Expressiveness of Full First Order Constraints in the Algebra of Finite or Infinite Trees

Alain Colmerauer, Thi-Bich-Hanh Dao

Expressiveness of full first order constraints in the algebra of finite or infinite trees

Alain Colmerauer and Thi-Bich-Hanh Dao

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

Abstract. We are interested in the expressiveness of constraints represented by general first order formulae, with equality as unique relational symbol and functional symbols taken from an infinite set F. The chosen domain is the set of trees whose nodes, in possibly infinite number, are labeled by elements of F. The operation linked to each element f of F is the mapping $(a_1, \ldots, a_n) \mapsto b$, where b is the tree whose initial node is labeled f and whose sequence of daughters is $a_1, \ldots, a_n$.
We first consider constraints involving long alternated sequences of quantifiers $\exists\forall\exists\forall\ldots$. We show how to express winning positions of two-partners games with such constraints and apply our results to two examples.
We then construct a family of strongly expressive constraints, inspired by a constructive proof of a complexity result by Pawel Mielniczuk. This family involves the huge number $\alpha(k)$, obtained by evaluating top down a power tower of 2's, of height k. With elements of this family, of sizes at most proportional to k, we define a finite tree having $\alpha(k)$ nodes, and we express the result of a Prolog machine executing at most $\alpha(k)$ instructions.
By replacing the Prolog machine by a Turing machine we rediscover the following result of Sergei Vorobyov: the complexity of an algorithm, deciding whether a constraint without free variables is true, cannot be bounded above by a function obtained by finite composition of elementary functions including exponentiation.
Finally, taking advantage of the fact that we have at our disposal an algorithm for solving such constraints in all their generality, we produce a set of benchmarks for separating feasible examples from purely speculative ones. Among others we solve constraints involving alternated sequences of more than 160 quantifiers.

1 Introduction

The algebra of (possibly) infinite trees plays a fundamental act in computer science: it is a model for data structures, program schemes and program executions. As early as 1976, Gérard Huet proposed an algorithm for unifying infinite terms, that is solving equations in that algebra [11]. Bruno Courcelle has studied the properties of infinite trees in the scope of recursive program schemes [8, 9]. Alain Colmerauer has described the execution of Prolog II, III and IV programs in terms of solving equations and disequations in that algebra [4–6, 1]. Michael

Maher has introduced and justified a complete theory of the algebra of infinite trees [12]. Among others, he has shown that in this theory, and thus in the algebra of infinite trees, any first order formula is equivalent to a Boolean combination of conjunctions of equations (partially or totally) existentially quantified. Sergei Vorobyov has shown that the complexity of an algorithm, deciding whether a formula without free variables is true in that theory, cannot be bounded above, by a function obtained by finite composition of elementary functions, including exponentiation [14]. Pawel Mielniczuk has shown a similar result in the theory of feature trees, but with a more constructive method, which has inspired some of our examples [13].

We have recently developed an algorithm for solving general first order constraints in the algebra of infinite trees [10]. The purpose of this paper is not the presentation of this algorithm, but of examples, first imagined as tests, then extended to show the expressiveness of such general constrains. The paper is organized as follows.

(1) We end this first section by making clear the notions of tree algebra and first order constraints in that algebra.

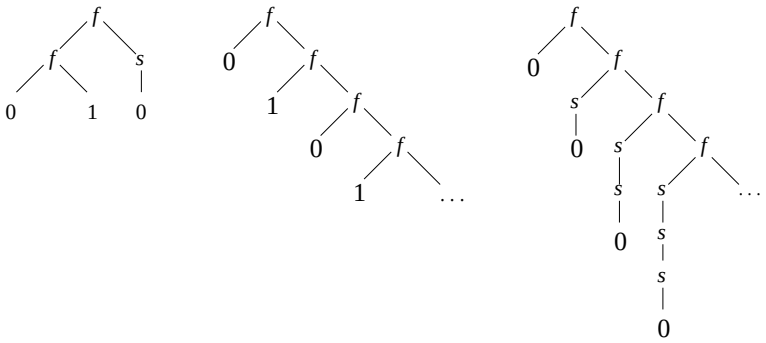
(2) In the second section we consider constraints involving long alternated sequences of quantifiers $\exists\forall\exists\forall\ldots$. We show how to express winning positions of two-partners games with such constraints and apply our results to two examples.

(3) In the third section, we investigate the most expressive family of constraints we know. It involves the truly huge number $\alpha(k)$, obtained by evaluating top down a tower of powers of 2's, of height k. With elements of this family, of sizes at most proportional to k, we define a finite tree having $\alpha(k)$ nodes, and we express the result of a Prolog machine executing at most $\alpha(k)$. By replacing the Prolog machine by a Turing machine we rediscover the complexity result of Sergei Vorobyov mentioned at the beginning of this section. This part has been strongly influenced by the work of Pawel Mielniczuk [13].

(4) We conclude by discussions and benchmarks separating the feasible examples from the purely speculative ones.

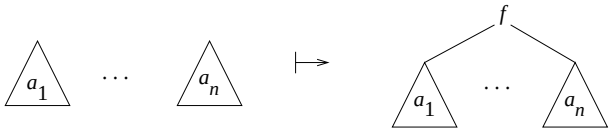
1.1 The algebra of infinite trees

Trees are well known objects in the computer science world. Here are some of them:



Their nodes are labeled by the symbols $0, 1, s, f$, of respective arities $0, 0, 1, 2$, taken from a set F of functional symbols, which we assume to be infinite. Note that the first tree is the only one having a finite set of nodes, but that the second one has still a finite set of (patterns of) subtrees. We denote by $\mathbf{A}$ the set of all trees [1] constructed on F.

We introduce in $\mathbf{A}$ a set of construction operations [2], one for each element $f \in F$ which is the mappings $(a_1, \ldots, a_n) \mapsto b$, where n is the arity of f and b the tree whose initial node is labeled f and the sequence of daughters is $(a_1, \ldots, a_n)$ and which be schematized as



We thus obtain the *algebra of infinite trees* constructed on F, which we denote by $(\mathbf{A}, F)$.

1.2 Tree constraints

We are interested in the expressiveness of constraints represented by general first order formulae, with equality as unique relational symbol and functional symbols taken from an infinite set F. These *tree constraints* are of one of the 9 forms:

$$s = t,\ \mathit{true},\ \mathit{false},\ \neg(p),\ (p \wedge q),\ (p \vee q),\ (p \to q),\ \exists x\, p,\ \forall x\, p,$$

where p and q are shorter tree constraints, x a variable taken from an infinite set and s, t terms, that are expressions of one of the forms

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

where $n \geq 0$, $f \in F$, with arity n, and the t_i's are shorter terms.

The variables represent elements of the set $\mathbf{A}$ of trees constructed on F and the functional symbols f are interpreted as construction operations in the algebra of infinite trees $(\mathbf{A}, F)$. Thus a constraint without free variables is either true or false and a constraint $p(x_1, \ldots, x_n)$ with n free variables x_i establish an n-ary relation in the set of trees.

---

[1] More precisely we define first a *node* to be a word constructed on the set of strictly positive integers. A *tree* a, constructed on F, is then a mapping of type $E \to F$, where E is a non-empty set of nodes, each one $i_1 \ldots i_k$ (with $k \geq 0$) satisfying the two conditions: (1) if $k > 0$ then $i_1 \ldots i_{k-1} \in E$, (2) if the arity of $a(i_1 \ldots i_k)$ is n, then the set of nodes of E of the form $i_1 \ldots i_k i_{k+1}$ is obtained by giving to i_{k+1} the values $1, \ldots, n$.

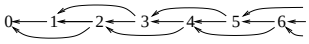
[2] In fact, the *construction* operation linked to the n-ary symbol f of F is the mapping $(a_1, \ldots, a_n) \mapsto b$, where the a_i's are any trees and b is the tree defined as follows from the a_i's and their set of nodes E_i's: the set E of nodes of a is $\{\varepsilon\} \cup \{ix \mid x \in E_i \text{ and } i \in 1..n\}$ and, for each $x \in E$, if $x = \varepsilon$, then $a(x) = f$ and if x is of the form iy, with i being an integer, $a(x) = a_i(y)$.

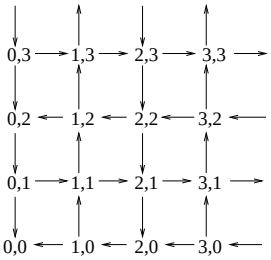
2 Long nesting of alternated quantifiers

We first introduce the notions of k-winning and k-losing position in any two-partners games and in two examples. We show how to express, in any domain, the set of k-winning positions by a constraint. We end the section by expressing the k-winning positions of the two examples by tree constraints involving an alternated embedding of $2k$ quantifiers.

2.1 Winning positions in a two-partners game

Let (V, E) be a directed graph, with V a set of vertices and $E \subseteq V \times V$ a set of edges. The sets V and E may be empty and the elements of V are also called *positions*. We consider a two-partners game which, given an initial position x_0, consists, one after another, in choosing a position x_1 such that $(x_0, x_1) \in E$, then a position x_2 such that $(x_1, x_2) \in E$, then a position x_3 such that $(x_2, x_3) \in E$ and so on... The first one who cannot play any more has lost and the other one has won. For example the two following infinite graphs correspond to the two following games:





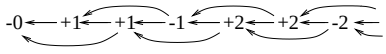
Game 1 A non-negative integer i is given and, one after another, each partner subtracts 1 or 2 from i, but keeping i non-negative. The first person who cannot play any more has lost.

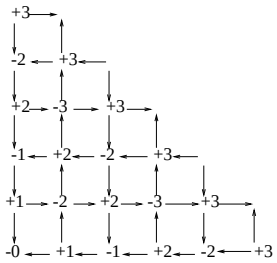
Game 2 An ordered pair (i, j) of non-negative integers is given and, one after another, each partner chooses one of the integers i, j. Depending on the fact that the chosen integer u is odd or even, he then increases or decreases the other integer v by 1, but keeping v non-negative. The first person who cannot play any more has lost.

Let $x \in V$ be any vertex of the directed graph (V, E) and suppose that it is the turn of person A to play. The position x is said to be *k-winning* if, no matter the way the other person B plays, it is always possible for A to win in making at most k moves. The position x is said to be *k-losing* if, no matter the way A plays, B can always force A to lose and to play at most k moves.

Consider the two preceding graphs and mark with $+k$ the positions which are k-winning and with $-k$ the positions which are k-losing, with each time k being as small as possible. Vertex 0 of the first graph and vertex $(0, 0)$ of the second one being the only 0-losing positions, are marked with -0. Starting from

the vertices marked with -0 and following the arrows in reverse direction, we find successively the set of vertices to be marked by $+1$, then -1, then $+2$, then -2, then $+3$, then -3, and so on. We get





and convince ourselves that the set of k-winning positions of game 1 is

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

and of game 2

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

where $\mathbf{N}$ is the set of non-negative integers.

2.2 Expressing k-winning positions by a constraint

Let $\mathbf{D}$ be a *domain*, that is a non-empty set and let $G = (V, E)$ the graph of a two-partners game, with $V \subseteq \mathbf{D}$. We will express the k-winning positions of G by a constraint in $\mathbf{D}$ involving an embedding $\exists\forall\exists\ldots$ of $2k$ alternated quantifiers.

Let us introduce in $\mathbf{D}$ the properties *move*, $winning_k$ et $losing_k$, defined by

$$\begin{array}{ll} move(x,y) & \leftrightarrow (x,y) \in E, \\ winning_k(x) & \leftrightarrow x \text{ is a } k\text{-winning position of } G, \\ losing_k(x) & \leftrightarrow x \text{ is a } k\text{-losing position of } G. \end{array} \tag{1}$$

In $\mathbf{D}$ we then have the equivalences, for all $k \geq 0$:

$$\begin{array}{ll} winning_0(x) & \leftrightarrow false, \\ winning_{k+1}(x) & \leftrightarrow \exists y\ move(x,y) \wedge losing_k(y), \\ losing_k(x) & \leftrightarrow \forall y\ move(x,y) \rightarrow winning_k(y). \end{array} \tag{2}$$

Contrary to what we may believe, it follows that we have:

$$winning_k(x) \rightarrow winning_{k+1}(x), \qquad losing_k(x) \rightarrow losing_{k+1}(x).$$

Indeed, from the first and the last equivalence of (2) we conclude that these implications hold for $k = 0$ and, if we assume that they hold for a certain $k \geq 0$, from the last two equivalences in (2) we conclude that they also hold for $k+1$.

From (3) we deduce an explicit formulation of $winning_k$, for all $k \geq 0$:

$$winning_k(x) \leftrightarrow \left[\begin{array}{l} \exists y\, move(x,y) \wedge \neg(\\ \exists x\, move(y,x) \wedge \neg(\\ \exists y\, move(x,y) \wedge \neg(\\ \exists x\, move(y,x) \wedge \neg(\\ \ldots \\ \exists y\, move(x,y) \wedge \neg(\\ \exists x\, move(y,x) \wedge \neg(\\ false \qquad \underbrace{)\ldots)}_{2k} \end{array}\right] \tag{3}$$

where of course all the quantifiers apply on elements of $\mathbf{D}$. By moving down the negations, we thus get an embedding of $2k$ alternated quantifiers.

In equivalence (3) it is possible to use a more general definition of *move* than the one given in (1). We first remark, that for any non-negative k, the following property holds:

Property 1 Let three directed graphs be of the form $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G = (V_1 \cup V_2, E_1 \cup E_2)$. The graphs G_1 and G have the same set of k-winning positions, if both:

1. the sets of vertices V_1 and V_2 are disjoint,
2. for all $x \in V_2$, there exists $y \in V_2$ with $(x, y) \in E_2$.

Indeed, from the first condition it follows that E_1 and E_2 are disjoint and thus that the set of k-winning positions of G is the union of the set of k-winning positions of G_1 with the set of k-winning positions of G_2. This last set is empty because of the second condition.

It follows that:

Property 2 (Generalized *move* relation) *Equivalence* (3) *holds also for any move relation obeying to the two conditions:*

1. *for all* $x \in V$ *and* $y \in V$, $move(x,y) \leftrightarrow (x,y) \in E$,
2. *for all* $x \in \boldsymbol{D}{-}V$ *there exists* $y \in \boldsymbol{D}{-}V$ *such that* $move(x,y)$.

2.3 Formalizing game 1 in the algebra of infinite trees

We now reconsider game 1 introduced in section 2.1. As domain $\mathbf{D}$ we take the set $\mathbf{A}$ of trees constructed on a set F of functional symbols including among others the symbols $0, s$, of respective arities $0, 1$. We code the vertices i of the game graph by the trees[3] $s^i(0)$. Let $G = (V, E)$ be the graph obtained this way.

As generalized relation *move* we then can take in the algebra of infinite trees:

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

[3] Of course, $s^0(0) = 0$ and $s^{i+1}(0) = s(s^i(0))$.

and according to property 2 the set of k-winning positions of game 1 is the set of solutions in x of the constraint $winning_k(x)$ defined in (3).

For example, with $k = 1$ the constraint $winning_k(x)$ is equivalent to

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

and with $k = 2$ to

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

2.4 Formalizing game 2 in the algebra of infinite trees

We also reconsider game 2 introduced in section 2.1. As domain **D** we take the set **A** of trees constructed on a set F of functional symbols including among others the symbols $0, f, g, c$, of respective arities $0, 1, 1, 2$. We code the vertices (i, j) of the game graph by the trees $c(\overline{i}, \overline{j})$ with $\overline{i} = (fg)^{\frac{i}{2}}(0)$ if i is even, and $\overline{i} = g(\overline{i-1})$ if i is odd[4]. Let $G = (V, E)$ be the graph obtained this way.

The perspicacious reader will convince himself that, as generalized relation *move* , we can take in the algebra of infinite trees:

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

with

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

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

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

According to property 2, the set of k-winning positions of game 2 is the set of solutions in x of the constraint $winning_k(x)$ defined in (3).

For example, with $k = 1$ the constraint $winning_k(x)$ is equivalent to

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

and with $k = 2$ to

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

[4] Of course, $(fg)^0(x) = x$ and $(fg)^{i+1}(x) = (fg)^i(f(g(x)))$.

3 Quasi-universality of tree constraints

After all these quantifiers, we move to constraints, which are so expressive that their solving becomes quasi-undecidable.

3.1 Defining a huge finite tree by a constraint

We set $\alpha(k) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$, with k occurrences of 2. More precisely we take

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

with $k \geq 0$. The function α increases in a stunning way, since $\alpha(0) = 1$, $\alpha(1) = 2$, $\alpha(2) = 4$, $\alpha(3) = 16$, $\alpha(4) = 65536$ and $\alpha(5) = 2^{65536}$. Thus $\alpha(5)$ is greater than 10^{20000}, a number probably much greater than the number of atoms of the universe or the number of nanoseconds which elapsed since its creation!

We suppose that the set $\mathbf{A}$ of trees is constructed on a set F of functional symbols including among others the symbols $0, 1, 2, 3, s, f$, of respective arities $0, 0, 0, 0, 1, 4$. For $k \geq 0$ let us introduce the constraint:

$$huge_k(x) \stackrel{\text{def}}{=} \exists z \; triangle_k(3, x, z, 0)$$

with still for $k \geq 0$,

$$triangle_0(t, x, z, y) \stackrel{\text{def}}{=} z = x \wedge z = y$$

$$triangle_{k+1}(t, x, z, y) \stackrel{\text{def}}{=} \left[\begin{array}{l} [\exists u_1 \, \exists u_2 \, z = f(x, u_1, u_2, y)] \\ \wedge \\ \left[\begin{array}{l} \forall t' \, \forall y' \, \forall z' \\ \left[\begin{array}{l} (t' = 1 \vee t' = 2) \wedge \\ triangle_k(t', z, z', y') \end{array}\right] \rightarrow \\ \left[\begin{array}{l} (t' = 1 \wedge form1(y')) \vee \\ \left(t' = 2 \wedge \left[\begin{array}{l} \exists u \, \exists v \; form2(u, y', v) \wedge \\ (t = 1 \rightarrow trans1(u, v)) \wedge \\ (t = 2 \rightarrow trans2(u, v)) \wedge \\ (t = 3 \rightarrow trans3(u, v)) \end{array}\right]\right) \end{array}\right] \end{array}\right] \end{array}\right] \qquad (4)$$

and

$$form1(x) \stackrel{\text{def}}{=} \exists u_1 \ldots \exists u_4 \; x = f(u_1, f(u_2, u_2, u_2, u_2), f(u_3, u_3, u_3, u_3), u_4)$$

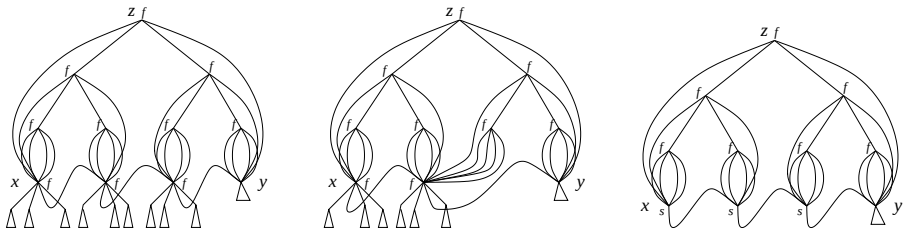
$$form2(x, z, y) \stackrel{\text{def}}{=} \exists u_1 \ldots \exists u_6 \; z = f(u_1, f(u_1, u_2, u_3, x), f(y, u_4, u_5, u_6), u_6)$$

$$trans1(x, y) \stackrel{\text{def}}{=} \exists u_1 \ldots \exists u_4 \; x = f(u_1, u_2, u_3, u_4) \wedge (y = u_2 \vee y = u_3)$$

$$trans2(x, y) \stackrel{\text{def}}{=} trans1(x, y) \vee x = y$$

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

To give a feeling of what $triangle_k(t,x,z,y)$ means, here are three trees x, z, y such that $triangle_2(t,x,z,y)$ with $t=1$, $t=2$ and $t=3$, from left to right:



Let us agree that the size $|p|$ of a constraint p, is the number of occurrences of all symbols except parentheses and commas. (Constraints could be written in infix notation.) We then have the double property:

Property 3 (small constraint, big tree)

$$\boxed{|huge_k(x)| = 9 + 158k \qquad \text{and} \qquad huge_k(x) \;\leftrightarrow\; x = s^{\alpha(k)-1}(0).}$$

To prove the equality, it is sufficient to count:

$$\begin{array}{ll} |huge_k(x)| & = |triangle_k(t,x,z,y)| + 2, \\ |triangle_0(t,x,z,y)| & = 7, \\ |triangle_{k+1}(t,x,z,y)| & = |triangle_k(t,x,z,y)| + (54+27+23+27+23+4) \end{array}$$

and to conclude. The proof of the equivalence (in the algebra of infinite trees) is the subject of next subsection.

3.2 Proof of the second part of property 3

We write $x\{f,k_1,...,k_m\}y$ for expressing that x is a tree whose initial node is labeled f and that there exists $i \in \{k_1,\ldots,k_m\}$ such that tree y is the ith daughter of x. We also agree that:

$$\begin{array}{l} x\{f,k_1,...,k_m\}^0 y \quad \leftrightarrow x = y, \\ x\{f,k_1,...,k_m\}^{n+1} y \leftrightarrow \exists u\; x\{f,k_1,...,k_m\}u \wedge u\{f,k_1,...,k_m\}^n y \end{array}$$

with $n \geq 0$.

Given the definition of $huge_k(x)$, to show the second part of property 3 it is sufficient to show that, in the algebra of infinite trees, the last of the three following equivalences holds:

$$\begin{array}{l} (\exists z\; triangle_k(1,x,z,y)) \leftrightarrow x\{f,2,3\}^{\alpha(k)-1} y \\ (\exists z\; triangle_k(2,x,z,y)) \leftrightarrow \bigvee_{i=0}^{\alpha(k)-1} x\{f,2,3\}^i y \\ (\exists z\; triangle_k(3,x,z,y)) \leftrightarrow x\{s,1\}^{\alpha(k)-1} y \end{array} \qquad (5)$$

Let us show by induction on k that the three equivalences hold. They hold for $k = 0$. Let us assume that they hold for a certain $k \geq 0$ and let us proof that they hold for $k+1$. Definition (4) can be reformulated as

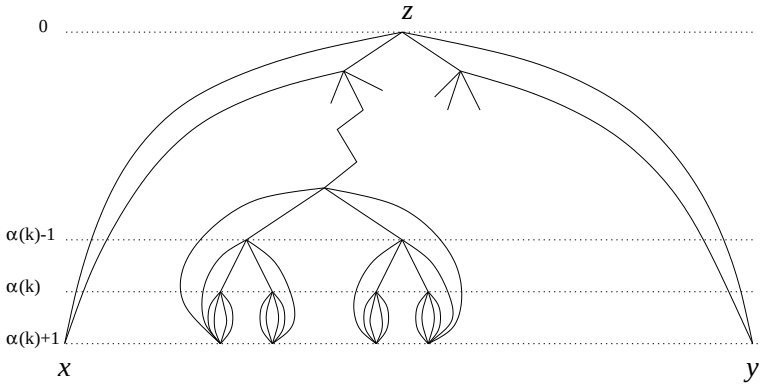
$$
\begin{array}{l}
\mathit{triangle}_{k+1}(t,x,z,y) \;\leftrightarrow \\
\left[\begin{array}{l}
[\exists u_1\, \exists u_2\, z = f(x,u_1,u_2,y)] \\
\wedge \\
\left[\begin{array}{l}
\forall y' \\
(\exists z'\; \mathit{triangle}_k(1,z,z',y')) \rightarrow \\
\mathit{form1}(y')
\end{array}\right]
\end{array}\right]
\wedge
\left[\begin{array}{l}
\forall y' \\
(\exists z'\; \mathit{triangle}_k(2,z,z',y')) \rightarrow \\
\left[\begin{array}{l}
\exists u\, \exists v\; \mathit{form2}(u,y',v) \wedge \\
(t{=}1 \rightarrow \mathit{trans1}(u,v)) \wedge \\
(t{=}2 \rightarrow \mathit{trans2}(u,v)) \wedge \\
(t{=}3 \rightarrow \mathit{trans3}(u,v))
\end{array}\right]
\end{array}\right]
\end{array}
$$

Taking into account our assumptions and using our new notations, we get

$$
\begin{array}{l}
\mathit{triangle}_{k+1}(t,x,z,y) \;\leftrightarrow \\
\left[\begin{array}{l}
[z\{f,1\}x \wedge z\{f,4\}y] \\
\wedge \\
\left[\begin{array}{l}
\forall y' \\
z\{f,2,3\}^{\alpha(k)-1}y' \rightarrow \\
\mathit{form1}(y')
\end{array}\right]
\end{array}\right]
\wedge
\left[\begin{array}{l}
\forall y' \\
[\bigvee_{i=0}^{\alpha(k)-1} z\{f,2,3\}^i y'] \rightarrow \\
\left[\begin{array}{l}
\exists u\, \exists v\; \mathit{form2}(u,y',v) \wedge \\
(t=1 \rightarrow u\{f,2,3\}v) \wedge \\
(t=2 \rightarrow u\{f,2,3\}v \vee u = v) \wedge \\
(t=3 \rightarrow u\{s,1\}v)
\end{array}\right]
\end{array}\right]
\end{array}
$$

Since the top of a tree x satisfying $\mathit{form1}(x)$ and the top of a tree z satisfying $\mathit{form2}(x,z,y)$ are respectively of the form

the top of a tree z satisfying $\mathit{triangle}(t,x,z,y)$ is of the form



It follows that

$$\exists z\, \mathit{triangle}_{k+1}(t,x,z,y) \;\leftrightarrow$$

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

Since, in a binary tree the number of nodes of depth n is equal to 2^n,

$$\exists z\, \mathit{triangle}_{k+1}(t,x,y,z) \;\leftrightarrow$$

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

We conclude that the equivalences (5) hold for $k+1$, which ends the proof.

3.3 Expressing a logic program performing a multiplication

Let $\mathit{step}(x,y)$ be a formula involving two free variables x and y. If we modify formula $\mathit{triangle}_k(t,x,z,y)$ by setting

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

and if we introduce the formula

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

we then have

$$\mathit{iteration}_k(x,y) \;\leftrightarrow\; \bigvee_{n=0}^{\alpha(k)} (\exists u_0 \ldots \exists u_n\; x = u_0 \wedge u_n = y \wedge \bigwedge_{i=1}^{n} \mathit{step}(u_{i-1}, u_i)) \qquad (6)$$

The binary relation defined by *iteration* is in some way a *bounded* transitive closure of the relation defined by *step*.

Let T be the theory of trees, that is a set of first order propositions which entails all the properties of the algebra of infinite trees which can be expressed as first order propositions. According to logic programming, the formula

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

in the theory

$$T \cup \left\{ \begin{array}{l} \forall i\, \forall j\, \forall k\, \forall k' \\ (times\,(0,j,0) \leftarrow true) \quad \wedge \\ (times\,(s(i),j,k') \leftarrow times\,(i,j,k) \wedge plus\,(j,k,k')) \wedge \\ (plus\,(0,j,j) \leftarrow true) \quad \wedge \\ (plus\,(s(i),j,s(k)) \leftarrow plus\,(i,j,k)) \quad \wedge \end{array} \right\}$$

is equivalent to

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

Given the way a Prolog interpreter works and given equivalence (6), the constraint

$$iteration_{\,k}(c(f(s^i(0), s^j(0), x),\ 0),\ 0)$$

with

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

is equivalent in the algebra of infinite trees to

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

provided that $i(j+2)+1 \leq \alpha(k)$. For $k = 5$ we can consider that this restriction is quasi-satisfied. Thus we have a systematic way to replace a logic Horn clauses program by a tree constraint.

3.4 Universality versus complexity

Instead of a Prolog machine we can take a Turing machine M, and express by $step\,(x,y)$ the fact that M may move from configuration x to configuration y by executing one instruction. We then conclude that:

Property 4 *The result produced by a Turing machine, executing at most $\alpha(k)$ instructions, can be expressed by a tree constraint of size less or equal to a number proportional to k.*

Here also, by taking $k = 5$ it is possible to express any result that the most powerful computer could compute. Thus the tree constraints have a quasi-universal expressiveness and the complexity of the algorithms for solving them must be very high. Let us examine this point in more details and in the case of constraints without free variables.

Let us consider an algorithm as a Turing machine M whose execution terminates for all word $x \in V^\star$ given as input. The complexity of M is the mapping of type $\mathbf{N} \to \mathbf{N}$:

$$n \mapsto \max \left\{ i \in \mathbf{N} \;\middle|\; \begin{array}{l} \text{there exists } x \in V^\star, \text{ with } |x| = n, \text{ such that } M \\ \text{executes } i \text{ instructions, with } x \text{ as input.} \end{array} \right\}$$

Let Φ_α be a set of non-decreasing functions of type $\mathbf{N} \to \mathbf{N}$ such that

1. the functions of the form $n \mapsto an + f(bn)$, with $a \in \mathbf{N}$, $b \in \mathbf{N}$ and $f \in \Phi_\alpha$, belong also to Φ_α,
2. there exists a language L, recognizable by a Turing machine of complexity bounded above by α, but by no Turing machine of complexity bounded above by an element of Φ_α.

Property 5 *Let T be a Turing machine deciding whether a tree constraint without free variables holds. The complexity of T can not be bounded above by an element of Φ_α.*

Proof. Let us suppose that there exists such a machine T with a complexity bounded above by an element f of Φ_α and let us show that this leads us to a contradiction. Since Φ_α is not empty, the language $L \subseteq V^\star$ in part 2 of the definition of Φ_α, exists. According to property 4, to each word $x \in V^\star$, corresponds a tree constraints p_x, without free variables, such that

1. $x \in L$ if and only if p_x holds,
2. $|p_x| \leq b|x|$, for some constant $b \in \mathbf{N}$,
3. the transformation $x \mapsto p_x$ can be performed by a Turing machine S with a complexity bounded above by $n \mapsto an$, for some constant $a \in \mathbf{N}$. (This point could be more detailed.)

By linking together the executions of machines S and T, we then build a machine M' which recognizes L and whose complexity is bounded above by $n \mapsto an + f(bn)$, a function which by definition belongs to Φ_α. Thus there is a contradiction about the properties of L, which ends the proof.

Under the condition of having shown that, as set Φ_α, we can take the set of functions, of type $\mathbf{N} \to \mathbf{N}$, obtained by finite composition of the elementary functions: $n \mapsto \text{cst}$, $+$, $\times$, $n \mapsto 2^n$, we rediscover the result of Sergei Vorobyov [14], but in the spirit of Pawel Mielniczuk [13]:

Property 6 *The complexity of an algorithm, which decides whether a tree constraint, without free variables, holds, can not be bounded above by a function obtained by finite composition of elementary functions mentioned above.*

4 Discussions and conclusion

The presented examples show the contribution of embedded quantifiers and operators $\neg, \wedge, \vee, \rightarrow$ in the expressiveness of tree constraints. They do not really use the fact that the trees may be infinite and are also valid in the algebra of finite trees. It would be interesting to give examples involving infinite trees for coding cyclic structures like finite states automata, context-free grammars or λ-expressions, as it has been done in [3, 7] in the frame of logic programming.

At subsection 3.4 we have provided a glimpse of the huge theoretical complexity of an algorithm for solving tree constraints. However, we have succeeded

in producing benchmarks on all our examples [10]. The results are summarized in the following table, with CPU times given in milliseconds:

k	$winning_k$ game 1	$winning_k$ game 2	$huge_k$	$iteration_k$ 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	-	-	-

The algorithm is programmed in C++ and the benchmarks are performed on a 350Mhz Pentium II processor, with 512Mb of RAM.

It must be noted that we were able to compute the k-winning positions of game 1 with $k = 80$, which corresponds to a formula involving an alternated embedding of more than 160 quantifiers. We were prepared to experience difficulties in computing the tree of $\alpha(k)$ nodes, beyond $k = 3$, since $\alpha(4)$ is already 65536. With respect to multiplication by $iteration_k$, we were unable to succeed beyond $k = 2$ and had to satisfy ourselves with the computation of 1×1!

These test have also removed some of our doubts about the correctness of the complicated formulae of our examples, even if, for readability, we have introduced predicates for naming sub-formulae. Of course the definitions of theses predicates are supposed not to be circular and the solver unfold and eliminates them in a first step.

If circular definitions are accepted then our constraints look like generalized completions of logic programs [2]. Our solver can also take into account such possibly circular definitions by delaying their unfoldings as much as possible. With bad luck the solver does not terminate, with luck it terminates and generates obligatory a simplified constraint without intermediary predicates.

References

1. Benhamou F., P. Bouvier, A. Colmerauer, H. Garetta, B. Giletta, J.L. Massat, G.A. Narboni, S. N'Dong, R. Pasero, J.F. Pique, Touraïvane, M. Van Caneghem and E. Vétillard, *Le manuel de Prolog IV*. PrologIA, Marseille, June 1996.
2. Clark K.L., Negation as failure, in *Logic and Databases*, edited by H. Gallaire and J. Minker, Plenum Press, New York, pp. 293–322, 1978.
3. Colmerauer A., Prolog and Infinite Trees, in *Logic Programming,* K.L. Clark and S.A.. Tarnlund editors, Academic Press, New York, pp. 231–251, 1982.
4. Colmerauer A., Henry Kanoui andt Michel Van Caneghem, Prolog, theoretical principles and current trends, in *Technology and Science of Informatics*, North

Oxford Academic, vol. 2, no 4, August 1983. English version of the journal *TSI*, AFCET-Bordas, where the paper appears under the title: Prolog, bases théoriques et développements actuels.

5. Colmerauer A., Equations and Inequations on Finite and Infinite Trees, in *Proceeding of the International Conference on Fifth Generation Computer Systems (FCGS-84)*, ICOT, Tokyo, pp. 85–99, 1984.
6. Colmerauer A., An Introduction to Prolog III, *Communications of the ACM*, 33(7) : 68–90, 1990.
7. Coupet-Grimal S. and 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), pp. 95–169, March 1983.
9. Courcelle B., Equivalences and Transformations of Regular Systems - Applications to Program Schemes and Grammars, *Theoretical Computer Science*, 42, pp. 1–122, 1986.
10. Dao T.B.H., Résolution de contraintes du premier ordre dans la théorie des arbres fini ou infinis, *Journées Francophones de Programmation Logique et Programmation par Contraintes* (JFPLC'2000), Marseille, June 2000, proceedings to be published by Hermes Science Publications.
11. Huet G., *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$., 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, submitted to *Journal of Symbolic Computation*, also available at `http://www.tcs.uni.wroc.pl/~mielni`.
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, pp. 275-287, New Brunswick, NJ, July/August, 1996.