

Pouvoir d'expression des contraintes du 1^{er} ordre dans l'algèbre des arbres finis ou infinis

Alain Colmerauer

Thi-Bich-Hanh Dao*

3 avril 2000

Résumé

Nous nous intéressons au pouvoir d'expression de contraintes représentées par des formules générales du premier ordre avec pour seul symbole de relation l'égalité et pour symboles de fonctions 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 les éléments de F . L'opération associée à chaque élément f de F , d'arité n , est l'application $(a_1, \dots, a_n) \mapsto b$, où b est l'arbre dont le nœud initial est étiqueté f et dont la suite de fils est a_1, \dots, a_n .

Nous examinons tout d'abord des contraintes faisant intervenir de longues suites alternées de quantificateurs $\exists \forall \exists \forall \dots$. Nous montrons comment exprimer les positions gagnantes d'un jeu à deux adversaires par de tels contraintes et appliquons nos résultats à deux exemples de jeux.

Nous construisons ensuite une famille de contraintes très expressives, inspirée d'une démonstration constructive d'un résultat de complexité de Pawel Mielniczuk. Cette famille fait intervenir le nombre immensément grand $\alpha(k)$, obtenu en évaluant de haut en bas une tour de puissances de 2 de hauteur k . Elle nous permet, en une contrainte de longueur au plus proportionnelle à k , de définir un arbre ayant exactement $\alpha(k)$ nœuds ou d'exprimer le résultat d'une machine Prolog exécutant au plus $\alpha(k)$ instructions.

En remplaçant la machine Prolog par une machine de Turing nous retrouvons le résultat suivant de Sergei Vorobyov : la complexité d'un algorithme décidant si une contrainte sans variable est vraie, ne peut être majorée par une fonction obtenue par composition finie de fonctions élémentaires (comprenant l'exponentiation).

Enfin, profitant du fait que nous disposons d'un algorithme pour résoudre de telles contraintes dans toute leur généralité, nous effectuons un ensemble de benchmarks permettant de départager les exemples réalisables des exemples purement spéculatifs. Nous arrivons notamment à résoudre des contraintes faisant intervenir des suites alternées de plus de 160 quantificateurs.

1 Introduction

L'algèbre des arbres (éventuellement) infinis joue un rôle fondamental en informatique. Elle modélise aussi bien des structure de données que des schémas ou des déroulements de programme. Dès 1976, Gérard Huet a proposé un algorithme pour unifier des termes infinis, c'est-à-dire résoudre des équations dans cette algèbre [11]. Bruno Courcelle a étudié les propriétés des arbres infinis notamment dans le cadre des schémas récursifs de programme [8, 9]. Alain Colmerauer a modélisé le fonctionnement de Prolog II, III et IV

*Laboratoire d'Informatique de Marseille, CNRS, Universités de la Méditerranée et de Provence

par résolution d'équations et de diséquations dans les arbres infinis [4, 5, 6, 1]. Michael Maher a proposé et justifié une théorie complète de l'algèbre des arbres infinis [12]. Entre autres il a montré que dans cette théorie, et donc dans l'algèbre des arbres infinis, toute formule du premier ordre était équivalente à une combinaison booléenne de conjonctions d'équations quantifiées (entièrement ou partiellement) existentiellement. Sergei Vorobyov a montré qu'aucun algorithme pour décider si une telle formule, sans variables libres, est vraie dans l'algèbre des arbres infinis n'a une complexité majorée par une composition finie de fonctions élémentaires comprenant l'exponentiation [14]. Pawel Mielniczuk a montré un résultat analogue dans la théorie des arbres avec traits, mais avec une méthode plus constructive, qui a inspiré certains de nos exemples [13].

Pour notre part nous avons développé un algorithme pour résoudre des contraintes générales du premier ordre dans l'algèbre des arbres [10]. L'objet de ce papier n'est pas de présenter cet algorithme, mais des exemples, d'abord imaginés pour le tester, puis développés pour montrer l'expressivité de contraintes aussi générales. Le papier est organisé comme suit.

(1) Nous terminons cette première section en précisant les notions d'algèbre des arbres infinis et contraintes du premier ordre dans cette algèbre.

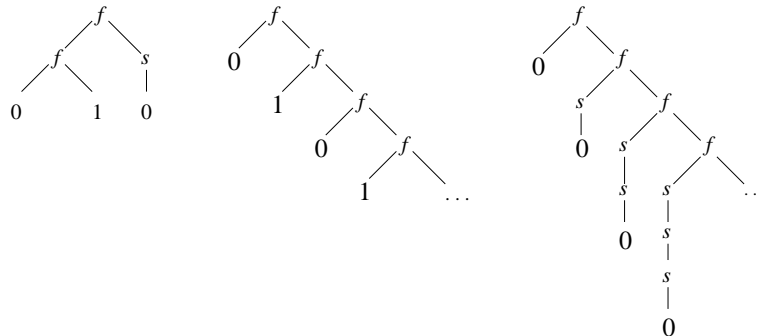
(2) Dans la deuxième section, nous examinons des contraintes faisant intervenir de longues suites alternées de quantificateurs $\exists \forall \exists \forall \dots$. Nous montrons comment exprimer les positions gagnantes d'un jeu à deux adversaires par de telles contraintes et appliquons nos résultats à deux exemples.

(3) Dans la troisième section, nous nous intéressons à la famille de contraintes la plus expressive que nous connaissons et qui fait intervenir le nombre immensément grand $\alpha(k)$ obtenu en évaluant de haut en bas une tour de puissances de 2 de hauteur k . Avec des éléments de cette famille, de taille linéaire en k , nous contraignons une variable à être égale à un arbre fini, dont le nombre de nœuds est $\alpha(k)$, et nous exprimons le résultat d'une machine Prolog exécutant au plus $\alpha(k)$ instructions. En remplaçant la machine Prolog par une machine de Turing nous retrouvons le résultat de complexité de Sergei Vorobyov mentionné au début de cette section. Cette partie a été fortement inspirée par le travail de Pawel Mielniczuk [13].

(4) Nous terminons par quelques remarques et notamment des benchmarks départageant les exemples réalisables des exemples purement spéculatifs.

1.1 L'algèbre des arbres infinis

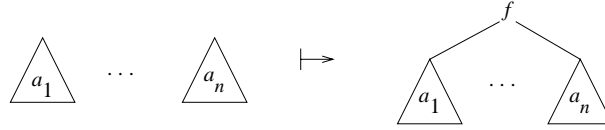
Les arbres sont des objets bien connus en informatique. En voici quelques-uns :



Leurs nœuds sont étiquetés par les symboles $0, 1, s, f$ d'arités respectives $0, 0, 1, 2$ pris dans

un ensemble F de symboles fonctionnels, ensemble que nous supposons infini. 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. Nous désignerons par \mathbf{A} l'ensemble de tous les arbres¹ construits sur F .

On munit l'ensemble \mathbf{A} d'un ensemble d'opérations de constructions², une pour chaque élément $f \in F$. Elle consiste en l'application $(a_1, \dots, a_n) \mapsto a$, 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, \dots, a_n . Schématiquement on a :



On obtient ainsi l'*algèbre des arbres infinis* construits sur F , que l'on note (\mathbf{A}, F) .

1.2 Les contraintes d'arbres

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

$$s=t, \text{ vrai, faux, } \neg(p), (p \wedge q), (p \vee q), (p \rightarrow q), \exists x p, \forall x p,$$

où p et q 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

$$x, f t_1 \dots t_n$$

avec $n \geq 0$, $f \in F$ 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 et les symboles de fonctions f seront interprétés comme des opérations de construction dans l'algèbre des arbres infinis (\mathbf{A}, F) . Une contrainte sans variable libre sera donc vraie ou fausse et une contrainte $p(x_1, \dots, x_n)$ faisant intervenir n variables libres x_i établira une relation n -aire dans l'ensemble des arbres.

2 Longues imbrications de quantifications alternées

On introduit tout d'abord les notions de positions k -gagnantes et k -perdantes et on les illustre à travers deux exemples de jeux à deux adversaires. On montre comment exprimer, dans un domaine quelconque, l'ensemble des positions k -gagnantes d'un jeu par

¹Plus précisément on définit d'abord un *nœud* comme un mot construit sur l'ensemble des entiers positifs. Un *arbre* a , construit sur F , est alors une application de type $E \rightarrow F$, où E est un ensemble non vide de nœuds dont chaque élément $i_1 \dots i_k$ (avec $k \geq 0$) respecte deux conditions : (1) si $k > 0$ alors $i_1 \dots i_{k-1} \in E$, (2) si l'arité de $a(i_1 \dots i_k)$ est n alors, l'ensemble des nœuds de E de la forme $i_1 \dots i_k i_{k+1}$ s'obtient en donnant à i_{k+1} les valeurs $1, \dots, n$.

²En fait, l'opération *de construction* associée au symbole n -aire f de F est l'application $(a_1, \dots, a_n) \mapsto a$, où les a_i sont des arbres quelconques et a est l'arbre défini comme suit à partir des a_i et de leurs ensembles E_i de nœuds : l'ensemble E de nœuds de a est $\{\varepsilon\} \cup \{ix \mid x \in E_i \text{ et } i \in 1..n\}$ et, pour tout $x \in E$, si $x = \varepsilon$, on a $a(x) = f$ et si x est de la forme iy , avec i un entier, $a(x) = a_i(y)$.

une contrainte. On termine la section par des contraintes d'arbre exprimant les positions k -gagnantes des deux exemples de jeux et faisant intervenir une imbrication de $2k$ quantifications alternées.

2.1 Positions gagnantes dans un jeu à deux adversaires

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 telle 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 les deux graphes infinis suivants schématisent les jeux suivants :



Jeu 1 On se donne un entier positif ou nul i et à tour de rôle on soustrait 1 ou 2 de i sans jamais rendre i strictement négatif. La première personne qui ne peut plus jouer a perdu.

Jeu 2 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.

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.

Considérons les deux graphes précédents et marquons $+k$ les positions qui sont k -gagnantes et $-k$ les positions qui sont k -perdantes, avec chaque fois k le plus petit possible. Le sommet 0 du premier graphe et le sommet $(0, 0)$ du deuxième graphe étant les seuls sommets 0-perdants, on les marque de -0 . En partant des sommets marqués -0 et en remontant les flèches à l'envers on détermine, par vagues successives, les ensembles de sommets à marquer : $+1, -1, +2, -2, +3, -3, \dots$. On obtient



et on se convainc que l'ensemble des positions k -gagnantes du premier jeu est

$$\{i \in \mathbf{N} \mid i < 3k \text{ et } i \bmod 3 \neq 0\}$$

et celui du deuxième

$$\{(i, j) \in \mathbf{N}^2 \mid i+j < 2k \text{ et } (i+j) \bmod 2 = 1\}.$$

où \mathbf{N} est l'ensemble des entiers naturels.

2.2 Expression des positions k -gagnantes par une contrainte

Donnons nous un *domaine* \mathbf{D} , c'est-à-dire un ensemble non vide et un jeu à deux adversaires représenté par le graphe $G = (V, E)$, avec $V \subseteq \mathbf{D}$. Nous allons exprimer les positions k -gagnantes de G par une contrainte dans \mathbf{D} faisant intervenir une imbrication $\exists \forall \exists \dots$ de $2k$ quantificateurs alternés.

Introduisons dans \mathbf{D} les propriétés *coup*, *gagnant_k* et *perdant_k*, définies par

$$\begin{aligned} \text{coup}(x, y) &\leftrightarrow (x, y) \in E, \\ \text{gagnant}_k(x) &\leftrightarrow x \text{ est une position } k\text{-gagnante de } G, \\ \text{perdant}_k(x) &\leftrightarrow x \text{ est une position } k\text{-perdante de } G. \end{aligned} \tag{1}$$

Dans \mathbf{D} on a alors les équivalences, pour tout $k \geq 0$:

$$\begin{aligned} \text{gagnant}_0(x) &\leftrightarrow \text{faux}, \\ \text{gagnant}_{k+1}(x) &\leftrightarrow \exists y \text{ coup}(x, y) \wedge \text{perdant}_k(y), \\ \text{perdant}_k(x) &\leftrightarrow \forall y \text{ coup}(x, y) \rightarrow \text{gagnant}_k(y). \end{aligned} \tag{2}$$

Contrairement à ce que l'on pourrait croire, il s'ensuit qu'on a bien :

$$\text{gagnant}_k(x) \rightarrow \text{gagnant}_{k+1}(x), \quad \text{perdant}_k(x) \rightarrow \text{perdant}_{k+1}(x).$$

En effet, de la première et de la dernière équivalences de (2) on conclut que ces implications sont vraies pour $k = 0$ et, si l'on suppose qu'elle sont vraies pour un certain $k \geq 0$, des deux dernières équivalences de (2) on conclut qu'elles le sont aussi pour $k+1$.

De (3) on déduit une formulation explicite de *gagnant_k*, pour tout $k \geq 0$:

$$\text{gagnant}_k(x) \leftrightarrow \left[\begin{array}{l} \exists y \text{ coup}(x, y) \wedge \neg(\\ \exists x \text{ coup}(y, x) \wedge \neg(\\ \exists y \text{ coup}(x, y) \wedge \neg(\\ \exists x \text{ coup}(y, x) \wedge \neg(\\ \dots \\ \exists y \text{ coup}(x, y) \wedge \neg(\\ \exists x \text{ coup}(y, x) \wedge \neg(\\ \text{faux} \quad \underbrace{\quad \quad \quad}_{2k} \end{array} \right] \tag{3}$$

où bien entendu toutes les quantifications portent sur des éléments de \mathbf{D} . En descendant les négations, on obtient ainsi une imbrication de $2k$ quantificateurs alternés.

Dans l'équivalence (3) on peut utiliser une définition de *coup* plus générale que celle donnée en (1). On remarque que, pour tout entier k positif ou nul, on a la propriété suivante :

Propriété 1 Soient trois graphes orientés de la forme $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ et $G = (V_1 \cup V_2, E_1 \cup E_2)$. Les graphes G_1 et G ont le même ensemble de positions k -gagnantes si à la fois :

1. les ensembles de sommets V_1 et V_2 sont disjoints,
2. pour tout $x \in V_2$, il existe $y \in V_2$ avec $(x, y) \in E_2$.

En effet, de la première condition on déduit que E_1 et E_2 sont disjoints et donc que l'ensemble des positions k -gagnantes de G est l'union de l'ensemble des positions k -gagnantes de G_1 avec l'ensemble des positions k -gagnantes de G_2 . Ce dernier ensemble est vide du fait de la deuxième condition.

On en conclut que :

Propriété 2 (Relation coup généralisé) L'équivalence (3) est aussi vraie pour toute relation coup respectant les deux conditions :

1. pour tous $x \in V$ et $y \in V$, $\text{coup}(x, y) \leftrightarrow (x, y) \in E$,
2. pour tout $x \in D - V$ il existe $y \in D - V$ tel que $\text{coup}(x, y)$.

2.3 Formalisation du jeu 1 dans l'algèbre des arbres infinis

Reprenons le jeu 1 introduit à la section 2.1. Prenons comme domaine \mathbf{D} l'ensemble \mathbf{A} des arbres construits sur un ensemble F de symboles fonctionnels comprenant notamment les symboles $0, s$, d'arité respectives $0, 1$. Codons les sommets i du graphe du jeu par les arbres³ $s^i(0)$. Soit $G = (V, E)$ le graphe ainsi obtenu.

Pour relation coup généralisée on peut alors prendre dans l'algèbre des arbres infinis :

$$\text{coup}(x, y) \stackrel{\text{def}}{=} x = s(y) \vee x = s(s(y)) \vee (\neg(x = 0) \wedge \neg(\exists u x = s(u)) \wedge x = y)$$

et d'après la propriété 2 l'ensemble des positions k -gagnantes du jeu 1 est l'ensemble des solutions en x de la contrainte $\text{gagnant}_k(x)$ définie en (3).

Par exemple, pour $k = 1$ la contrainte $\text{gagnant}_k(x)$ est équivalente à

$$x = s(0) \vee x = s(s(0))$$

et pour $k = 2$ à

$$x = s(0) \vee x = s(s(0)) \vee x = s(s(s(s(0)))) \vee x = s(s(s(s(s(0))))))$$

2.4 Formalisation du jeu 2 dans l'algèbre des arbres infinis

Reprenons le jeu 2 de la section 2.1. Prenons comme domaine \mathbf{D} l'ensemble \mathbf{A} des arbres construits sur un ensemble F de symboles fonctionnels comprenant notamment les symboles $0, f, g, c$, d'arités respectives $0, 1, 1, 2$. Codons les sommets (i, j) du graphe du jeu par les arbres $c(\vec{i}, \vec{j})$ avec $\vec{i} = (fg)^{\frac{i}{2}}(0)$ si i est pair, et $\vec{i} = g(\overline{i-1})$ si i est impair⁴. Soit $G = (V, E)$ le graphe ainsi obtenu.

Le lecteur perspicace se convaincra que pour relation généralisée coup on peut prendre dans l'algèbre des arbres infinis :

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

³Bien entendu, $s^0(0) = 0$ et $s^{i+1}(0) = s(s^i(0))$.

⁴Bien entendu, $(fg)^0(x) = x$ et $(fg)^{i+1}(x) = (fg)^i(fg(x))$.

avec

$$\begin{aligned}
\text{transition}(x, y) &\stackrel{\text{def}}{=} \left[\begin{array}{l} \exists u \exists v \exists w \\ \left[\begin{array}{l} (x = c(u, v) \wedge y = c(u, w)) \vee \\ (x = c(v, u) \wedge y = c(w, u)) \end{array} \right] \\ \wedge \\ \left[\begin{array}{l} (\exists i u = g(i) \wedge \text{succ}(v, w)) \vee \\ (\neg(\exists i u = g(i)) \wedge \text{pred}(v, w)) \end{array} \right] \end{array} \right] \\
\text{succ}(v, w) &\stackrel{\text{def}}{=} \left[\begin{array}{l} ((\exists j v = g(j)) \wedge w = f(v)) \vee \\ (\neg(\exists j v = g(j)) \wedge w = g(v)) \end{array} \right] \\
\text{pred}(v, w) &\stackrel{\text{def}}{=} \left[\begin{array}{l} (\exists j v = f(j) \wedge \left[\begin{array}{l} (\exists k j = g(k) \wedge w = j) \vee \\ (\neg(\exists k j = g(k)) \wedge w = v) \end{array} \right]) \vee \\ (\exists j v = g(j) \wedge \left[\begin{array}{l} (\exists k j = g(k) \wedge w = v) \vee \\ (\neg(\exists k j = g(k)) \wedge w = j) \end{array} \right]) \vee \\ (\neg(\exists j v = f(j)) \wedge \neg(\exists j v = g(j)) \wedge \\ \neg(v = 0) \wedge w = v) \end{array} \right]
\end{aligned}$$

D'après la propriété 2 l'ensemble des positions k -gagnantes du jeu 2 est l'ensemble des solution en x de la contrainte $\text{gagnant}_k(x)$ définie en (3).

Par exemple, pour $k = 1$ la contrainte $\text{gagnant}_k(x)$ est équivalente à

$$x = c(g(0), 0) \vee x = c(0, g(0))$$

et pour $k = 2$ à

$$\left[\begin{array}{l} x = c(0, g(0)) \vee x = c(g(0), 0) \vee x = c(0, g(f(g(0)))) \vee \\ x = c(g(0), f(g(0))) \vee x = c(f(g(0)), g(0)) \vee x = c(g(f(g(0))), 0) \end{array} \right]$$

3 Quasi universalité des contraintes dans les arbres

Après toutes ces quantifications, nous abordons des contraintes tellement expressives que leur résolution devient quasi indécidable.

3.1 Une contrainte définissant un arbre fini énorme

On pose $\alpha(k) = 2^{2^{\dots^2}}$, avec k occurrences de 2. Plus précisément on prend

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

avec $k \geq 0$. 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 suppose que l'ensemble d'arbre \mathbf{A} est construit sur un ensemble F de symboles fonctionnels comprenant notamment les symboles $0, 1, 2, 3, s, f$, d'arités respectives $0, 0, 0, 0, 1, 4$. Pour $k \geq 0$ introduisons la contrainte :

$$\acute{e}norme_k(x) \stackrel{\text{def}}{=} \exists z \text{ triangle}_k(3, x, z, 0)$$

avec toujours pour $k \geq 0$,

$$\begin{aligned} \text{triangle}_0(t, x, z, y) &\stackrel{\text{def}}{=} z = x \wedge z = y \\ \text{triangle}_{k+1}(t, x, z, y) &\stackrel{\text{def}}{=} \left[\begin{array}{l} [\exists u_1 \exists u_2 z = f(x, u_1, u_2, y)] \\ \wedge \\ \left[\forall t' \forall y' \forall z' \right. \\ \left. \left[(t' = 1 \vee t' = 2) \wedge \right. \right. \\ \left. \left. \text{triangle}_k(t', z, z', y') \right] \rightarrow \right. \\ \left. \left[(t' = 1 \wedge \text{forme1}(y')) \vee \right. \right. \\ \left. \left. \left[(t' = 2 \wedge \left[\begin{array}{l} \exists u \exists v \text{ forme2}(u, y', v) \wedge \\ (t = 1 \rightarrow \text{trans1}(u, v)) \wedge \\ (t = 2 \rightarrow \text{trans2}(u, v)) \wedge \\ (t = 3 \rightarrow \text{trans3}(u, v)) \end{array} \right] \right] \right] \right] \end{array} \right] \end{array} \quad (4)$$

et

$$\begin{aligned} \text{forme1}(x) &\stackrel{\text{def}}{=} \exists u_1 \dots \exists u_4 x = f(u_1, f(u_2, u_2, u_2, u_2), f(u_3, u_3, u_3, u_3), u_4) \\ \text{forme2}(x, z, y) &\stackrel{\text{def}}{=} \exists u_1 \dots \exists u_6 z = f(u_1, f(u_1, u_2, u_3, x), f(y, u_4, u_5, u_6), u_6) \\ \text{trans1}(x, y) &\stackrel{\text{def}}{=} \exists u_1 \dots \exists u_4 x = f(u_1, u_2, u_3, u_4) \wedge (y = u_2 \vee y = u_3) \\ \text{trans2}(x, y) &\stackrel{\text{def}}{=} \text{trans1}(x, y) \vee x = y \\ \text{trans3}(x, y) &\stackrel{\text{def}}{=} x = s(y) \end{aligned} \quad (5)$$

Convenons que la taille $|p|$ d'une contrainte p est le nombre d'occurrences de tous les symboles à l'exception des parenthèses et des virgules. (Les contraintes pourraient être écrites en notation infixée.) On a la double propriété :

Propriété 3 (petite contrainte, gros arbre)

$$\boxed{|\acute{e}norme_k(x)| = 9 + 158k \quad \text{et} \quad \acute{e}norme_k(x) \leftrightarrow x = s^{\alpha(k)-1}(0).}$$

Pour montrer l'égalité, il suffit de compter :

$$\begin{aligned} |\acute{e}norme_k(x)| &= |\text{triangle}_k(t, x, z, y)| + 2, \\ |\text{triangle}_0(t, x, z, y)| &= 7, \\ |\text{triangle}_{k+1}(t, x, z, y)| &= |\text{triangle}_k(t, x, z, y)| + (54 + 27 + 23 + 27 + 23 + 4) \end{aligned}$$

et de conclure. La démonstration de l'équivalence (dans l'algèbre des arbres infinis) fait l'objet de la prochaine sous-section.

3.2 Preuve de la deuxième partie de la propriété 3

Ecrivons $x\{f, k_1, \dots, k_m\}y$ pour signifier que x est un arbre dont la racine est étiquetée f et qu'il existe $i \in \{k_1, \dots, k_m\}$ tel que l'arbre y soit le i -ième fils de x . Convenons aussi que :

$$\begin{aligned} x\{f, k_1, \dots, k_m\}^0 y &\leftrightarrow x = y, \\ x\{f, k_1, \dots, k_m\}^{n+1} y &\leftrightarrow \exists u x\{f, k_1, \dots, k_m\}u \wedge u\{f, k_1, \dots, k_m\}^n y \end{aligned}$$

avec $n \geq 0$.

Compte tenu de la définition de $\text{énorme}_k(x)$, pour montrer la deuxième partie de la propriété 3 il suffit de montrer que dans l'algèbre des arbres on a la dernière des trois équivalences :

$$\begin{aligned} (\exists z \text{ triangle}_k(1, x, z, y)) &\leftrightarrow x\{f, 2, 3\}^{\alpha(k)-1} y \\ (\exists z \text{ triangle}_k(2, x, z, y)) &\leftrightarrow \bigvee_{i=0}^{\alpha(k)-1} x\{f, 2, 3\}^i y \\ (\exists z \text{ triangle}_k(3, x, z, y)) &\leftrightarrow x\{s, 1\}^{\alpha(k)-1} y \end{aligned} \quad (6)$$

Montrons par induction sur k que les trois équivalences sont vraies. Elles sont vraies pour $k = 0$. Supposons qu'elles soient vraies pour un certain $k \geq 0$ et montrons qu'elles sont vraies pour $k+1$. De (4) on déduit tout d'abord que

$$\text{triangle}_{k+1}(t, x, z, y) \leftrightarrow \left[\begin{array}{l} [\exists u_1 \exists u_2 z = f(x, u_1, u_2, y)] \\ \wedge \\ \left[\begin{array}{l} \forall y' \\ (\exists z' \text{ triangle}_k(1, z, z', y')) \rightarrow \\ \text{forme1}(y') \end{array} \right] \\ \wedge \\ \left[\begin{array}{l} \forall y' \\ (\exists z' \text{ triangle}_k(2, z, z', y')) \rightarrow \\ \left[\begin{array}{l} \exists u \exists v \text{ forme2}(u, y', v) \wedge \\ (t=1 \rightarrow \text{trans1}(u, v)) \wedge \\ (t=2 \rightarrow \text{trans2}(u, v)) \wedge \\ (t=3 \rightarrow \text{trans3}(u, v)) \end{array} \right] \end{array} \right] \end{array} \right]$$

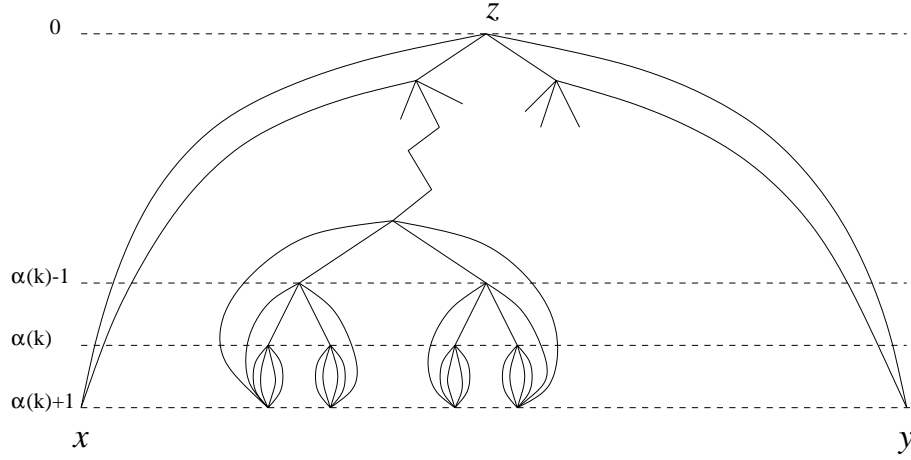
Du fait que l'on a supposé les équivalences (6) vraies pour k et en se servant des nouvelles notations, on a

$$\text{triangle}_{k+1}(t, x, z, y) \leftrightarrow \left[\begin{array}{l} [z\{f, 1\}x \wedge z\{f, 4\}y] \\ \wedge \\ \left[\begin{array}{l} \forall y' \\ z\{f, 2, 3\}^{\alpha(k)-1} y' \rightarrow \\ \text{forme1}(y') \end{array} \right] \\ \wedge \\ \left[\begin{array}{l} \forall y' \\ \left[\begin{array}{l} \bigvee_{i=0}^{\alpha(k)-1} z\{f, 2, 3\}^i y' \rightarrow \\ \left[\begin{array}{l} \exists u \exists v \text{ forme2}(u, y', v) \wedge \\ (t=1 \rightarrow u\{f, 2, 3\}v) \wedge \\ (t=2 \rightarrow u\{f, 2, 3\}v \vee u = v) \wedge \\ (t=3 \rightarrow u\{s, 1\}v) \end{array} \right] \end{array} \right] \end{array} \right]$$

du fait que le haut de l'arbre x satisfaisant $forme1(x)$ et le haut de l'arbre z satisfaisant $forme2(x, z, y)$ sont respectivement de la forme



le haut de l'arbre z satisfaisant $triangle(t, x, z, y)$ est de la forme



Il s'ensuit que

$$\exists z \text{ triangle}_{k+1}(t, x, z, y) \leftrightarrow \exists z \left[\begin{array}{l} [z\{f, 2\}^{\alpha(k)+1}x \wedge z\{f, 3\}^{\alpha(k)+1}y] \\ \wedge \\ \left[\bigwedge_{i=0}^{\alpha(k)} \left[\forall y' z\{f, 2, 3\}^i y' \rightarrow \right. \right. \\ \left. \left[\exists u \exists v \right. \right. \\ \left. \left. y'\{f, 2\}u \wedge y'\{f, 3\}v \right] \right] \right] \\ \wedge \\ \left[\forall y' \forall u \forall v \right. \\ \left. \left[z\{f, 2, 3\}^{\alpha(k)} y' \wedge \right. \right. \\ \left. \left. y'\{f, 2\}u \wedge y'\{f, 3\}v \right] \rightarrow u=v \right] \\ \wedge \\ \left[\bigwedge_{i=0}^{\alpha(k)-1} \left[\forall y' \forall u \forall v \forall u' \forall v' \right. \right. \\ \left. \left[z\{f, 2, 3\}^i y' \wedge \right. \right. \\ \left. \left. y'\{f, 2\}u' \wedge u'\{f, 3\}^{\alpha(k)-i} u \wedge \right. \right. \\ \left. \left. y'\{f, 3\}v' \wedge v'\{f, 2\}^{\alpha(k)-i} v \wedge \right. \right. \\ \left. \left. \left[\begin{array}{l} (t = 1 \rightarrow u\{f, 2, 3\}v) \wedge \\ (t = 2 \rightarrow u\{f, 2, 3\}v \vee u = v) \wedge \\ (t = 3 \rightarrow u\{s, 1\}v) \end{array} \right] \right] \right] \right] \end{array} \right]$$

Du fait que, dans un arbre binaire, le nombre de nœuds de profondeur n est égal à 2^n ,

$$\exists z \text{ triangle}_{k+1}(t, x, y, z) \leftrightarrow \exists u_1 \dots \exists u_{\alpha(k)} \left[\begin{array}{l} x = u_1 \wedge u_{\alpha(k+1)} = y \wedge \\ \left[\bigwedge_{i=1}^{\alpha(k+1)-1} \left[\begin{array}{l} (t=1 \rightarrow u_i \{f, 2, 3\} u_{i+1}) \wedge \\ (t=2 \rightarrow u_i \{f, 2, 3\} u_{i+1} \vee u_i = u_{i+1}) \wedge \\ (t=3 \rightarrow u_i \{s, 1\} u_{i+1}) \end{array} \right] \right] \end{array} \right]$$

On conclut qu'on a bien les équivalences (6) pour $k+1$, ce qui termine la démonstration.

3.3 Expression d'un programme logique effectuant une multiplication

Soit $\text{étape}(x, y)$ une formule faisant intervenir deux variables libres x et y . Si on modifie la formule $\text{triangle}_k(t, x, z, y)$ en posant

$$\text{trans3}(x, y) \stackrel{\text{def}}{=} x = y \vee \text{étape}(x, y)$$

et si on introduit la formule

$$\text{itération}_k(x, y) \stackrel{\text{def}}{=} \exists z \exists u \text{triangle}_k(3, x, z, u) \wedge \text{trans3}(u, y)$$

on a alors

$$\text{itération}_k(x, y) \leftrightarrow \bigvee_{n=0}^{\alpha(k)} (\exists u_0 \dots \exists u_n x = u_0 \wedge u_n = y \wedge \bigwedge_{i=1}^n \text{étape}(u_{i-1}, u_i)) \quad (7)$$

La relation binaire définie par itération est en quelque sorte une fermeture transitive *limitée* de la relation binaire définie par étape .

Soit T la théorie des arbres, c'est-à-dire un ensemble de propositions du premier ordre suffisant pour déduire toute propriété des arbres exprimable sous forme d'une proposition du premier ordre. D'après la programmation logique, la formule

$$\text{fois}(s^i(0), s^j(0), x),$$

dans la théorie

$$T \cup \left\{ \begin{array}{l} \forall i \forall j \forall k \forall k' \\ (\text{fois}(0, j, 0) \leftarrow \text{vrai}) \wedge \\ (\text{fois}(s(i), j, k') \leftarrow \text{fois}(i, j, k) \wedge \text{plus}(j, k, k')) \wedge \\ (\text{plus}(0, j, j) \leftarrow \text{vrai}) \wedge \\ (\text{plus}(s(i), j, s(k)) \leftarrow \text{plus}(i, j, k)) \wedge \end{array} \right.$$

est équivalente à

$$x = s^{i \times j}(0).$$

Compte tenu du fonctionnement des interpréteurs Prolog et d'après l'équivalence (7), il s'ensuit que, dans l'algèbre des arbres, la contrainte

$$\text{itération}_k(c(f(s^i(0), s^j(0), x), 0), 0)$$

avec

$$\text{étape}(x, y) \stackrel{\text{def}}{=} \left[\begin{array}{l} \exists i \exists j \exists k \exists k' \exists l \\ (x = c(f(0, j, 0), l) \wedge y = l) \\ (x = c(f(s(i), j, k'), l) \wedge y = c(f(i, j, k), c(p(j, k, k'), l))) \\ (x = c(p(0, j, j), l) \wedge y = l) \\ (x = c(p(s(i), j, s(k)), l) \wedge y = c(p(i, j, k), l)) \end{array} \right] \vee$$

est aussi équivalente à

$$x = s^{i \times j}(0)$$

mais à condition que $i(j+2)+1 \leq \alpha(k)$. Pour $k = 5$ on peut considérer que cette condition est quasi satisfaite. On dispose ainsi un moyen systématique de remplacer un programme logique par une contrainte dans les arbres.

3.4 Universalité versus complexité

A la place d'une machine Prolog on peut prendre une machine de Turing M , et exprimer par $\text{étape}(x, y)$ le fait que M puisse passer de la configuration x à la configuration y en exécutant une instruction. On en conclut que :

Propriété 4 *Le résultat de l'exécution d'une machine de Turing, exécutant au plus $\alpha(k)$ instructions, peut s'exprimer par une contrainte d'arbres, de dimension inférieure ou égale à un nombre proportionnel à k*

Ici encore en prenant $k = 5$ on pourra exprimer tout ce que l'ordinateur le plus puissant pourrait calculer. Les contraintes d'arbres ont donc un pouvoir d'expression quasi universel ce qui a pour conséquence que la complexité des algorithmes pour les résoudre ne peut qu'être très élevée. Examinons ce point plus en détails dans le cas de contraintes sans variables libres.

Assimilons un algorithme à une machine de Turing M dont l'exécution se termine quel que soit le mot $x \in V^*$ qui lui est fourni en entrée. La complexité de M est l'application de type $\mathbf{N} \rightarrow \mathbf{N}$:

$$n \mapsto \max \left\{ i \in \mathbf{N} \mid \begin{array}{l} \text{il existe } x \in V^*, \text{ avec } |x| = n, \text{ tel que } M \\ \text{exécute } i \text{ instructions, avec } x \text{ comme entrée.} \end{array} \right\}$$

Soit Φ_α un ensemble de fonctions croissantes de type $\mathbf{N} \rightarrow \mathbf{N}$ telles que

1. les fonctions de la forme $n \mapsto an + f(bn)$, avec $a \in \mathbf{N}$, $b \in \mathbf{N}$ et $f \in \Phi_\alpha$, appartiennent aussi à Φ_α ,
2. il existe un langage L , reconnaissable par une machine de Turing de complexité majorée par α , mais par aucune machine de Turing de complexité majorée par un élément de Φ_α .

Propriété 5 *Soit T une machine de Turing permettant de décider si une contrainte d'arbres sans variables libres est vraie. La complexité de T ne peut être majorée par un élément de Φ_α .*

Preuve. Supposons qu'il existe une telle machine T de complexité majorée par un élément f de Φ_α et montrons que nous aboutissons à une contradiction. Du fait que Φ_α n'est pas vide le langage $L \subseteq V^*$, intervenant au point 2 de la définition de Φ_α , existe. D'après la propriété 4, à tout mot $x \in V^*$, on peut alors associer une contrainte d'arbres p_x sans variables libres telle que

1. $x \in L$ si et seulement si p_x est vraie,
2. $|p_x| \leq b|x|$, pour une certaine constante $b \in \mathbf{N}$,
3. la transformation $x \mapsto p_x$ puisse être exécutée par une machine de Turing S de complexité majorée par $n \mapsto an$, où a est une constante. (Ce point pourrait être plus détaillé.)

En enchaînant les exécutions des machines S et T , on construit alors une machine M' qui reconnaît L et dont la complexité est majorée par $n \mapsto an + f(bn)$, une fonction qui par définition appartient à Φ_α . Il y a donc contradiction sur les propriétés de L , ce qui termine la preuve.

A condition de montrer que pour ensemble Φ_α on peut prendre l'ensemble des fonctions, de type $\mathbf{N} \rightarrow \mathbf{N}$, obtenues par composition finie des fonctions élémentaires $n \mapsto \text{cst}$, $+$, \times , $n \mapsto 2^n$, on retrouve le résultat de Sergei Vorobyov [14], mais à la façon de Pawel Mielniczuk [13] :

Propriété 6 *La complexité d'un algorithme, qui décide si une contrainte d'arbres sans variables libres est vraie, ne peut être majorée par une fonction obtenue par composition finie de fonctions élémentaires mentionnées ci-dessus.*

4 Discussions et conclusion

Les exemples que nous avons présentés ont mis en évidence l'apport des quantifications imbriquées et des opérateurs $\neg, \wedge, \vee, \rightarrow$ dans l'expressivité des contraintes d'arbres. Ils n'utilisent pas vraiment le fait que les arbres puissent être infinis et sont aussi corrects dans l'algèbre des arbres finis. Il serait donc intéressant de mettre en avant des exemples faisant intervenir des structures cycliques, telles que les automates à ensembles d'états finis, les grammaires hors contexte ou les λ -expressions, ainsi qu'il a été fait en [3, 7] dans le cadre de la programmation logique.

A la section 3.4 nous avons entrevu l'énorme complexité théorique d'un algorithme pour résoudre des contraintes d'arbres (sans variables libres). Nous avons pu cependant effectuer des benchmarks sur tous nos exemples. Les résultats sont résumés dans le tableau qui suit. Les temps sont donnés en milli-secondes :

k	$gagnant_k$ jeu 1	$gagnant_k$ jeu 2	$énorme_k$	$itération_k$ 1×1
0	0	0	0	-
1	0	150	0	-
2	10	360	10	70
3	10	610	230	-
4	20	840	-	-
5	30	1180	-	-
10	300	5 970	-	-
20	4 270	236 350	-	-
40	89 870	-	-	-
80	3 841 220	-	-	-

L'algorithme de résolution est programmé en C++ et les tests sont effectués sur un processeur Pentium II à 350Mhz, avec 512Mo de mémoire vive.

Il est remarquable que nous ayons pu calculer les positions k -gagnantes du jeu 1, avec $k = 80$, ce qui correspond à une formule faisant intervenir une imbrication alternée de plus de 160 quantificateurs. Il fallait s'attendre à ce nous ayons du mal à calculer l'arbre de $\alpha(k)$ noeuds au-delà de $k = 3$, puisque $\alpha(4)$ vaut déjà 65536. Pour la multiplication par itération k , nous n'avons pu aller au-delà de $k = 2$ et avons du nous contenter de calculer 1×1 !

Ces tests ont aussi permis d'enlever des doutes sur l'exactitude des formules compliquées de nos exemples, même si pour les rendre plus lisibles, nous avons introduit des prédicats pour nommer des sous-formules, en prenant soin de ne pas créer de définition circulaire. Dans une première phase, notre algorithme de résolution élimine ces prédicats intermédiaires.

Si on permet des définitions circulaires on obtient quelque chose qui ressemble à une « completion » généralisée d'un programme logique à la Keith Clark [2]. Notre algorithme de résolution peut aussi prendre en compte de telles définitions circulaires en retardant le plus possible leurs développements. Avec de la malchance l'algorithme ne termine pas, avec de la chance il termine et fournit obligatoirement une contrainte simplifiée sans prédicats intermédiaires.

Références

- [1] Benhamou F., P. Bouvier, A. Colmerauer, H. Garetta, B. Gilletta, J.L. Massat, G.A. Narboni, S. N'Dong, R. Pasero, J.F. Pique, Touraïvane, M. Van Caneghem et E. Vétillard. *Le manuel de Prolog IV*. PrologIA, Marseille, juin 1996.
- [2] Clark K.L., Negation as failure, dans *Logic and Databases*, édité par H. Gallaire et J. Minker, Plenum Press, New York, pp. 293–322, 1978.
- [3] Colmerauer A., Prolog and Infinite Trees, dans *Logic Programming*, K.L. Clark and S-A. Tarnlund editeurs, Academic Press, New York, pages 231–251, 1982.
- [4] Colmerauer A., Henry Kanoui et Michel Van Caneghem. Prolog, theoretical principles and current trends, dans *Technology and Science of Informatics*, North Oxford Academic, vol. 2, no 4, août 1983. Version anglaise de la revue *TSI*, AFCET-Bordas, où l'article est paru sous le titre : Prolog, bases théoriques et développements actuels.
- [5] Colmerauer A., Equations and Inequations on Finite and Infinite Trees, dans *Proceeding of the International Conference on Fifth Generation Computer Systems (FCGS-84)*, ICOT, Tokyo, pages 85–99, 1984.
- [6] Colmerauer A., An Introduction to Prolog III, *Communications of the ACM*, 33(7) : 68–90, 1990.
- [7] Coupet-Grimal S. et O. Ridoux, On the use of advanced logic programming features in computational linguistics. *The Journal of Logic Programming*, 24(1-2), pages 121–159.
- [8] Courcelle B., Fundamental Properties of Infinite Trees, *Theoretical Computer Science*, 25(2), pages 95–169, mars 1983.
- [9] Courcelle B., Equivalencies and Transformations of Regular Systems - Applications to Program Schemes and Grammars, *Theoretical Computer Science*, 42, pages 1–122, 1986.

- [10] Thi-Bich-Hanh Dao, Résolution de contraintes du premier ordre dans la théorie des arbres fini ou infinis, soumis à, *Journées Francophones de Programmation Logique et Programmation par Contraintes* (JFPLC'2000), Marseille, juin 2000, actes à paraître chez Hermes Science Publications.
- [11] Huet G., *Résolution d'équations dans les langages d'ordre 1, 2, ..., ω* . Thèse d'Etat, Université Paris 7, 1976.
- [12] Maher M.J., *Complete Axiomatization of the Algebra of Finite, Rational and Infinite Trees*, Technical report, IBM - T.J.Watson Research Center, 1988.
- [13] Mielniczuk P., Basic Theory of Feature Trees, <http://www.tcs.uni.wroc.pl/~mielni>, soumis à *Journal of Symbolic Computation*.
- [14] Vorobyov S., An Improved Lower Bound for the Elementary Theories of Trees, *Proceeding of the 13th International Conference on Automated Deduction (CADE'96)*. Springer Lecture Notes in Artificial Intelligence, vol 1104, pages 275-287, New Brunswick, NJ, juillet/aout, 1996.