From Specifications to Programs: A Fork-Algebraic Approach to Bridge the Gap

*Gabriel A. Baum, †Marcelo F. Frias, †Armando M. Haeberer,
*Pablo E. Martínez López

*Universidad Nacional de La Plata,
LIFIA, Departamento de Informática,
C.C.11, Correo Central, 1900, La Plata,
Provincia de Buenos Aires, República Argentina.
e-mail: {gbaum, fidel}@info.unlp.edu.ar.
†Laboratório de Métodos Formais,
Departamento de Informática,
Pontifícia Universidade Católica do Rio de Janeiro,
Rua Marquês de São Vicente 225, 22453-900,
Rio de Janeiro, RJ, Brazil.
e-mail: {mfrias, armando}@inf.puc-rio.br.

Abstract. The development of programs from first-order specifications has as its main difficulty that of dealing with universal quantifiers. This work is focused in that point, i.e., in the construction of programs whose specifications involve universal quantifiers. This task is performed within a relational calculus based on *fork algebras*. The fact that first-order theories can be translated into equational theories in abstract fork algebras suggests that such work can be accomplished in a satisfactory way. Furthermore, the fact that these abstract algebras are representable guarantees that all properties valid in the standard models are captured by the axiomatization given for them, allowing the reasoning formalism to be shifted back and forth between any model and the abstract algebra. In order to cope with universal quantifiers, a new algebraic operation — *relational implication* — is introduced. This operation is shown to have deep significance in the relational statement of first-order expressions involving universal quantifiers. Several algebraic properties of the relational implication are stated showing its usefulness in program calculation. Finally, a non-trivial example of derivation is given to asses the merits of the relational implication as an specification tool, and also in calculation steps, where its algebraic properties are clearly appropriate as transformation rules.

1 Introduction

The last few years have witnessed a renewed interest of the computing science community in relational programming calculi. This interest is mainly due to the advantage of relational calculi for dealing with non-determinism. At the same time, the postulated advantage of relational calculi for describing, in early steps of specification construction, *what-to-do* instead of *how-to-do*, gives relevance

to such calculi. Even while the first advantage cannot be denied (for example when comparing relational calculi against functional calculi), the second claimed advantage must be thoroughly discussed, since in relational frameworks, everything that can be said can be said easily, but not everything can be said. Thus relational calculi may fall for short when dealing with the problem of software specification.

This paper is divided in three main parts. The first part introduces fork algebras and some well known results about their expressiveness. The second one deals with the fork algebras representation problem, and its relevance in the process of program construction from specifications. Finally, in the third part — containing the main results regarding formal program construction —, the relational implication is introduced and some algebraic rules are provided. Also, a case study is presented to show the suitability of relational implication in dealing with problems whose natural specifications involve universal quantifiers. While relational implication was already introduced in [7], where a simple case study is also given, it is here where for the first time a thorough study about it is carried on. We will present properties of the relational implication which result essential when deriving algorithms from specifications containing universal quantifiers.

2 Fork Algebras: Models, Axiomatization and Expressiveness

Fork Algebras arose in computer science as an equational calculus for formal program specification and development within a relational framework [1, 3, 7, 8]; they showed however to have inherence also in the fields of algebra and logic, as shown in [5, 6, 11, 12, 13]. In this section we will present the models that motivated the usage of fork algebras as a calculus for program specification and development. Also given is an abstract axiomatization, which constitutes a finite base for those models as proved in Theo. 6. Finally, some known results on the expressiveness of fork algebras are mentioned, and a discussion on the implications of these results in the field of program specification is carried on.

Proper fork algebras (PFAs for short) are algebras of relations [2, 15] extended with a new operator — called *fork* — devised to deal with complex objects with a tree–like structure. These "trees" may have branches of infinite height. Even though this may seem a drawback, it can be useful for modeling infinite processes or infinite computations as it is done in [3]. Furthermore, this tree–like structure allows handling of as many variables as necessary when representing first–order formulas as fork algebra terms.

In order to define PFA's[1], we will first define the class of powerset $\star$PFAs by

[1] It is important to remark that PFA's are quasi–concrete structures since, as was pointed out by Andréka and Németi in a private communication, concrete structures must be fully characterized by their underlying domain, something that does not happen with proper fork algebras because of the (hidden) operation $\star$.

Definition 1. A powerset $\star$PFA is a two sorted structure with domains $\mathcal{P}(V)$ and U

$$< \mathcal{P}(V), U, \cup, \cap, ', \emptyset, V, |, Id, ^{\smile}, \underline{\nabla}, \star >$$

such that

1. V is an equivalence relation with domain U,
2. $|$, Id and $^{\smile}$ stand respectively for composition between binary relations, the diagonal relation on U and the converse of binary relations, thus making the reduct $< \mathcal{P}(V), \cup, \cap, ', \emptyset, V, |, Id, ^{\smile} >$ an algebra of binary relations,
3. $\star : U \times U \to U$ is an injective function when its domain is restricted to V,
4. whenever xVy and xVz, also $xV \star (y, z)$,
5. $R \underline{\nabla} S = \{< x, \star(y, z) >: xRy \text{ and } xSz\}$.

Definition 2. The class of PFA's is defined as $\mathbf{S}\,\mathbf{Rd}\,[\star\text{PFA}]$, where $\mathbf{Rd}$ takes reducts to the similarity type $< \cup, \cap, ', \emptyset, V, |, Id, ^{\smile}, \underline{\nabla} >$ and $\mathbf{S}$ takes subalgebras.

It is important to notice that, in the characterization of proper fork algebras, we use variables ranging over two different classes of entities. Some variables range over relations (like R and S in Def. 1) and others range over individuals (like x, y, z also in Def. 1), thus leading to a two sorted theory. In the same way that dummy variables ranging over individuals are avoided in functional languages, we are going to present an abstract characterization of proper fork algebras as a first-order class whose variables range only over relations. This relationship between proper fork algebras and the class of abstract fork algebras — defined below —, is made precise in Sect. 3.

Definition 3. An abstract fork algebra (AFA for short) is an algebraic structure

$$< R, +, \bullet, ^{-}, 0, \infty, ;, 1, ^{\smile}, \nabla >$$

satisfying the following set of axioms

1. Axioms stating that $< R, +, \bullet, ^{-}, 0, \infty, ;, 1, ^{\smile} >$ is a relation algebra[2] where $< R, +, \bullet, ^{-}, 0, \infty >$ is the Boolean reduct, $< R, ;, 1 >$ is the monoid reduct, and $^{\smile}$ stands for relational converse,
2. $r \nabla s = (r; (1 \nabla \infty)) \bullet (s; (\infty \nabla 1))$,
3. $(r \nabla s) ; (t \nabla q)^{\smile} = (r; t^{\smile}) \bullet (s; q^{\smile})$,
4. $(1 \nabla \infty)^{\smile} \nabla (\infty \nabla 1)^{\smile} \preceq 1$, where $\preceq$ is the partial ordering induced by the Boolean structure.

Next we will introduce some non-fundamental fork algebra operation that will be used in the specification and derivation process.

Definition 4. Let the operations π, ρ and $\otimes$ be defined by

1. $\pi = (1 \nabla \infty)^{\smile}$ (*first projection*)

[2] Equational axiomatic systems for relation algebras are given in [2, 15].

2. $\rho = (\infty \nabla 1)^\smile$ (*second projection*)
3. $R \otimes S = (\pi;R) \nabla (\rho;S)$ (*cross product*)
4. $2 = 1 \nabla 1$ (*equality filter*)

Operations π and ρ stand for projecting relations (see Fig.1) regarding the underlying pair formation operation $\star$. Since 1 stands for the diagonal relation, the relation $2^\smile$ acts like a filter, filtering those pairs $< x \star y >$ such that $x \neq y$.

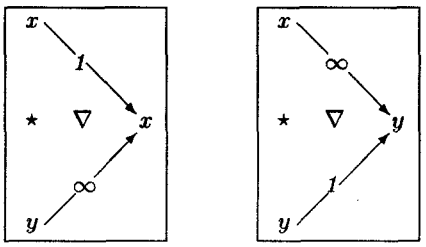

Fig. 1. The projections π and ρ.

2.1 The Expressiveness of Abstract Fork Algebras

In order to describe shortly the relationship existing between first order logic with equality and fork algebras, we can say that first order theories can be *interpreted* as equational theories in fork algebras. More formally, let L be a first order language. Let us denote by $< A, L' >$ the extension of the similarity type of abstract fork algebras with a sequence of constant symbols whose names are sequentially assigned from the symbols in L. Then the following theorem holds.

Theorem 5. *There exists a recursively defined mapping T, translating formulas in L into equations in $< A, L' >$, satisfying*

$$\Gamma \vdash \alpha \qquad \Longleftrightarrow \qquad \{T(\gamma) = \infty : \gamma \in \Gamma\} \vdash_\nabla T(\alpha) = \infty.$$

The symbol $\vdash_\nabla$ in Th.5 stands for provability in fork algebras, i.e., proofs are made in equational logic and the extralogical axioms defining the fork algebra operators are assumed to hold.

The result shown in Th.5 was already known for other algebraic systems closely related with fork algebras, as quasi–projective relation algebras and pairing relation algebras. The work on the interpretability of first–order theories into quasi–projective relation algebras was extensively developed by Tarski and Givant in [15], while the version for pairing relation algebras was developed by Maddux in [10].

Theorem 5 has an strong application in program specification within the framework of abstract fork algebras. If we use as our primitive specification

language some first–order theories — assumption more than reasonable since first–order languages are simple and expressive formal languages — Th.5 guarantees that by applying the mapping T to a first–order specification of a given problem, we obtain a faithful abstract relational specification of it.

Regarding the issue presented in Sect. 1 on the suitability of relation algebras for expressing *what–to–do* instead of *how–to–do*, it must be noted that the answer is not so straight. Although relations have some natural operations as converse or complement which are powerful in the specification construction process (both allow to specify non–functional problems), the expressiveness of the involved relational language cannot be ignored. It is, for example, a well known fact that the language of relation algebras — as presented in [15] — has a very limited expressive power [9], and thus is not adequate for complex program specification. On the other hand, the language of abstract fork algebras has been shown to be expressive enough as to cope with the specification and derivation process, as shown in [7].

3 The Representation Problem for Fork Algebras

Fork algebras' expressiveness theorem establish that the specifications and the properties of the application domain, which may be expressed in first–order logic, can also be expressed in the *equational* theory of abstract fork algebras. However, this expressibility is insufficient for one to formulate, within the theory, many of the fundamental aspects of the program construction process. The process of program construction by calculations within relational calculi requires more than the possibility to express the specification of requirements. It also requires the calculus to provide heuristics that guide the syntactic manipulations in the development process. If every AFA would be representable as a PFA, then we could look at elements from AFAs as binary relations, thus inheriting all the heuristic power of binary relations into the abstract calculus. Fortunately, as will be shown in the next theorem, AFAs are representable.

Theorem 6. *Every abstract fork algebra is isomorphic to some proper fork algebra.*

A first proof of this theorem for complete and atomistic AFAs is given in [4]. Later on Gyuris [6] and Frias et al. [5] proved a representation theorem for the whole class. Strictly speaking, since PFAs are not a concrete class because of the hidden operation $\star$, this can be considered as a weak–representation theorem[3]. Anyway, the theorem provides all the machinery we need for the process of program construction. A strong representation theorem for fork algebras was proved by Németi in [12], although in non–well founded set theories.

The representation theorem may seem of interest only for theoreticians, but it has a great impact in program specification and development within the framework of fork algebras. An immediate corollary of the theorem is that

[3] This was pointed out by Andréka and Németi in a private communication.

$Cn(\mathsf{PFA}) \subseteq Cn(\mathsf{AFA})$[4], and thus any first–order property valid in the standard models is reflected in the abstract ones. This simple fact, along with the suitability of proper fork algebras for reasoning about programs, makes the pair <abstract, proper> fork algebras a framework with particular heuristic power not shared by some other calculi, either relational or functional.

4 Calculating Algorithms from First–Order Specifications Involving Universal Quantifiers

In the course of this section, we will introduce the *relational implication*, a slight modification of the right residual, which is shown to be very adequate for representing first–order formulas involving universal quantifiers. Furthermore, relational implication has a nice representation in the standard model which, together with the discussion carried on in the last paragraph of Sect. 3, leads to an abstract operation with deep significance for the process of program specification and development.

Finally, we will present a problem whose first–order specification involves universal quantification. We will obtain from it a relational specification in terms of the relational implication. The main steps of a smooth derivation of a recursive specification will be given, showing the adequacy of the relational implication in the task of software specification and construction.

After some experiences with problems whose first–order specification involves universal quantification, the formula

$$\phi(x,\ y) := (\forall z)(x\ R\ z \ \Rightarrow\ y\ S\ z)$$

(or some minor variants of it) recurrently appears. It is used, for example, for specifying minimization and sorting problems [7] — two classical case studies — and in this paper it will be used in the specification of a non trivial problem about binary trees.

It is not difficult to see that the formula above is equivalent to the formula

$$\Psi(x,y) := x\ \overline{R\ ;\ \overline{S^{\smile}}}\ y,$$

thus, in our abstract framework we can define the relational implication of relations R and S by

$$R{\rightarrow}S := \overline{R\ ;\ \overline{S^{\smile}}}.$$

In the standard models, the relational implication can be understood as x is related with y via $R{\rightarrow}S$ if and only if the image of x by R is contained in the image of y by S — see Fig. 2.

Since the relation algebraic characterization of "$\rightarrow$" is hard to manipulate — complement doesn't behave nicely over composition —, we will present some properties of "$\rightarrow$" that allow us to avoid using its definition in derivations.

[4] Cn denotes the set of first–order valid formulas in the class.

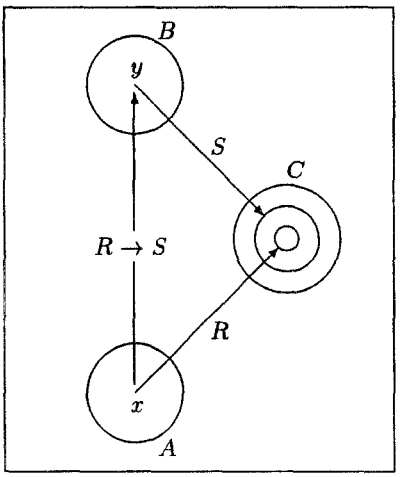

Fig. 2. The *relational implication.*

Besides of some easy properties, such as $((P+Q)\rightarrow R) = (P\rightarrow R) \bullet (Q\rightarrow R)$ and $(P\rightarrow(Q\bullet R)) = (P\rightarrow Q) + (P\rightarrow R)$, — which follow directly from the definition — we will present some more elaborated properties, leading to recursive relational expressions for computing the relational implication.

Proposition 7. *For any relations P, Q and R, we have*

$$\begin{aligned} &Dom\,(P+Q);((P+Q)\rightarrow R) = \\ &\quad Dom\,(P);\left((P\rightarrow R)\bullet\left(Dom\,(Q);(Q\rightarrow R)+\left(\overline{Dom\,(Q)}\bullet 1\right);\infty\right)\right) \\ &\qquad\qquad + \\ &\quad Dom\,(Q);\left((Q\rightarrow R)\bullet\left(Dom\,(P);(P\rightarrow R)+\left(\overline{Dom\,(P)}\bullet 1\right);\infty\right)\right). \end{aligned}$$

Proposition 8. *If A is a functional relation, then* $Dom\,(A);(A\rightarrow P) = A;P^{\smile}$.

Proposition 9. *If B is a functional relation, then* $Dom\,(B);((B;P)\rightarrow Q) = B;(P\rightarrow Q)$.

Proposition 10. *Let A and B be functional relations, and let* $P = A+B;P$ *and* $T = P\rightarrow Q$. *Moreover, let also* $Dom\,(P) = Dom\,(A)+Dom\,(B)$ *and* $Dom\,(A)\bullet Dom\,(B) = 0$. *Then*

$$Dom\,(P);T = A;Q^{\smile}+B;T.$$

Proposition 11. *Let A, B and C be functional relations, and let* $P = A+B;P+C;P$ *and* $T = P\rightarrow Q$. *Moreover, let us suppose that* $Dom\,(P) = Dom\,(A)+Dom\,(B)+Dom\,(C)$ *and* $Dom\,(A)\bullet Dom\,(B) = 0$, $Dom\,(A)\bullet Dom\,(C) = 0$, *and* $Dom\,(B)\bullet Dom\,(C) = 0$. *Then*

$$Dom\,(P);T = A;Q^{\smile}+B;T+C;T.$$

¿From Props. 10 and 11 we can see that the recursiveness of the relation P allows to obtain a recursive specification for the relation T.

Some other nice properties of the relational implication are[5]

Proposition 12. *Let* $T = P \rightarrow Q$. *If* $P = \inf\{X : X = A;B + X\}$, A *and* B *are functional relations, and* $Dom(A) \bullet Dom(B) = 0$, *then*

$$T = A^{\smile};Q + B;T.$$

Proposition 13. *Let* $T = P \rightarrow Q$. *If* $P = \inf\{X : X = A;B + X;C + X\}$, A, B *and* C *are functional relations, and* $Dom(A) \bullet Dom(B) = Dom(A) \bullet Dom(C) = Dom(B) \bullet Dom(C) = 0$, *then*

$$T = A^{\smile};Q + B;T + C;T.$$

In this section we present a problem that is easily specified using first–order logic, and whose relational specification relies on the relational implication. We then proceed with the main steps of a smooth derivation that leads us to a recursive expression for our problem.

The problem we present as example is stated as

> "Given a Binary Tree T without repeated nodes, and two elements x and y belonging to T, find the *Minimum Common Ancestor* of x and y, i.e., that node in T which is the closest ancestor to both x and y".

Let us take a relation HA (abbreviating has_ancestor), which is meant to give — given a tree T and an element x in T — the ancestors of x in T. A formal specification of a relation MCA capturing the problem in the language of first–order logic is given by

$$[T, x, y]\mathsf{MCA}\ a \iff [T, x]\mathsf{HA}\ a \wedge [T, y]\mathsf{HA}\ a \wedge (\forall z)(([T, x]\mathsf{HA}\ z \wedge [T, y]\mathsf{HA}\ z) \Rightarrow [T, a]\mathsf{HA}\ z).$$

A first–order specification of HA is given by the following formula:

$$[T, x]\ \mathsf{HA}\ \ a\ \textit{iff}\ (\exists T')(T' \sqsubseteq T \ \wedge\ T'\ \mathsf{root}\ a \ \wedge\ x\ \mathsf{in}\ T'),$$

where the relations HA, in, root and $\sqsubseteq$ are the basic relations on the underlying data type Binary–trees. The relation $\sqsubseteq$ is meant to give all the trees that contain a given tree T' as subtree; we will also use the converse of $\sqsubseteq$, noted by $\sqsupseteq$. The relation in relates a given tree with its elements and the relation root gives the root of the tree.

The specification in first–order logic is translated into an equation in AFA almost directly using the relational implication. If we define the relation CA (abbreviating common_anc) by

$$\mathsf{CA} = \left(\left(\begin{matrix} 1 \\ \otimes \\ \pi \end{matrix}\right);\mathsf{HA}\right) \bullet \left(\left(\begin{matrix} 1 \\ \otimes \\ \rho \end{matrix}\right);\mathsf{HA}\right), \tag{1}$$

[5] Props. 12 and 13 were proved by G. Schmidt and M. Frias at the Workshop of the Relational Methods in Computer Science Group, held in Rio de Janeiro, August 1994

then the following equation provides a relational specification for our problem: $\mathsf{MCA} = \mathsf{MCA}^{\circ};\rho$, where

$$\mathsf{MCA}^{\circ} = \begin{pmatrix} \pi \\ \nabla \\ \mathsf{CA} \end{pmatrix} \bullet (\mathsf{CA} \rightarrow \mathsf{HA}). \tag{2}$$

We also have an equation specifying the relation HA, which is obtained by pattern matching between its first–order specification and the definition of the relational operators in the elementary theory of binary relations [14]:

$$\mathsf{HA} = \begin{pmatrix} \sqsupseteq \\ \otimes \\ 1 \end{pmatrix}; \begin{pmatrix} \pi;\mathsf{root} \\ \nabla \\ \begin{pmatrix} \mathsf{in} \\ \otimes \\ 1 \end{pmatrix}; 2 \end{pmatrix}; \pi. \tag{3}$$

¿From this relational specification, unfolding a recursive definition for $\sqsupseteq$, making simple manipulations in fork algebras, and folding HA, we obtain the following recursive equation[6].

$$\mathsf{HA} = \begin{pmatrix} 1_{T_1} \\ \otimes \\ 1 \end{pmatrix}; \begin{pmatrix} \mathsf{root} \\ \otimes \\ 1 \end{pmatrix}; \breve{2} + \begin{pmatrix} 1_{T>1} \\ \otimes \\ 1 \end{pmatrix}; \begin{pmatrix} \pi;\mathsf{root} \\ \nabla \\ \begin{pmatrix} \mathsf{in} \\ \otimes \\ 1 \end{pmatrix}; 2 \end{pmatrix}; \pi + \begin{pmatrix} 1_{T>1} \\ \otimes \\ 1 \end{pmatrix}; \begin{pmatrix} \mathsf{right+left} \\ \otimes \\ 1 \end{pmatrix}; \mathsf{HA} \tag{4}$$

¿From the recursive specification for HA, we can proceed with the derivation of CA. By unfolding Eq. 4 in Eq. 1, making some fork algebra manipulation, and then folding CA, we finally obtain:

$$\begin{aligned} \mathsf{CA} = {} & \begin{pmatrix} 1_{T_1};\mathsf{root} \\ \otimes \\ \breve{2} \end{pmatrix}; \breve{2} + \\ & \begin{pmatrix} 1_{T>1} \\ \otimes \\ 1 \end{pmatrix}; \underbrace{\left(Dom\left(\begin{pmatrix} \mathsf{in} \\ \otimes \\ \pi \end{pmatrix}; \breve{2} \right) \bullet Dom\left(\begin{pmatrix} \mathsf{in} \\ \otimes \\ \rho \end{pmatrix}; \breve{2} \right) \right)}_{\phi}; \pi;\mathsf{root} + \\ & \begin{pmatrix} 1_{T>1} \\ \otimes \\ 1 \end{pmatrix}; \left(\begin{pmatrix} \mathsf{right} \\ \otimes \\ 1 \end{pmatrix} + \begin{pmatrix} \mathsf{left} \\ \otimes \\ 1 \end{pmatrix} \right); \mathsf{CA} \end{aligned} \tag{5}$$

[6] 1_{T_1} stands for the relation $\{< x, x >: \text{heigth}(x) = 1\}$, and $1_{T>1}$ stands for the relation $\{< x, x >: \text{heigth}(x) > 1\}$. Given a binary tree, right and left give respectivelly its right and left subtree.

Calling A the first term, B the second term, $C = (C_1+C_2)$ the non-recursive part of the third term and P to CA, Eq. 5 has the shape:

$$P = A + B + (C_1+C_2);P \tag{6}$$

Now, unfolding Eq. 6 on Eq. 2, and distributing fork over join, we have

$$MCA^{\circ} = \left(\begin{pmatrix}\pi\\ \nabla\\ A\end{pmatrix} + \begin{pmatrix}\pi\\ \nabla\\ B\end{pmatrix} + \begin{pmatrix}\pi\\ \nabla\\ (C_1+C_2);P\end{pmatrix}\right) \bullet ((A + B + (C_1+C_2);P)\rightarrow HA). \tag{7}$$

Applying Props.7, 8, 9, and making elementary manipulations in fork algebras, the term on the right hand side of Eq.7 equals

$$\begin{pmatrix}\pi\\ \nabla\\ A\end{pmatrix} \bullet (A;HA^{\smile}) \tag{I}$$

$$+ \begin{pmatrix}\pi\\ \nabla\\ B\end{pmatrix} \bullet (B;HA^{\smile}) \bullet (Dom\,(C_1;P);C_1;(P\rightarrow HA)) \tag{II}$$

$$+ \begin{pmatrix}\pi\\ \nabla\\ B\end{pmatrix} \bullet (B;HA^{\smile}) \bullet (Dom\,(C_2;P);C_2;(P\rightarrow HA)) \tag{III}$$

$$+ \begin{pmatrix}\pi\\ \nabla\\ B\end{pmatrix} \bullet (B;HA^{\smile}) \bullet \left(Dom\,(B);\left(\overline{Dom\,((C_1+C_2);P)}\bullet 1\right);\infty\right) \tag{IV}$$

$$+ \begin{pmatrix}\pi\\ \nabla\\ (C_1+C_2);P\end{pmatrix} \bullet (B;HA^{\smile}) \bullet (Dom\,(C_1;P);C_1;(P\rightarrow HA)) \tag{V}$$

$$+ \begin{pmatrix}\pi\\ \nabla\\ (C_1+C_2);P\end{pmatrix} \bullet (B;HA^{\smile}) \bullet (Dom\,(C_2;P);C_2;(P\rightarrow HA)) \tag{VI}$$

$$+ \begin{pmatrix}\pi\\ \nabla\\ (C_1+C_2);P\end{pmatrix} \bullet (B;HA^{\smile}) \bullet \left(Dom\,(B);\left(\overline{Dom\,((C_1+C_2);P)}\bullet 1\right);\infty\right). \tag{VII}$$

Since[7] $\pi\nabla A \preceq A;HA^{\smile}$, $\pi\nabla B \preceq B;HA^{\smile}$, $\pi\nabla(C_1+C_2);P \leq B;HA^{\smile}$, and II, III and VII equal *0*, we obtain:

$$MCA = \begin{pmatrix}\pi\\ \nabla\\ A\end{pmatrix};\rho \; + \left(\begin{pmatrix}\pi\\ \nabla\\ B\end{pmatrix} \bullet \left(Dom\,(B);\left(\overline{Dom\,((C_1+C_2);P)}\bullet 1\right);\infty\right)\right);\rho \tag{a}$$

[7] These properties were proved algebraically. The reader may convince himself of their validity by thinking about the standard models. This is totally valid because of the representation theorem.

$$+ \left(\left(\begin{array}{c} \pi \\ \nabla \\ (C_1+C_2);P \end{array} \right) \bullet (Dom\,(C_1;P);C_1;(P \rightarrow HA)) \right) ; \rho \qquad (b)$$

$$+ \left(\left(\begin{array}{c} \pi \\ \nabla \\ (C_1+C_2);P \end{array} \right) \bullet (Dom\,(C_2;P);C_2;(P \rightarrow HA)) \right) ; \rho. \qquad (c)$$

¿From (a) we obtain the base case of the recursive specification for MCA, while (b) and (c) lead to the recursive parts of the algorithm. We finally obtain (replacing A, B and C by their definitions),

$$\begin{aligned}
\mathsf{MCA} = & \left(\begin{array}{c} 1_{T_1};\mathsf{root} \\ \otimes \\ \smile \\ 2 \end{array} \right) ; \breve{2} \\
& + Dom \left(\begin{array}{c} Dom\left(\left(\begin{array}{c} \mathsf{right} \\ \otimes \\ 1 \end{array} \right) ;\mathsf{CA} \right) \bullet 1 \\ \hline \nabla \\ \hline Dom\left(\left(\begin{array}{c} \mathsf{left} \\ \otimes \\ 1 \end{array} \right) ;\mathsf{CA} \right) \bullet 1 \end{array} \right) ; \left(Dom\left(\left(\begin{array}{c} \mathsf{in} \\ \otimes \\ \pi \end{array} \right) ; \breve{2} \right) \bullet Dom\left(\left(\begin{array}{c} \mathsf{in} \\ \otimes \\ \rho \end{array} \right) ; \breve{2} \right) \right) ;\pi;\mathsf{root} \\
& + Dom\left(\left(\begin{array}{c} 1_{T>1} \\ \otimes \\ 1 \end{array} \right) ; \left(\begin{array}{c} \mathsf{right} \\ \otimes \\ 1 \end{array} \right) ;\mathsf{CA} \right) ; \left(\begin{array}{c} \mathsf{right} \\ \otimes \\ 1 \end{array} \right) ;\mathsf{MCA} \\
& + Dom\left(\left(\begin{array}{c} 1_{T>1} \\ \otimes \\ 1 \end{array} \right) ; \left(\begin{array}{c} \mathsf{left} \\ \otimes \\ 1 \end{array} \right) ;\mathsf{CA} \right) ; \left(\begin{array}{c} \mathsf{left} \\ \otimes \\ 1 \end{array} \right) ;\mathsf{MCA}
\end{aligned} \qquad (8)$$

It is important to notice that the terms involving domains are tests for *if–then–else*–like statements. Hence, Eq. 8 leads to the following function:

$$\begin{aligned}
&\mathsf{MCA(t, x, y)} \\
&\quad = \mathsf{root(t)} && , \textit{if } \mathsf{height(t)} = 1 \ \wedge \ \mathsf{x} = \mathsf{y} = \mathsf{root(t)} \\
&\quad = \mathsf{root(t)} && , \textit{if } \{\text{“ x } \textit{and} \text{ y } \textit{are not in the same subtree”}\} \\
&\quad = \mathsf{MCA(right(t), x, y)} && , \textit{if } \{\text{“ x } \textit{and} \text{ y } \textit{are in the right subtree”}\} \\
&\quad = \mathsf{MCA(left(t), x, y)} && , \textit{if } \{\text{“ x } \textit{and} \text{ y } \textit{are in the left subtree”}\}
\end{aligned}$$

5 Conclusions

We have discussed three relevant concepts about fork algebras, namely, the applications of the expressiveness and representability theorems to program development, and also the heuristic power of the relational implication in “breaking” recursively, universally quantified expressions.

With respect to the derivation itself, it is important to note that it was developed by strict calculation from the axioms both of fork algebra and the particular abstract data type Binary–trees. In other words, the only means we have used to

introduce semantics along the calculation were Binary–trees formally expressed properties. At no point we have used non formal knowledge about the discourse domain.

References

1. Berghammer, R., Haeberer, A.M., Schmidt, G., and Veloso, P.A.S,. *Comparing Two Different Approaches to Products in Abstract Relation Algebras*, in Proceedings of the Third International Conference on Algebraic Methodology and Software Technology, AMAST '93, Springer Verlag, 1993, 167–176.
2. Chin, L.H. and Tarski, A., *Distributive and Modular Laws in the Arithmetic of Relation Algebras*, in University of California Publications in Mathematics. University of California, 1951, 341–384.
3. Frias, M.F., Aguayo N.G. and Novak B., *Development of Graph Algorithms with Fork Algebras*, in Proceedings of the XIX Latinamerican Conference on Informatics, 1993, 529–554.
4. Frias, M.F., Baum, G.A., Haeberer, A.M. and Veloso, P.A.S., *Fork Algebras are Representable*, in Bulletin of the Section of Logic, University of Łódź, (24)2, 1995, pp.64–75.
5. Frias, M.F., Haeberer, A.M. and Veloso, P.A.S., *A Finite Axiomatization for Fork Algebras*, to appear in Journal of the IGPL, 1996.
6. Gyuris, V., *A Short Proof for Representability of Fork Algebras*, Journal of the IGPL, vol 3, N.5, 1995, pp.791–796.
7. Haeberer, A.M., Baum, G.A. and Schmidt G., *On the Smooth Calculation of Relational Recursive Expressions out of First–Order Non–Constructive Specifications Involving Quantifiers*, in Proceedings of the Intl. Conference on Formal Methods in Programming and Their Applications, LNCS 735, Springer Verlag, 1993, 281–298.
8. Haeberer, A.M. and Veloso, P.A.S., *Partial Relations for Program Derivation: Adequacy, Inevitability and Expressiveness*, in Constructing Programs from Specifications – Proceedings of the IFIP TC2 Working Conference on Constructing Programs from Specifications. North Holland., 1991, 319–371.
9. Löwenheim, L., *Über Möglichkeiten im Relativkalkül*, Math. Ann. vol. 76, 1915, 447–470.
10. Maddux, R., *Finitary Algebraic Logic*, Zeitschr. f. math. Logik und Grundlagen d. Math., vol. 35, 1989, 321–332.
11. Mikulás, S., Sain, I., Simon, A., *Complexity of Equational Theory of Relational Algebras with Projection Elements.* Bulletin of the Section of Logic, Vol.21, N.3, 103–111, University of Łódź, October.1992.
12. Németi I., *Strong Representability of Fork Algebras, a Set Theoretic Foundation*, to appear in Journal of the IGPL, 1996.
13. Sain, I. and Németi, I., *Fork Algebras in Usual as well as in Non–well–founded Set Theories*, preprint of the Mathematical Institute of the Hungarian Academy of Sciences, 1994.
14. Tarski, A., *On the Calculus of Relations*, Journal of Symbolic Logic, vol. 6, 1941, 73–89.
15. Tarski, A. and Givant, S.,*A Formalization of Set Theory without Variables*, A.M.S. Coll. Pub., vol. 41, 1987.