Lower Bound for the Elementary Theories of Trees », in *Proceeding of the 13th International Conference on Automated Deduction, CADE'96*, Lecture Notes in Artificial Intelligence, vol. 1104, Springer, 1996, p. 275-287.

Abstract. — The algebra of finite or infinite trees plays a fundamental role in computer science. Among others, the execution of Prolog II, III and IV programs was modeled by Alain Colmerauer as solving constraints in the form of equations and disequations in this algebra. In a work published in the journal Constraints with Alain Colmerauer [6], we studied the expressiveness of more general constraints in this algebra, which are first-order logical formulas, constructed using variables designating trees, construction operations, the equality relation, logical connectors and quantifiers. We have presented examples to show an immense expressivity power of such general constraints. This paper presents a reduced version of this work on the expressiveness of these constraints [6]. Moreover, we describe the method we have developed to solve such constraints.

Keywords. — Tree algebra, Constraints, Expressiveness of constraint, Constraint solving.

Manuscrit reçu le 27 mai 2024, accepté le 12 juillet 2024.