How to say Greedy in Fork Algebras

Marcelo F. Frias* Gabriel A. Baum†
Esteban de la Canal†

* Departamento de Computación
F.C.E.yN., Universidad de Buenos Aires
mfrias@sol.info.unlp.edu.ar

* † L.I.F.I.A.
Facultad de Informática
Universidad Nacional de La Plata
CC 11,1900 - La Plata, Buenos Aires, Argentina
Tel/Fax: +54 221 4228252
URL: http://www-lifia.info.unlp.edu.ar/
{gbaum,steve}@sol.info.unlp.edu.ar

Abstract

Because of their expressive power, binary relations are widely used in program specification and development within formal calculi. The existence of a finite equational axiomatization for algebras of binary relations with a fork operation guarantees that the heuristic power coming from binary relations is captured inside an abstract equational calculus.

In this paper we show how to express the greedy program design strategy into the first order theory of fork algebras.

1 Introduction

In the field of program specification and development within programming calculi, relational calculi are gaining more interest with the passing of time. As an example of this, calculi formerly based only in functions have been extended with relational operators [5, 18]. The reason to do so, is that even though calculi based on functions have shown to be fruitful for program development from *functional* specifications [3, 4], finding such specifications requires still a big effort, since the bridge between the problem and its functional specification may be difficult to be crossed. On the other hand, relational frameworks allow to define new operators adequate for relations but not for functions.

From the previous remarks, the process of program construction within relational calculi can be represented — in a simplified vision — by the diagram in Fig. 1.

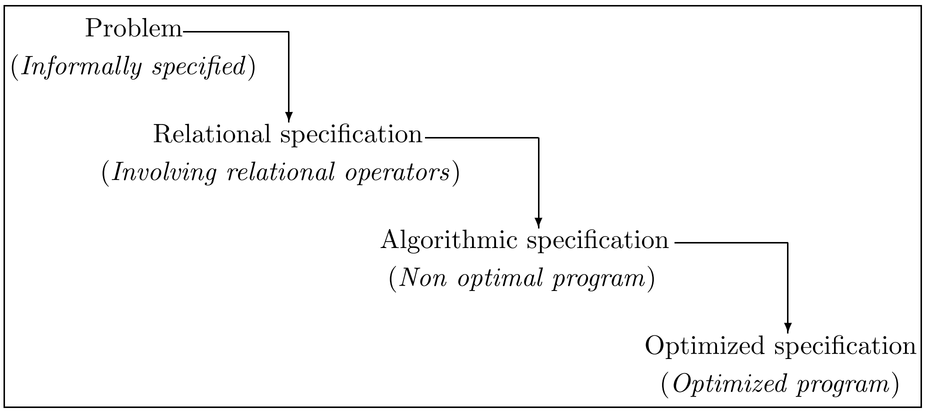

Figure 1: The stages in the relational development of algorithms.

Fork algebras were devised as a relational framework for program specification and construction by formal calculations [2, 6, 7, 9, 16, 17]. Fork algebras were also used in algebraic logic, as a framework for equational axiomatization of theories coming from both classical and non classical logics [8, 12, 13, 14, 24, 25]. Among the algebraic properties of fork algebras, their representability originated a lot of work [8, 12, 13, 20, 21]. The representability of fork algebras has been always misunderstood regarding its applications in program construction. Its application was always described as "the portability of properties of the problem domain into the abstract calculus of fork algebras". In this paper we show that the results provided by the representation theorem are by far more important. We show that not only the heuristic power coming from concrete binary relations is captured inside the abstract calculus, but also design strategies for program development can be successfully expressed. This result makes fork algebras a programming calculus by far more powerful than it was previously thought. In Section 2 we will present basic definitions to be used throughout the paper. In Section 3 we define fork algebras as an algebraic class, and present arithmetical properties of fork algebras useful in the process of program development. In Section 4 the expressive power of fork algebras is analyzed, and the finitization theorem for fork algebras is presented. In Section 5, the methodology for program derivation and a formalization of the Greedy strategy are presented. Finally, in Section 6 we present the final comments about this work.

2 Basic Definitions and Results

Along this section and the rest of the paper we will assume the reader has a nodding acquaintance with elementary concepts of set-theory and first-order logic. As a reference text in both areas we refer the reader to [23]. Given a binary relation X in a set A, and $a, b \in A$, we will denote the fact that a and b are related via the relation X by $\langle a, b\rangle \in X$ or $a X b$ indistinctly.

Definition 2.1 Let E be a binary relation on a set A, and let R be a set of binary relations satisfying:

1. $\bigcup R \subseteq E$,
2. If by Id we denote the identity relation on the set A, then $\emptyset$, E and Id belong to R,
3. R is closed under set union, intersection and complement relative to E,
4. R is closed under relational composition (denoted by $|$) and converse (denoted by $\breve{}$). These two operations are defined by

$$X\,|\,Y = \{\langle a,b\rangle : \exists c \text{ such that } a\,X\,c \;\wedge\; c\,Y\,b\}$$

$$\breve{X} = \{\langle a,b\rangle : b\,X\,a\}\,.$$

Then, the structure $\langle R, \cup, \cap, {}^{-}, \emptyset, E, |, Id, \breve{}\,\rangle$ is called an *algebra of binary relations*.

Definition 2.2 A *relation algebra* (RA for short) is an algebraic structure of type $\langle A, +, \cdot, {}^{-}, 0, 1, ;, 1', \breve{}\,\rangle$, where $+, \cdot$ and $;$ are binary operations, ${}^{-}$ and $\breve{}$ are unary, and $0, 1$ and $1'$ are distinguished elements. Furthermore, the reduct $\langle A, +, \cdot, {}^{-}, 0, 1\rangle$ is a Boolean algebra, and the following identities are satisfied for all $x, y, z \in A$:

$$x;(y;z) = (x;y);z, \tag{Ax. 1}$$

$$(x+y);z = x;z + y;z, \tag{Ax. 2}$$

$$(x+y)\breve{} = \breve{x}+\breve{y}, \tag{Ax. 3}$$

$$\breve{\breve{x}} = x, \tag{Ax. 4}$$

$$x;1' = 1';x = x, \tag{Ax. 5}$$

$$(x;y)\breve{} = \breve{y};\breve{x}, \tag{Ax. 6}$$

$$x;y \cdot z = 0 \text{ iff } z;\breve{y} \cdot x = 0 \text{ iff } \breve{x};z \cdot y = 0. \tag{Ax. 7}$$

Alternative axiomatizations for the calculus of relations can be obtained by replacing Ax. 7 in Def. 2.2 by any of the two following formulas[1]:

$$(x;y)\cdot z \preceq (x \cdot z;\breve{y});(y \cdot \breve{x};z), \tag{1}$$
$$x;y \preceq z \iff \breve{x};\overline{z} \preceq \overline{y} \iff \overline{z};\breve{y} \preceq \overline{x}. \tag{2}$$

This axiomatization is not complete, i.e., there are properties (even equations) of algebras of binary relations which cannot be derived from de axioms.

In Def. 2.3 below we introduce some terminology to be used in further sections.

[1]The symbol $\preceq$ in formulas (1) and (2), stands for the ordering induced by the Boolean algebra substructure.

DEFINITION 2.3 A relation F is called *functional* if it satisfies the formula

$$\breve{F};F \preceq 1'.$$

A relation I is called *injective* if it satisfies the formula

$$I;\breve{I} \preceq 1'.$$

A relation S is called *symmetric* if it satisfies the condition

$$\breve{S} = S.$$

A relation T is called *transitive* if it satisfies the formula

$$T;T \preceq T.$$

A relation D is called a *left-ideal* if it satisfies the condition

$$D = 1;D,$$

and a *right-ideal* if it satisfies

$$D = D;1.$$

A relation C is called a *constant* if C is functional, left-ideal and satisfies the condition

$$C;1 = 1.$$

By $Dom\,(R)$ we denote the term $\left(R;\breve{R}\right)\cdot 1'$ (the *domain* of the relation R), and by $Ran\,(R)$ we denote the term $\left(\breve{R};R\right)\cdot 1'$ (the *range* of the relation R).

Notice that when restricted to algebra of binary relations, the conditions in Def. 2.3 characterize familiar notions. For example, a binary relation satisfying the condition $T;T \preceq T$ will in effect be transitive.

3 Proper and Abstract Fork Algebras

Proper fork algebras (PFAs for short) are extensions of algebras of binary relations with a new operator called *fork*, and denoted by $\underline{\nabla}$. This new operator induces a structure on the underlying domain of PFAs. The objects, instead of being binary relations on a plain set, are binary relations on a structured domain $\langle A, \star\rangle$, where $\star$ fulfills some simple conditions. Fig. 2 shows the relationship existing between fork and $\star$, namely, that fork is defined in terms of $\star$ by the condition:

$$x R \underline{\nabla} S y \iff \exists u, v (y = \star(u, v) \ \wedge\ x R u \ \wedge\ x S v). \tag{3}$$

In order to define PFAs, we will first define the class of $\star$PFAs by

DEFINITION 3.1 A $\star$PFA is a two–sorted structure with domains $\mathcal{P}\,(V)$ and U

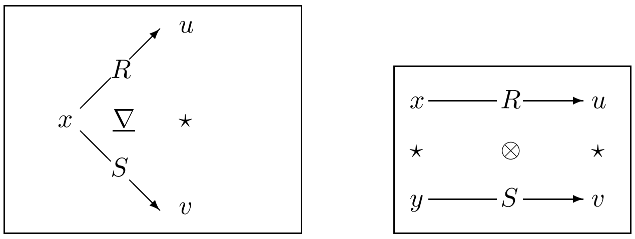

Figure 2: The operators *fork* and *cross*

$$\langle \mathcal{P}(V), U, \cup, \cap, {}', \emptyset, V, |, Id, \breve{}, \underline{\nabla}, \star \rangle$$

such that

1. V is an equivalence relation on the set U,
2. $|$, Id and $\breve{}$ stand respectively for composition between binary relations, the diagonal relation on U and the converse of binary relations, thus making the reduct $\langle \mathcal{P}(V), \cup, \cap, {}', \emptyset, V, |, Id, \breve{} \rangle$ 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 $x\,V\,y$ and $x\,V\,z$, also $x\,V \star (y, z)$,
5. $R \underline{\nabla} S = \{\langle x, \star(y, z)\rangle : x\,R\,y \ \wedge \ x\,S\,z\}$.

If by S we denote the operator on classes of algebras that closes a given class under subalgebras, and by Rd we denote the operator that obtains reducts to the similarity type $\langle \cup, \cap, {}', \emptyset, V, |, Id, \breve{}, \underline{\nabla} \rangle$, we obtain the following definition.

Definition 3.2 The class of PFAs is defined as S Rd $\star$PFA.

In Defs. 3.1 and 3.2, the function $\star$ performs the role of pairing, encoding pairs of objects into single objects. It is important to notice that there are $\star$ functions which are far from being set-theoretical pair formation, i.e., there are models in which $\star(x, y)$ is not the same as $\langle x, y \rangle$.

Once we have a complete definition of fork, there is another operation that is of interest in the specification and development of programs, and whose usefulness will be evident in further sections. This operation, called *cross*, given a pair of binary relations performs a kind of parallel product. A graphic representation of cross is given in Fig. 2. Its set theoretical definition is given by

$$R \otimes S = \{\langle \star(x, y), \star(u, v)\rangle : x\,R\,u \ \wedge \ y\,S\,v\}.$$

It is not difficult to check that cross is definable from the other relational operators with the use of fork. It is a trivial exercise to show that

$$R \otimes S = ((Id \underline{\nabla} V)\breve{}\,|R) \underline{\nabla} ((V \underline{\nabla} Id)\breve{}\,|S).$$

If we keep in mind the set theoretical definition of the relational operators, the elementary theory of binary relations [22] extended with the axiom (3) defining fork is a reasonable framework for software specification. Since it contains

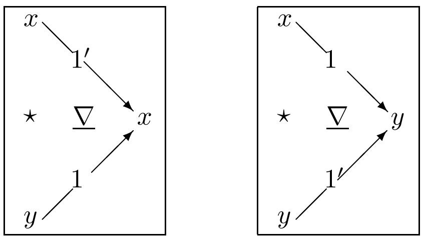

Figure 3: The projections π and ρ.

all first-order logic, it is clearly expressive enough. Programs could be specified as the relation established between input and output data. In doing so, we should work with variables ranging over two different kinds of objects. Variables ranging over relations would represent programs, while variables ranging over individuals represent data to be used by those programs. This controversial situation is not new in program construction, for it was already suffered by people working in functional frameworks. Their solution, in order to obtain simple frameworks, was to look for more abstract calculi on which everything be a function (variables ranging over individuals, often called *dummy* variables, were eliminated).

It is in the search of an abstract framework for relational calculi, that abstract fork algebras (to be introduced next) appear.

Definition 3.3 An abstract fork algebra is an algebraic structure

$$\langle R, +, \cdot, ^{-}, 0, 1, ;, 1', ^{\smile}, \nabla \rangle$$

satisfying the following axioms.

Axioms stating that the reduct $\langle R, +, \cdot, ^{-}, 0, 1, ;, 1', ^{\smile} \rangle$ is a relation algebra in which $\langle R, +, \cdot, ^{-}, 0, 1 \rangle$ is the Boolean reduct (where $\preceq$ denotes the induced partial ordering), $\langle R, ;, 1' \rangle$ is the monoid reduct, and $^{\smile}$ stands for relational converse,

$$r \nabla s = (r ; (1' \nabla 1)) \cdot (s ; (1 \nabla 1')) , \tag{Ax. 8}$$

$$(r \nabla s) ; (t \nabla q)^{\smile} = \left(r ; \breve{t}\right) \cdot (s ; \breve{q}) , \tag{Ax. 9}$$

$$(1' \nabla 1)^{\smile} \nabla (1 \nabla 1')^{\smile} \preceq 1'. \tag{Ax. 10}$$

From the abstract definition of fork induced by the axioms in Def. 3.3, it is possible to define cross by the equation

$$R \otimes S = ((1' \nabla 1)^{\smile} ; R) \nabla ((1 \nabla 1')^{\smile} ; S). \tag{4}$$

There are two relations that, because of their meaning in the standard models of fork algebras, behave as projections (see Fig. 3). These relations, namely, $(1' \nabla 1)^{\smile}$ and $(1 \nabla 1')^{\smile}$, are named respectively π and ρ.

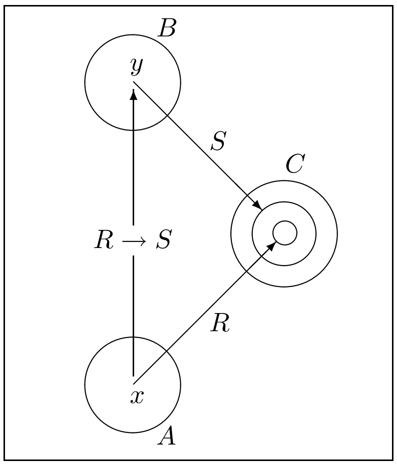

Figure 4: The *relational implication*.

Along the paper many times we will use a two-dimensional notation which simplifies the understanding of long terms and equations. For example, instead of writing $(a+b)\nabla(c+d)$, we may write

$$\begin{pmatrix} a+b \\ \nabla \\ c+d \end{pmatrix}.$$

The two-dimensional notation is particularly adequate for operations like fork and cross (see Fig. 2).

3.1 The Relational Implication

We define another operation on binary relations called *relational implication*. We define the relational implication of relations R and S, in terms of the relational operators previously defined, by

$$R \to S = \overline{R;\breve{\overline{S}}}, \tag{5}$$

The set theoretical definition (see Fig. 4 for a graphical interpretation) is given by

$$R \to S = \{\langle x, y\rangle : \forall z\,(x\,R\,z \Rightarrow y\,S\,z)\}.$$

4 Expressiveness and Finitization

In this section we analyze two important characteristics of fork algebras, namely, their expressiveness and their finite axiomatizability. We will present results stating that first-order theories can be interpreted into equational theories in abstract fork algebras, and also that the class of PFAs has a finitely axiomatizable theory. Finally, we will show the relationship existing between these results and the development of programs within 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 $\langle A, L' \rangle$ 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 (whose proof is given in [14]) holds.

THEOREM 4.1 *There exists a recursively defined mapping T translating formulas in L into equations in $\langle A, L' \rangle$ satisfying*

$$\Gamma \vdash \alpha \qquad \Longleftrightarrow \qquad \{T(\gamma) = 1 : \gamma \in \Gamma\} \vdash_{\nabla} T(\alpha) = 1.$$

The symbol $\vdash_{\nabla}$ in Thm. 4.1 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.

Theorem 4.1 has a strong application in program development 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), Thm. 4.1 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.

Fork algebras' expressiveness theorems 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 is necessary to be able to check their correctness and termination, supply general rules, strategies and heuristics, and demonstrate their validity.

By now, no relationship has been established between PFAs and AFAs, despite of the more or less obvious fact that every PFA satisfies the axioms of AFA, and thus is an AFA itself. This means that *some* of the AFAs are algebras where the objects are binary relations, but their could be some other AFAs where this property does not hold. The consequence of the existence of such AFA is that our calculus, even though simple and expressive, would be of little heuristic value because of the lack of intuition about the objects being manipulated. Fortunately, as will be shown in the following paragraphs, this is not the case and abstract fork algebras are algebras of binary relations.

The finitization theorem, which establishes that the axioms defining AFAs give an axiomatization for PFAs [8, 10, 12], provides important arguments for overcoming the limitations of the equational theory of fork algebras in program construction, as well as guarantees that fork algebras are algebras of binary relations.

If we want to show that the axioms characterizing AFAs offer an axiomatization of PFAs, we can proceed as follows. First, it is easy to show that the axioms characterizing AFAs hold in the class of PFAs. If as a second step we prove a representation theorem, asserting that every AFA is isomorphic to some PFA, we are done, since both results together guarantee that AFAs offer a finite axiomatization for PFAs.

THEOREM 4.2 *Every abstract fork algebra is isomorphic to a proper fork algebra.*

A complete proof of this theorem is given in [11, 15],As a corollary of Thm. 4.2, we obtain the following result.

THEOREM 4.3 *Th(AFA) = Th(PFA), i.e., AFAs and PFAs are elementarily equivalent classes of algebras.*

This elementary equivalence between AFAs and PFAs is extremely useful in our setting, since first-order formulas in the language of fork algebras now have a clear meaning when being considered as assertions about binary relations (or programs). It will be shown in Section 5 that, while equations suffice to express algorithms, first order formulas about relations can be used to describe design strategies for program development. This adds a new dimension to the development of algorithms within fork algebras.

5 Program Construction within Fork Algebras — The greedy strategy

A programming calculus can be viewed as a set of rules to obtain, in a more or less systematic way, programs out of specifications.

A very interesting and popular approach, is the one based on functional programming languages [4]. In these functional frameworks, specifications and programs are expressed in the same language, and transformation rules are defined in a suitable, frequently ad-hoc, metalanguage. A main drawback of functional settings, is the lack of expressiveness of their specification languages which are confined to functional expressions. These functional specifications, though inefficient when viewed as programs, are running programs, and thus, when specifying a problem, we previously need to have some algorithm solving it.

On the other hand, relational calculi have a more expressive specification language (because of the existence of the converse and complement of relations), allowing for more declarative specifications. However, choosing a relational framework is not a guarantee for a calculus to be totally adequate. These frameworks, as for instance the one proposed by Möller in [19], even though having a powerful specification language, also have some methodological drawbacks. The process of program derivation is aimed to use only abstract properties of relations, on which variables ranging over individuals (often called dummy variables) are avoided. Nevertheless, since no complete set of abstract rules exists capturing all the information of the relational (semantical) framework, the process goes back and forth between abstract and concrete properties of relations.

When using fork algebras as a programming calculus [2, 6, 7, 9, 16, 17], we have (as shown in Thm. 4.1), the expressiveness of first-order logic. Furthermore, as Thm. 4.3 shows, the axioms of AFAs provide a *complete* characterization of PFAs. These results enable us to use first-order logic as a specification language, certain fork algebra equations as programs, and to reason about the properties of specifications and programs within the theory. Moreover, Thm. 4.3, when establishing the elementary equivalence between $Th(\mathsf{AFA})$ and $Th(\mathsf{PFA})$, allows to formulate strategies and heuristics of the program construction process in the shape of first-order formulas about relations.

A very popular strategy used in the solution of problems and in the design of programs is the *greedy strategy*. This strategy is particularly useful in optimization problems, as for example finding the *minimum spanning tree*, or the *coin exchange* problem. In the next paragraphs, we will show how to formalize this strategy within fork algebras as a predicate over relations, as well as discuss its correctness.

It is well known the relationship existing between greedy algorithms and weighted matroids, namely, that the greedy strategy applied on a problem adequate for being modelized with a weighted matroid structure yields an optimal solution.

Definition 5.1 Weighted matroids are combinatorial structures $\langle \mathcal{S}, \mathcal{I} \rangle$ in which $\mathcal{S}$ is a finite set, and $\mathcal{I}$ is a subset of the powerset of $\mathcal{S}$ (called the set of independent subsets of $\mathcal{S}$) satisfying the following conditions:

- If $A \in \mathcal{I}$ and $B \subseteq A$, then $B \in \mathcal{I}$ (thus, the empty set $\emptyset$ belongs always to $\mathcal{I}$).
- If $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|A| < |B|$, then there exists $x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$ (this is known as the *exchange property*).

The greedy algorithm applied to a weighted matroid, is given by the following pseudocode:

```
1. $A \leftarrow \emptyset$;
2. Sort $\mathcal{S}$ in increasing order by weight $w(x)$;
3. **for each** $x$ taken in the order imposed above **do**        (6)
4.     **if** $A \cup \{x\} \in \mathcal{I}$ **then**
5.         $A \leftarrow A \cup \{x\}$;
6. **return** $A$.
```

It is well known the fact that the previous algorithm returns an independent subset of maximal size and minimum weight.

Let us consider a problem P whose relational specification has the form

$$P = \sqsupseteq ;Cond \cdot Max \cdot (\sqsupseteq ;Cond \cdot Max \quad \rightarrow \quad \leq)\,, \tag{7}$$

where

1. $\sqsupseteq$ generates as output all the subsets of a set given as input,
2. $Cond$ is a filter satisfying that for a given finite set S, $\sqsupseteq ;Cond$ gives as output a family of independent subsets of S. Thus, for each set S, the structure $\langle S, Im_S(\sqsupseteq ;Cond)\rangle$ is a weighted matroid,
3. $\sqsupseteq ;Cond \cdot Max$ produces as output the independent subsets of maximum size.
4. $\leq$ is a linear ordering given by the weight of the elements.

Thus, the term

$$\sqsupseteq ;Cond \cdot Max \cdot (\sqsupseteq ;Cond \cdot Max \ \rightarrow \ \leq)$$

retrieves independent subsets of maximum size and minimum weight from a matroid, and thus P can be implemented by means of the greedy algorithm. We will define a predicate *Greedy* that will characterize those problems that can be solved by the greedy algorithm. We will use an implementation of sets based on lists which will simplify the treatement of sets.

$$
\begin{aligned}
&Greedy(P, \sqsupseteq, Cond, Max, \leq) \iff \\
&\quad P = \sqsupseteq ;Cond \cdot Max \cdot (\sqsupseteq ;Cond \cdot Max \rightarrow \leq) \ \wedge && (8)\\
&\quad \sqsupseteq = 1'_{L^0} + 1'_{L^{>0}};Cons^{\smile};\begin{pmatrix} 1' \\ \otimes \\ \sqsupseteq \end{pmatrix};Cons + 1'_{L^{>0}};Tl;\sqsupseteq \ \wedge && (9)\\
&\quad Cond \preceq 1' \ \wedge \ Cond;\sqsupseteq \preceq \sqsupseteq ;Cond \ \wedge && (10)\\
&\quad \overline{\begin{matrix} (\sqsupseteq ;Cond) \cdot Max \\ \nabla \\ (\sqsupseteq ;Cond) \cdot \overline{Max} \end{matrix} ;\begin{pmatrix} HAS \\ \otimes \\ 1' \end{pmatrix}} ;\notin ;Cons;Cond;1 = 0 \ \wedge && (11)\\
&\quad \begin{pmatrix} HAS \\ \nabla \\ (\sqsupseteq ;Cond) \cdot Max \end{pmatrix};\notin ;Cons;Cond = 0. && (12)
\end{aligned}
$$

Formula (8) states that problem P can be specified as in (7). Formula (9) states that $\sqsupseteq$ generates all the subsets. Formula (10) states that *Cond* is a filter and that sets satisfying *Cond* are closed under subsets. Formula (11) describes the exchange property of matroids. Finally, formula (12) states that subsets produced by the relation $(\sqsupseteq ;Cond) \cdot Max$ have maximum size. Thus, formula *Greedy* characterizes those problems that can be solved using the greedy algorithm.

In our formalism of fork algebras, we can give a general characterization for greedy algorithms in terms of predicates over relations. In order to do this in a modular way, we will first define some auxiliary predicates.

In order to characterize a relation P producing the sorted output of the objects in the set $\mathcal{S}$ (see steps 1–3 in (6)), we will use a predicate $MINIMAL$. $MINIMAL$ is defined as

$$
\begin{aligned}
MINIMAL(P, \leq) \iff P = \\
\begin{pmatrix} Has \\ \otimes \\ 1' \end{pmatrix};\notin ;\pi \cdot \begin{pmatrix} Has \\ \otimes \\ 1' \end{pmatrix};\notin ;\pi \rightarrow \leq
\end{aligned}
$$

where the relation $\leq$ is a linear ordering of objects and the relation $\notin$ is the filter

$$\{\langle x \star A, x \star A\rangle : x \notin A\}.$$

If we now consider the relation P characterized by the term

$$\begin{pmatrix} Has \\ \otimes \\ 1' \end{pmatrix};\notin ;\pi \cdot \begin{pmatrix} Has \\ \otimes \\ 1' \end{pmatrix};\notin ;\pi \rightarrow \leq$$

this relation equals

$$\{\langle S \star S_0, y\rangle : y \text{ is the minimum in the set } S \setminus S_0\}\,.$$

The next predicate we will define is the predicate $NEXT$. This predicate will characterize the binary relation

$$\{\langle S \star S_0, y \star S_1\rangle : y \text{ is the minimum of } S \setminus S_0 \ \wedge\ S_1 = S_0 \cup \{y\}\}\,.$$

From the description of $NEXT$, it seems clear that the predicate $MINIMAL$ can be of use. $NEXT$ is defined by:

$$NEXT(P, \leq) \iff$$
$$\exists Q \left(MINIMAL(Q, \leq) \ \wedge\ P = \left(\begin{array}{c} Q \\ \nabla \\ \rho \end{array}\right) ; \left(\begin{array}{c} \pi \\ \nabla \\ Add \end{array}\right)\right). \tag{13}$$

Once we have defined the predicate $NEXT$, we will define an auxiliary predicate $Greedy\_Sol'$, on whose definition will be based the definition of the predicate $Greedy\_Sol$. The predicate $Greedy\_Sol'$ almost characterizes greedy algorithms. Its only difference is that it accepts not only the input for the problem, but also a sequence of already visited elements and the partial construction of the set A.

$$Greedy\_Sol'(P, \leq, Cond) \iff$$
$$\exists Q \text{ such that } NEXT(Q, \leq) \ \wedge$$
$$P = \left(\begin{array}{c} \breve{2} \\ \otimes \\ 1' \end{array}\right) ;\rho +$$
$$\left(\begin{array}{c} \pi;\pi \\ \nabla \\ \left(\begin{array}{c} Q \\ \otimes \\ 1' \end{array}\right) ; \left(\left(\begin{array}{c} \pi;\rho \\ \nabla \\ \left(\begin{array}{c} \pi \\ \otimes \\ 1' \end{array}\right) ;Add;Cond \end{array}\right) + \left(\begin{array}{c} \left(\begin{array}{c} \rho \\ \otimes \\ 1' \end{array}\right) \\ \nabla \\ \left(\begin{array}{c} \pi \\ \otimes \\ 1' \end{array}\right) ;Add;\neg Cond \end{array}\right) ;\pi \right) \end{array}\right) ; \left(\begin{array}{c} \left(\begin{array}{c} 1' \\ \otimes \\ \pi \end{array}\right) \\ \nabla \\ \rho;\rho \end{array}\right) ;P. \tag{14}$$

The term

$$\left(\begin{array}{c} \breve{2} \\ \otimes \\ 1' \end{array}\right) ;\rho$$

is different of 0 just in case all the generated objects were already considered and, in this case, projects as output the set A.

On the other hand, if still there are objects that were not considered, then Q (because $NEXT(Q, \leq)$ holds) will produce the smallest one. Thus, the term $(Q \otimes 1')$ produces as output the smallest nontreated element, and at the same time keeps a copy of the current state of the set A.

The term

$$\left(\begin{array}{c} \pi \\ \otimes \\ 1' \end{array}\right) ;Add;Cond$$

tests is the newly generated object satisfies the criteria for belonging to the set A, and if this is the case, adds the new element to the set A. If the new element does not satisfy the criteria, then nothing is done.

Finally, the predicate *Greedy_Sol* is defined as:

$$Greedy\_Sol(P, \leq, Cond) \iff$$

$$\exists Q \left(Greedy\_Sol'(Q, \leq, Cond) \wedge P = \left(\begin{array}{c} \left(\begin{array}{c} 1' \\ \nabla \\ Empty\_Visited \end{array} \right) \\ \nabla \\ Empty\_Set \end{array} \right) ; Q \right) . \tag{15}$$

The predicate *Greedy_Sol* guarantees an adequate initialization for the relations being defined by the predicate *Greedy_Sol′*.

Finally, it is important to mention that we have derived an algorithm for the problem of finding all the *minumum spanning trees* of a given connected graph. In order to do that we have just needed to state relations satisfying the predicate *Greedy* descripted above, and then instantiate the predicate *Greedy_Sol*.

6 Conclusions

In this paper we have presented an important application of the representation theorem for fork algebras, by showing that it is possible to express development strategies in the first-order language of fork algebras. This makes fork algebras a framework easier to handle than for example CIP-L [1], on which the rules are written in a metalanguage.

References

[1] Bauer, F.L., Berghammer, R., Broy, M., Dosch, W., Geiselbrechtinger, F., Gnatz, R., Hangel, E., Hesse, W., Krieg–Brückner, B., Laut, A., Matzner, T., Möller, B., Nickl, F., Partsch, H., Pepper, P., Samelson, K., Wirsing, M., and Wössner, H., *The Wide Spectrum Language CIP–L*, LNCS 183, Springer–Verlag, 1985.

[2] Baum, G.A., Frias, M.F., Haeberer, A.M. and Martínez López, P.E., *From Specifications to Programs: A Fork–algebraic Approach to Bridge the Gap*, in Proceedings of MFCS'96, Cracow, September 1996.

[3] Bird, R., *An Introduction to the Theory of Lists.* In M. Broy, editor, Logic of Programming and Calculi of Discrete Design, volume 36 of NATO ASI Series F, pp. 3–42. Springer–Verlag,1987.

[4] Bird, R., *A Calculus of Functions for Program Derivation*, in Turner D., editor, Research Topics in Functional Programming, University of Texas at Austin Year of Programming Series, Addison–Wesley 287–308.

[5] Bird, R. and de Moor, O. *Relational Program Derivation and Context–free Language Recognition.* In A .W. Roscoe, editor, A Classical Mind: Essays dedicated to C. A. R. Hoare. Prentice Hall, 1995.

[6] Frias, M.F. and Aguayo, N.G., *Natural Specifications vs. Abstract Specifications. A Relational Approach*, in Proceedings of SOFSEM '94, Milovy, Czech Republic, November 1994, 17–22.

[7] 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.

[8] 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.

[9] Frias, M.F. and Gordillo, S.E., *Semantic Optimization of Queries to Deductive Object–Oriented Database*, to appear in Proceedings of ADBIS'95, Moscow, June 1995, Springer–Verlag, pp.55–72.

[10] Frias, M.F., Haeberer, A.M. and Veloso, P.A.S., *On the Metalogical Properties of Fork Algebras*, in Abstracts of the Winter Meeting of the ASL, San Francisco, California, January 1995, The Bulletin of Symbolic Logic (1)3, 1995, pp.364–365.

[11] Frias, M.F., Haeberer, A.M. and Veloso, P.A.S., *A Finite Axiomatization for Fork Algebras*, Journal of the IGPL, to appear, 1996.

[12] Frias, M.F., Haeberer, A.M., Veloso, P.A.S. and Baum, G.A., *Representability of Fork Algebras*, in Proceedings of the Logic Colloquium '94, July, 1994, p. 51. Also in The Bulletin of Symbolic Logic (1)2, 1995, pp.234–235.

[13] Frias, M.F., Haeberer, A.M., Veloso, P.A.S. and Baum, G.A., *Widening Representable Fork Algebras*, presented at the Tenth International Congress of Logic, Philosophy and Methodology of Science, Florence, Italy, August 1995.

[14] Frias, M.F. and Orlowska, E., *Equational Reasoning in Non-Classical Logics*, Journal of Applied Non-Classical Logics, to appear.

[15] Gyuris, V., *A Short Proof for Representability of Fork Algebras*, Journal of the IGPL, vol 3, N.5, 1995, pp.791–796.

[16] 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 International Conference on Formal Methods in Programming and Their Applications, LNCS 735, Springer–Verlag, 1993, 281–298.

[17] 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., IFIP WG. 2.1, Bernhard Möller, 1991, 319–371.

[18] de Moor, O., *Categories, Relations and Dynamic Programming*, Mathematical Structures in Computer Science, Vol.4, pp.33–69, Cambridge University Press, 1994.

[19] Möller, B., *Relations as a Program Development Calculus*, in Constructing Programs from Specifications, B. Möller, editor, North–Holland, 1991, 373–397.

[20] Németi, I., *Representability of Fork Algebras depends on your Ontology*, Journal of the IGPL, to appear.

[21] 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, also appeared in Relational Methods in Computer Science, Dagstuhl–Seminar–Report 80, Schloss Dagstuhl, C. Brink and G. Schmidt eds., January 1994.

[22] Tarski, A., *On the Calculus of Relations*, Journal of Symbolic Logic, vol. 6, 1941, 73–89.

[23] *Handbook of Mathematical Logic*, Jon Barwise (Ed.), North Holland, 1977.

[24] Veloso, P.A.S. and Haeberer, A.M., *A Finitary Relational Algebra for Classical First-Order Logic*, Bull. Section of Logic, Polish Academy of Sciences, vol. 20, no. 2, 1991, 52–62.

[25] Veloso, P.A.S., Haeberer, A.M., and Frias, M.F., *Fork Algebras as Algebras of Logic*, in Proceedings of the Logic Colloquium '94, July, 1994, p. 127. Also in The Bulletin of Symbolic Logic, (1)2, 1995, pp.265–266.