ISSN 0103-9741

Monografias em Ciência da Computação
nº 01/94

On Fork Algebras and Reasoning about Programs

Paulo A. S. Veloso
Armando M. Haeberer

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Monografias em Ciência da Computação, Nº 01/94 ISSN 0103-9741
Editor: Carlos J. P. Lucena January, 1994

On Fork Algebras and Reasoning about Programs *

Paulo A. S. Veloso

Armando M. Haeberer

* Research partly sponsored by the Brazilian agencies CNPq, RHAE and FAPERJ.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br

ON FORK ALGEBRAS AND REASONING ABOUT PROGRAMS

Paulo A. S. VELOSO
{e-mail: veloso@inf.puc-rio.br}

Armando M. HAEBERER
{e-mail: armando@inf.puc-rio.br}

PUCRioInf MCC 01/94

Abstract

Fork algebras provide a useful basis for relational reasoning about programs and specifications. They arise as extensions of relational algebras with a new operator, called fork, which enables the introduction, by definition, of projections. In this paper we examine some fundamental issues concerning fork algebras, presenting the proofs of expressiveness and representability and discusstheir importance for reasoning about programs.

Key words:

Programming calculi, Relational Calculi, Formal Specifications, Algebras of Logic, Expressiveness. Representability.

Resumo

Álgebras de "fork" fornecem uma base apropriada para o raciocínio relacional sobre programas e especificações. Aparecendo como extensões das álgebras relacionais com um novo operator de bifurcação, "fork", elas permitem a introdução, por definição, de projeções. Este trabalho examina algumas questões fundamentais acerca de álgebras de "fork", apresentando demonstrações dos resultados de expressividade e representabilidade, bem como discute sua importância para derivação de programas.

Key words:

Cálculos de Programação, Cálculos Relacionais, Especificações Formais, Álgebras da Lógica, Expressividade. Representabilidade.

CONTENTS

INHERITANCE IN OBLOG SPECIFICATION
A SOUND AND COMPLETE SYSTEM FOR BEHAVIOUR REWRITING

Abstract

In the Oblog framework, an object-oriented language to system specification, we present a behaviour rewriting system. The system, that is sound and complete with respect to the inheritance characterization in Oblog, defines a new inheritance syntax and is developed with the theory of graph grammars. Categorial techniques are used to connect the semantic domain proposed for objects, with state machines and computations.

1. INTRODUCTION

Fork algebras provide a useful basis for relational calculi for programming. We examine some fundamental issues concerning fork algebras and their use in writing programs and reasoning about them. Fork algebras arise as extensions of relational algebras with a new operator, fork, which enables the introduction of projections by definition. The abstract calculus of fork algebras manipulates fork-relational terms without variables, free or bound, over individuals. This calculus, which provides a finitary algebra of classical first-order logic, is our formalism for reasoning about programs.

Two basic issues concerning a formalism for reasoning about programs concern its formal aspects (such as soundness and completeness) and its adequacy for reasoning about programs and specifications. We, accordingly, address both issues.

The (meta-)mathematical aspects of soundness and completeness of a calculus are connected to the limits, in principle, and there lies their importance. These are settled by two fundamental results, namely expressiveness (which shows that fork-relational terms have the expressive power of first-order formulas) and representability (which shows that any abstract model of our calculus is isomorphic to a intended one with relations of input-output pairs). As an added bonus, these results show that we have an algebra of classical first-order logic that is finitary, in constrast with other approaches [EU46, Hal62, HM74].

The adequacy of our calculus for reasoning about programs and specifications has been illustrated by several examples elsewhere [DB93, FW92, FAN93, Fri93, HVE90, HV91, HBS93, VHB92]. We argue here on the basis of general arguments connecting our basic results to the intended applications concerning programming.

The structure of this paper is as follows. Section 2 starts by reviewing Boolean and relational algebras [Tar41, JT52] and introduces fork algebras together with their theories and calculi; we then go on to the central mathematical results of expressiveness [VH91] and representability [BHV92, FBHV93]. Section 3 provides an overview of our approach, by arguing that we can view specifications and programs as relations and why this viewpoint is useful for reasoning about program development. Finally, section 4 provides some concluding remarks as well as indication of on-going and future work.

2. FORK ALGEBRAS

In this section we introduce some basic definitions concerning fork algebras and prove the crucial results concerning their ex[ressiveness and representability . Fork algebras are extensions of relational algebras, which in turn extend the well-known Boolean algebras. For this reason it is convenient to examine the progression from Boolean algebras, first to relational algebras, and then to fork algebras. We shall be interested in both concrete (or proper) and abstract versions of fork algebras: much as Boolean algebra provides an abstract theory of set-theoretical operations, relational algebras (and fork algebras) correspond to abstract theories of operations on (structured) relations.

2.1 BOOLEAN, RELATIONAL AND FORK ALGEBRAS

Consider a set W. We have the usual set-theoretical operations on subsets of W, namely union $\cup$, intersection $\cap$ and complement $^{\sim}$ (with respect to the universe W), in addition to special sets, namely the empty set $\varnothing$ and the universal set W. As usual, by a *field of sets* (FS, for short) we mean a structure $\mathcal{S} = \langle S,\cup,\cap,^{\sim},\varnothing,W\rangle$, where S is a set of subsets of W, with $\varnothing$ and W in S, which is closed under the above set-theoretical operations. The abstract version of a field of sets is a *Boolean algebra*: (BA, for short) a structure $\mathcal{B} = \langle B,+,\bullet,^{-},0,\infty\rangle$ satisfying the well-known axioms. As usual on a BA we can define a binary relation $\leq$ giving a lattice structure to it. Also, an element $a \in B$ is called an *atom* iff for every $b \in B$ such that $0 \neq b \leq a$ we have $b=a$; and Boolean algebra is *atomic* iff every non-zero element contains an atom. By Stone's representation theorem every Boolean algebra is isomorphic to a field of sets, where set-theoretical inclusion $\subseteq$ corresponds to the abstract partial order $\leq$.

Now let us examine algebras of relations [Tar41, CT51, JT52].

Consider now a set $V \subseteq W \times W$ of ordered pairs of elements of W. Then, we have some natural relation-theoretical operations on subsets of $W \times W$, namely *transposition* and *composition*, defined, respectively by $p' = \{ \langle u,v\rangle \in W\times W : \langle v,u\rangle \in p \}$ and $p|q = \{ \langle u,w\rangle \in W\times W : \langle u,v\rangle \in p$ and $\langle v,w\rangle \in q$ for some $v \in W \}$, as well as the special *diagonal* (or *identity*)*relation* $\Delta = \{ \langle u,v\rangle \in W\times W : v=u\}$. A *proper relation algebra* (PRA, for short) is a structure $\mathcal{P} = \langle P,\cup,\cap,|,^{\sim},',\varnothing,\Delta,V\rangle$, such that the reduct $\langle P,\cup,\cap,^{\sim},\varnothing,V\rangle$ is a field of sets with $\Delta \in P$ and P is closed under composition and transposition. The abstract version of a proper relation algebra is a *relational algebra* (RA, for short): a structure $\mathcal{R} = \langle R,+,\bullet,;,^{-},^{\wedge},0,1,\infty\rangle$ such that its reduct $\langle R,+,\bullet,^{-},0,\infty\rangle$ is a Boolean algebra, $\langle R,;,1\rangle$ is a monoid, and Schröder's rule ($r;s \leq t$ iff $r^{\wedge};t^{-} \leq s^{-}$ iff $t^{-};s^{\wedge} \leq r^{-}$) holds. We shall call a relation algebra *atomic* iff its Boolean reduct is so. We shall often be concerned with simple RA's, which are those satisfying Tarski's rule ($\infty;r;\infty=\infty$ whenever $r\neq 0$).

Let us now introduce proper and abstract fork algebras.

Consider a set U equipped with a binary injective operation $*:U\times U\rightarrow U$ (which we view as constructing pairs $y*z=[y,z]$ of elements of U). We now have another natural operation on subsets of U×U, namely (*proper*) *fork*, defined by $r\angle s = \{ <x,[y,z]>\in U\times U : <x,y>\in r$ and $<x,z>\in s \}$. A *proper fork algebra* (PFA, for short) is a structure $\mathscr{F}=\langle F,\cup,\cap,|,\angle,\smile,',\varnothing,\Delta,V\rangle$, where the reduct $\langle P,\cup,\cap,|,\smile,',\varnothing,\Delta,V\rangle$ is a proper relation algebra such that F is closed under $\angle$. The abstract version of a simple proper fork algebra is an *abstract fork algebra* (AFA, for short): a structure $\mathscr{A}=\langle A,+,\bullet,;,\nabla,^{-},^{\wedge},0,1,\infty\rangle$ such that $\langle A,+,\bullet,;,^{-},^{\wedge},0,1,\infty\rangle$ is a relational algebra satisfying Tarski's rule, and the following axioms hold

(1) $r\nabla s = (r;\pi)\bullet(s;\rho)$

(2) $(r\nabla s\ );(p\nabla q)^{\wedge} = (r;p^{\wedge})\bullet,(s;q^{\wedge})$

(3) $\pi\nabla\rho \leq 1$

(4) if $0\neq t\leq \infty\otimes\infty$ then $0\neq v\otimes w\leq t$ for some$v$ and$w$;

where projections and direct product are defined as follows:

(π) $\pi=(1\nabla\infty)^{\wedge}$ {first projection}

(ρ) $\rho=(\infty\nabla 1)^{\wedge}$ {second projection}

($\otimes$) $r\otimes s = (\pi;r)\ \nabla\ (\rho;s)$ {direct produc}

Notice that, because of Tarski's rule, our fork algebras are simple, in that their relational reducts are simple. We shall often be concerned with *atomic* fork algebras: those with atomic Boolean reducts. An extensive development of the arithmetic of fork algebras can be found in [Fri93a, Dur93].

2.2 Elementary Theories and Abstract Calculi

Let us now examine the languages for these structures. It is important to bear in mind a distinction between concrete, proper structures and their abstract counterparts: in the concrete versions one has individuals, whereas in the abstract versions one abstracts away from them and considers only the abstract objects, without, so to speak, looking into the individuals which constitute them. This distinction will be clarified by examining another way of introducing proper relation and fork algebras. The idea is extending (unsorted) first-order logic, with equality, by defining the concepts pertaining to relations.

First consider a single-sorted language $\mathcal{L}$, where the only sort is sort ι (of individuals) and the atomic formulas are of the form $x\approx y$, where x and y are variables of sort ι . Now, introduce a new sort $\imath$ (for binary relations over individuals) and a a ternary predicate symbol _(-,-) over sorts $\mathfrak{r}\, \iota\, \iota$.. We now have new atomic formulas of the form r(x,y) (which we also write as x r y). With this material we can introduce, by definitions, operations and constants for sort $\mathfrak{r}$. For instance, constant 1 is defined by the axiom $\forall x \forall y$ [x 1 y $\leftrightarrow$ x$\approx$y], and operation $;$ by $\forall x \forall y$ [x $r;s$ y $\leftrightarrow$ $\exists z$ (x r z $\wedge$ z s y)]; also equality = over sort $\mathfrak{r}$ is introduced by the definig axiom $r=s \leftrightarrow \forall x \forall y$ (x r y $\leftrightarrow$ x s y).

Thus, by introducing axioms like those of [Tar41], we obtain a a theory, called the *Elementary Theory of Binary Relations* (*ETBR*, for short) in the extended language $\mathcal{ELBR}$ (*Elementary Language of* (*Binary*) *Relations*). The new defining axioms explain the meaning of the new relative operations by means of first-order formulas involving individuals. In this sense, this theory provides a standard meaning for the relational theoretic terms. Indeed, for for any term p of sort r, we have a formula $\phi(x,y)$, without relational symbols, such that $ETBR \vdash (\forall x{:}\iota)(\forall y{:}\iota)[p(x,y) \leftrightarrow \phi(x,y)]$. This formula $\phi(x,y)=\mathcal{S}(p)$ will be called the *standard meaning* of relational term p. For instance, $\mathcal{S}(r;s)$ is the formula $\exists z$ (x r z $\wedge$ z s y).

The language of the RA's is the sub-language $\mathcal{ALBR}$ of $\mathcal{ELBR}$ consisting only of sort $\mathfrak{r}$ and $+,\bullet,;,^{-},^{\wedge},0,1,\infty$; and the restriction of *ETBR* to $\mathcal{ALBR}$ is the *Abstract Calculus ACBR of Binary Relations*.

Now, consider a model $\mathfrak{C}$ of *ETBR*. The realization of sort $\mathfrak{r}$ consists of elements which behave, in view of the standard meaning above, as sets of ordered pairs of elements of the realization of sort ι. In this sense, they are binary relations over individuals. Thus, the proper relation algebras are the reducts to language $\mathcal{ALBR}$ of the models $\mathfrak{C}$ of *ETBR*.

Now, let us examine the case of fork algebras. The situation corresponds to the one described above for algebras of relations.

We start with a first-order theory of individuals with injective binary operation $*$, form *ETBR* over it and extend the resulting theory by the axiom $(\forall x{:}\iota)(\forall y{:}\iota)[r\nabla s(x,y) \leftrightarrow (\exists v{:}\iota)(\exists w{:}\iota)(y \approx v*w \wedge r(x,v) \wedge s(x,w))]$ defining fork.

We thus obtain the *Elementary Theory ETFR of Fork Relations*.

The language of the PFA's is the sub-language $\mathcal{ALFR}$ of $\mathcal{ELFR}$ consisting only of sort $\mathfrak{r}$ and $+,\bullet,;,\nabla,^{-},^{\wedge},0,1,\infty$; and the restriction of *ETFR* to $\mathcal{ALFR}$ is the *Abstract Calculus ACFR of Fork Relations*.

As for the case of relations, we have a *standard meaning*, assigning to each fork term p of sort $\mathfrak{r}$ of $\mathcal{ELFR}$ a formula $\phi(x,y)=\mathcal{S}(p)$, without relational symbols and with variables x and y, over sort ι,, such that $ETFR \vdash (\forall x)(\forall y)[x\ p\ y \leftrightarrow \phi(x,y)]$. This formula $\phi(x,y)=\mathcal{S}(p)$ is *standard* *meaning* of relational term p. Also, the simple proper fork algebras are the reducts to language $\mathcal{ALFR}$ of the simple models $\mathfrak{C}$ of *ETFR*.

2.3 EXPRESSIVENESS OF FORK ALGEBRAS

Standard meaning reduces relational terms to first-order formulas. The question of expressiveness concerns the converse. Even though for RA's the answer is negative [Tar41], we. now have a positive answer [VH91, HV9]. Expressiveness will guarantee that the expressive power of our fork-relational terms is the same of first-order formulas: any n-ary relation defined by formula $\varphi(x_1, \ldots, x_n)$ can be defined by term t, in that in any structure the former is the domain, and range, of the latter.

Consider a *first-order language* $\mathcal{L}$ with equality $\approx$, given by a set of *predicate symbols* (we do not consider function symbols, since they can be replaced by their graphs). A *structure* $\mathfrak{B}$ for $\mathcal{L}$ consists of a *domain* B together with an n-ary relation $p^{\mathfrak{B}}$ *realizing* each n-ary predicate symbol p of $\mathcal{L}$. As usual, this can be extended to assign to a each formula φ its realization consisting of those n-tuples that satisfy it: $\varphi^{\mathfrak{B}} = \{\langle b_1,\ldots,b_n\rangle : \mathfrak{B} \models \varphi\,[\, b_1,\ldots,b_n \,]\}$ [Ebb80].

It is more convenient, however, to view a formula as defining a set of trees over B. For this purpose, we extend $\mathcal{L}$ to $\mathcal{L}^*$ by adding a binary operation symbol $*$. Given a structure $\mathfrak{B}$ for $\mathcal{L}$ we have an expansion $\mathfrak{B}^*$ to $\mathcal{L}^*$, with B^* consisting of the finite trees over B and $*$ realized as (injective) tree construction. We can replace $\mathfrak{B}$ by $\mathfrak{B}^*$ because $\varphi^{\mathfrak{B}}$ is the restriction of the realization of φ in $\mathfrak{B}^*$. Now, we can expand $\mathfrak{B}^*$ to a model $\hat{\mathfrak{B}}$ of *ETFR*, with a binary relational constant $\hat{p}$, corresponding to n-ary p, realized by $\{\langle b,b\rangle : b=b_1*(b_2*\ldots(b_{n-1}*b_n)\ldots)\ \&\ \mathfrak{B}^* \models p(x_1,\ldots,x_n)\,[\, b_1,\ldots,b_n \,]\}$

Theorem (Expressiveness) Given a first-order language $\mathcal{L}$, there exists a function $\mathcal{T}$ assigning to each formula $\varphi(x_1,\ldots,x_n)$ of $\mathcal{L}$, a closed term t with standard meaning $\mathcal{S}(t)=\psi(u,v)$, such that for every structure $\mathfrak{B}$ for $\mathcal{L}$, $\varphi^{\mathfrak{B}} = \{\langle b_1,\ldots,b_n\rangle : \hat{\mathfrak{B}} \models \psi\,[\, (b_1*(b_2*\ldots(b_{n-1}*b_n)\ldots)),(b_1*(b_2*\ldots(b_{n-1}*b_n)\ldots)) \,]\}$

Proof outline. We will define $\mathcal{T}$ by induction on the structure of φ.

Basis. If φ is $x_1 \approx x_2$, then we set $\mathcal{T}(x_1 \approx x_2) = 2^\wedge;2$. If φ is $p(x_1,\ldots,x_n)$, then we set $\mathcal{T}(p(x_1,\ldots,x_n)) = \hat{p}$. If φ is an atomic formula $p(y_1,\ldots,y_m)$ obtained from $p(x_1,\ldots,x_n)$ by a substitution σ, then we set $\mathcal{T}(p(y_1,\ldots,y_m)) = s^\wedge;\hat{p};s$, where $s^\wedge$ is a term, indicated in lemma B below, simulating σ.

Inductive step: By Lemma A below, we may replace φ by φ'. If φ is $\neg\psi$, then we set $\mathcal{T}(\neg\psi) = \mathcal{T}(\psi)^{-} \bullet 1$. If φ is $\psi \vee \theta$, then φ' is equivalent to $\psi' \vee \theta'$, and we set $\mathcal{T}(\psi \vee \theta) = \mathcal{T}(\psi') + \mathcal{T}(\theta')$. If φ is $(\exists x_\iota)\psi$, then φ' is equivalent to $(\exists x_v)\psi' \wedge x_n \approx x_n$, where ψ' is obtained from ψ by the substitution σ that interchanges x_i and x_n, and we set $\mathcal{T}((\exists x_v)\psi') = \pi^\wedge;\mathcal{T}(\psi');\pi$, whence we can obtain $\mathcal{T}(\varphi)$.

QED

Lemma A Given positive n and m, there exist terms d_n and e_m, such that, for every formula $\varphi(x_1,\dots,x_n)$, if φ' is the formula $\varphi \wedge x_1 \approx x_1 \wedge \dots \wedge x_n \approx x_n, \wedge \dots \wedge x_{n+m} \approx x_{n+m}$ we have $\mathcal{T}(\varphi') = \mathcal{T}(\varphi);e_m$ and $\mathcal{T}(\varphi) = \breve{d}_n;\mathcal{T}(\varphi');d_n$.

Lemma B Given a substitution σ on $\{1,\dots,n\}$, there exists a term s, such that for any formula $\varphi(x_1,\dots,x_n)$, if $\sigma(\varphi)$ is the formula obtained by applying σ to φ, then we can take $\mathcal{T}(\sigma(\varphi)) = s^\wedge;\mathcal{T}(\varphi);s$.

Corollary (Preservation of implication) Given formulas $\varphi(x_1,\dots,x_n)$ and $\psi(x_1,\dots,x_n)$, $\vdash \phi \to \psi$ iff $ETFR \vdash \mathcal{T}(\varphi) \leq \mathcal{T}(\psi)$.

2.4 Representability of Fork Algebras

We now show that atomic AFA's can be represented by PFA's [BHV92, FBHV93], in contrast with the case for relational algebras.

Lemma Let $= \langle A,+,\bullet,;,\nabla,^-,^\wedge,0,1,\infty \rangle$ be an atomic AFA. Then, for all $r,s,t,q \in A$, $r \neq 0 \to \pi;r \neq 0 \wedge \rho;r \neq 0$, $r \otimes s = 0 \leftrightarrow r = 0 \vee s = 0$, $r \otimes s \neq 0 \to \breve{\pi};(r \otimes s);\pi = r$, $r \otimes s \neq 0 \to \breve{\rho};(r \otimes s);\rho = s$, $(r \otimes s = t \otimes q) \wedge (r \neq 0) \wedge (s \neq 0) \wedge (t \neq 0) \wedge (q \neq 0) \to (r = t) \wedge (s = q)$, $r \otimes s$ is an atom iff both r and s are atoms, and if r is atom with $r \leq \infty \nabla \infty$ then $r = (r;\pi) \nabla (r;\rho)$.

Theorem (Representability) Every atomic AFA is isomorphic to a PFA.

Proof outline. Consider such an atomic AFA $\mathscr{A} = \langle A,+,\bullet,;,\nabla,^-,^\wedge,0,1,\infty \rangle$ with set of atoms $\mathcal{At}$. For $r \in A$, let $\mathcal{At}(r) = \{ \alpha \in \mathcal{At} : \alpha \leq r \}$, and set $U = \mathcal{At}(1)$. By the lemma, $\otimes$ is an injective operation on U. Thus, by defining $r \angle s = \{ \langle \alpha, \beta \otimes \gamma \rangle \in U \times U : \langle \alpha,\beta \rangle \in r$ and $\langle \alpha,\gamma \rangle \in s \}$, we have a PFA $\mathscr{F} = \langle F,\cup,\cap,|,\angle,^\sim,',\emptyset,\Delta,V \rangle$, with $V = U \times U$ and $F = \wp(V)$.

We introduce functions $\mathscr{D},\mathscr{R}:\mathcal{At} \to U$, for "inspecting the shapes of the atoms in domain range": $\mathscr{R}:\mathcal{At} \to U$ is defined by $\mathscr{R}(\alpha) = [(\alpha;\pi)^\wedge;(\alpha;\pi)] \otimes [(\alpha;\rho)^\wedge;(\alpha;\rho)]$, in case $\alpha \leq \infty \nabla \infty$, and $\mathscr{R}(\alpha) = \alpha^\wedge;\alpha$, otherwise, and $\mathscr{D}$ is "dual". We now define $\mathscr{H}: A \to F$ by $\mathscr{H}(r) = \{ \langle \mathscr{D}(\alpha),\mathscr{R}(\alpha) \rangle \in U \times U : \alpha \in \mathcal{At}(r) \}$. We can now show that $\mathscr{H}$ is an injective homomorphism from $\mathscr{A}$ into $\mathscr{F}$, the crucial steps being $\mathscr{H}(\infty) = V$ and $\mathscr{H}(r;\nabla s) = \mathscr{H}(r) \angle \mathscr{H}(r)$, whence, an isomorphism from $\mathscr{A}$ into a sub-algebra of $\mathscr{F}$. *QED*

Corollary The extension from *ACFR* to *ETFR* is conservative, in particular, for terms p and q, $ETFR \vdash p \leq q$ iff $ACFR \vdash p \leq q$.

These results show that our fork algebras provide a finitary algebra of first-order logic with equality.

3. Programming With Relations

We shall now examine some aspects underlying our fork-relational approach to programming [HV91, VH93, VHB92, HV90], whose formal basis rests on our expressiveness and representability results in section 2 . Reasoning about software artifacts (programs) involves proving that the program exhibits the required behavior on the basis of the specification of a data type.

3.1 BEHAVIOR AS RELATIONS

First, let us see why we can regard behavioral specifications and properties as relations.

It is well-known that one can faithfully reduce many-sorted first-order logic to the unsorted version, by considering relativization predicates that are intended to characterize the sorts. In terms of models, the universe $\mathcal{U}$ of the unsorted structure is regarded as the union of all sorts, which can be recovered by means of the relativization predicates provided.

In the relational setting, we consider each sort s as given by a partial identity δ_s, which characterizes it in the sense that $\delta_s = \{\langle u,u\rangle \in \mathcal{U} : u \in s\}$. We also consider binary relation symbols corresponding to the given operations and predicates [HV91].

For the data type lists with sorts $\mathcal{List}$ and $\mathcal{Elm}$, operations head, tail, cons, and predicate null, we have $\delta_{\mathcal{List}}$ and $\delta_{\mathcal{Elm}}$ corresponding to sorts $\mathcal{List}$ and $\mathcal{Elm}$, as well as relation symbols *head*, *tail*, *cons* and *null* corresponding to head, tail, cons, and null. We replace an expression like cons(x,y)≈z by *cons*(x∗y,z), and we can express functionality of relations and information concerning domain and range. We can express an axiom like $(\forall x{:}\mathcal{List})$ $[\text{null}(x) \vee (\exists y{:}\mathcal{Elm})(\exists z{:}\mathcal{List})\ \text{cons}(y,z) \approx x]$ by the equation $\delta_{\mathcal{List}} = \delta_{null} + (cons;cons^{\wedge})$, where $\delta_{null} = (null;null^{\wedge}) \bullet 1$.

Now, consider a property expressed by a formula, which may assumed already reduced to its unsorted version. Expressiveness shows how to translate such a formula into a term "saying the same thing" and this translation preserves implication. Also, a specification consisting of first-order axioms can be expressed relationally. Our results in section 2 show that first-order definitions and reasonings are exactly mirrored in *ACFR*. Therefore, reasoning within the abstract fork-relational calculus *ATFR* is both sound and complete with respect to standard first-order reasoning.

For instance, consider a sentence σ (say, a conjunction of specification sentences) and a sentence τ in the same language (say, expressing some property). By the above expressiveness result, we have terms $s = \mathcal{T}(\sigma)$ and $t = \mathcal{T}(\tau)$, and $\sigma \vdash \tau$ iff $ETFR \vdash s \leq t$ iff $ACFR \vdash s \leq t$.

Thus, we can safely replace first-order reasoning by reasoning within our relational calculus. But we do not have to; whenever it is more convenient we can resort to first-order reasoning, with the assurance that it can be translated into *ATFR*. And Representability provides an added bonus: we can reason by means of individuals (which is often more intuitive when one wishes to think in an input-output manner, by resorting to diagrams, for instance); if the conclusion no longer involves individuals it could be derived within *ATFR*..

3.2 Programs as Relations

Now, let us see why we can regard programs as relations.

The language $\mathcal{ALFR}$ of the abstract fork-relational calculus *ATFR* relational calculus has symbols for constant relations as well as for operations on relations, namely $+,\bullet,;,\nabla,^{-},^{\wedge},0,1$, and ∞. We are mostly interested in terms built from some relational constants by means of these operations. Standard meaning of such a term t is given by elementary theory *ETFR* of fork relations.

We call *algorithmic operations* those of $\mathcal{ALFR}$, except for complement $^{-}$. Given a set $\mathcal{BRCS}$ of *basic relational constant symbols*, we call *algorithmic terms* those of $\mathcal{BRCS}\cup\mathcal{ALFR}$ that do not involve $^{-}$. The reason for such a name is the fact that we can regard such terms as describing (possibly non-deterministic) programs. (Notice that we allow recursive terms to cover iterative and recursive programs.) There are two justifications for this.

First, the algorithmic operations preserve effective enumerability, in the following sense. Consider an effective domain $\mathcal{U}$, one where equality $\approx$ is effectively decidable and where the pair-forming operation $*:\mathcal{U}\times\mathcal{U}\longrightarrow\mathcal{U}$ is effectively computable. Then, the constant relations 0, ∞ , and 1 are effectively decidable. Also, the algorithmic operations preserve effective enumerability; for instance, if r and s are effectively enumerable, then one can effectively enumerate $r\nabla s$ by effectively enumerating those $\langle u,v\rangle\in r$ and $\langle u',w\rangle\in s$, deciding whether $u\approx u'$ and in such case computing $v*w$ and outputting $\langle u,v*w\rangle$. Thus, if the basic relational constant symbols in $\mathcal{BRCS}$ have effectively enumerable realizations, every algorithmic term denotes an effectively enumerable relation. Summing up, the standard meaning assigned to the algorithmic operations is programming-like, in that they preserve effective enumerability. In this sense, terms built from effectively enumerable relations by means of these algorithmic symbols can be regarded as programs.

Second, there is a natural correspondence between algorithmic operations (except, perhaps for $\bullet$ and $^{\wedge}$) and programming-language constructs. For instance, relative product corresponds to sequentialization, in that they have the same standard meaning. Even though intersection $\bullet$ does not quite follow this pattern, it can be simulated in terms of algorithmic operations via $r \bullet s = (r \nabla s);2^{\wedge}$ (where $2 = 1 \nabla 1$, which makes $2^{\wedge}$ a kind of equality test). This last equation shows a use of converse that has a natural, direct programming counterpart. The intuitive view of converse as "running backwards, from output to input" may appear not to correspond directly to a programming-language construct; but the very argument showing that converse preserves effective enumerability indicates how it can be simulated. Moreover, the idea of converse fits very well with the usual logic programming paradigm. Thus, there is a natural correspondence by means of which each algorithmic relational term can be translated into a program. (Notice that we allow recursive terms, which will translate into recursive programs.) That they have the same meaning comes from the connection, pointed out by van Emden and Kowalski, between deductive meaning and operational meaning.

The preceding remark may be interpreted as soundness: each algorithmic relational term can be translated into a program with the same meaning. The converse, completeness, would guarantee that no program is lost by limiting oneself to such terms. This comes from the above natural correspondence. For, each simple programming-language construct, such as sequentialization, case, etc., has a relational operation as direct counterpart. Thus a program text involving some basic symbols can be converted into a algorithmic relational term with the same meaning (here iteration and recursion are covered by recursive terms) [HV91, VHB92].

As a simple example of the above correspondence, consider doubling a natural number. The recursive program `dbl(x)` = **if** `is_zero(x)=true` **then** zero **else** `succ(succ(dbl(pred(x))))` **end_if** and the algorithmic term $dbl = \delta_{zero};zero+\delta_{not\_zero};pred;dbl;succ;succ$ correspond to each other. (Here δ_{zero} and δ_{not_zero} are partial identities over $\{0\}$ and the non-zero naturals, respectively, and can be obtained from relations corresponding to `is_zero`, `true` and `false`). A slightly simpler form is $dbl = \delta_{zero}+succ^{\wedge};dbl;succ;succ$.

3.3 FORK-RELATIONAL REASONING ABOUT PROGRAMS

Concepts related to correctness are easily handled in our relational setting. Partial correctness of p amounts to inclusion and termination to a property of the partial identity $\mathfrak{D}(p) = (p;p^{\wedge}) \bullet 1$; For instance, in order to establish termination of the above term dbl it suffices to prove $\delta_{zero} \leq \mathfrak{D}(dbl) \wedge \mathfrak{D}(dbl);succ \leq succ;\mathfrak{D}(dbl)$.

An important relationship between programs and/or specifications annotated with preconditions is that of refinement, whereby the degree of non-determinism is reduced without losing successful computations. In our case, we say that $<\phi,p>$ *refines* $<\theta,q>$ iff $\delta_\theta \leq \delta_\phi$ and $\delta_\theta;p \leq q$ [HV91].

Our argument so far appears to emphasize input-output behavior, more appropriate for terminating programs. But, it can be extended to any behavior described by first-order formulas, because of the results in section 2, and the fact that one write algorithmic terms corresponding to the traces of programs.

Now, let us consider the aspect of program derivation. In view of the above arguments, the essence of this activity amounts to deriving an algorithmic term from some given (not necessarily algorithmic) terms, corresponding to the specification. One crucial aspect is the elimination of non-algorithmic constructs (mainly complement/negation and relative sum/universal quantification). They are often eliminated in favor of recursion by relying on the inductive structure of the domain; and there are some general strategies for this task [VHB92, VH93].

We can express strategies in fork-relational terms. Thus, one could simulate derivations like those done in CIP. Also, general problem-solving strategies, such as divide-and-conquer, etc. can be expressed and used [HV91, VH93].

Reasoning within *ACFR* offers some conveniences for manipulation, such as not involving individuals (an advantage over usual programming or first-order reasoning, shared by the functional approach) and .conveniently expressing abstraction from future decisions by resorting to non-determinism (an advantage not easily found in the imperative or functional approaches) [VH93].

3.4 Fork-Relational Software Development

Now, let us briefly examine the roe that these concepts canm play n the context of software development [HV90].

The development of a software artifact for a given application generally goes through several intermediate stages, involving requirement specification, reification, implementation, optimization, etc. The objects involved in each stage are distinct from an epistemological viewpoint, but they purport to describe relations, and the relationship among them is refinement-like.

An *application* can be regarded as a pair $<\theta,q>$, where θ describes the data that are to be processed and q the desired input-output behavior. Consider now an *program p* with *precondition* ϕ where p is guaranteed to halt. If $<\phi,p>$ refines $<\theta,q>$ then we have an *acceptable program*, in that it halts for any required data producing an output in accordance with the desired behavior.

Similarly, the relationships of *correct implementation* (between implemented program and abstract program), of *total correctness* (between

abstract program and formal specification), and of *adequate formalization* (between formal specification and application concept) can be regarded as refinement-like. By transitivity, we can conclude that a correct implementation of a total correct program with respect to an adequate formalization is an acceptable program.
This approach does not solve the problems involved in validation by testing. For they are akin to the epistemological questions involved in the confirmation of scientific theories. But it does provide a precise framework where such issues can be addressed [HV90].

4. CONCLUSIONS

We have presented an alternative basis for relational programming calculi, obtained by extending abstract relational algebras with a fork operator, and examined some fundamental issues of its use in reasoning about programs. From the theoretical side, we have showed that our calculus has the crucial features of expressiveness of first-order logic and representability, which makes it sound and complete. From the more practical side, we have argued for the adequacy of our calculus for reasoning about programs on the basis of general arguments.
Operators, such as fork as well as direct product and projections, have been used by other researchers. But, they appear to have overlooked some important features due to the non-monolithic character of projections (in our approach). A crucial one is the fact that our new algebra has the expressive power of first-order logic. As algebras of first-order logic, our fork-relational algebras present an important advantage over "classical" ones: it is finite extension of Boolean algebras that has a finite axiomatization. Also, some usual alternative axiomatizations for fork and projections can be derived from ours.
Another important feature of our fork-relational algebras is their representability: any model of the abstract calculus is isomorphic to a proper one, consisting of binary relations of input-output pairs. This means that we can use input-output intuition provided by the proper algebras while retaining the advantages of the abstract calculus without individual variables for algebraic manipulations.
We thus have a sound and complete calculi, with the power of classical first-order logic, for reasoning about specifications and programs. On the other hand, the standard meaning of our fork-relational operations is computational, which allows us to regard terms with only algorithmic operations as programs, their actual conversion into a usual programming language being straightforward.

The adequacy of our calculus for program derivation has been extensively illustrated elsewhere by means of case studies [DB93, FW92, FAN93, Fri93, HVE90, HV91, HBS93, VHB92], where more references and comparisons with other approaches can be found.

REFERENCES

[BT92] Backhouse, R. & Tormenta, P., "Polynomial Relators" in Möller, B., Partsch, H. A. & Schuman, S. A., *Formal Program Development*, Springer, 1992.

[BHV92] Baum, G., Haeberer, A.M. & Veloso, P.A.S., "On the Representability of the ∇-Abstract Relational Algebra" *IGPL Newsletter*, vol. 1, no. 3.

[BU78] Bednarek, A. R. & Ulam, S. M., "Projective Algebra and the Calculus of Relations" *J. of Symbolic Logic*, vol. 43, no. 1, 56 - 64, March 1978.

[BHSV93] Berghammer, R., Haeberer, A.M., Schmidt, G., & Veloso, P.A.S., "Comparing two Different Approaches to Products in Abstract Relation Algebras" Proc. AMAST'93. Springer. 1993.

[BS89] Berghammer, R. & Scmidt, G., "Symmetric Quotients and Domain Constructors" *Information Processing Letters* vol. 33, 163 - 168, 1989.

[BZ86] Berghammer, R. & Zierer, H., "Relational Algebraic Semantics of Deterministic and Non-deterministic Programs" *Theoretical Computer Science*, vol. 43, 123 - 147, 1986.

[CT48] Chin, L. & Tarski, A., "Remarks on Projective Algebras" *Bull. of the American Mathematical Society* , vol. 54, 80 - 81, 1948.

[CT51] Chin, L. H. C. & Tarski, A., "Distributive and Modular Laws in the Arithmetic of Relation Algebras" in *University of California Publications in Mathematics*, of California, U., Ed. University of California, 1951, pp. 341 - 384.

[deM92] deMoor, O., "Categories, Relations and Dynamic Programming"in Möller, B., Partsch, H. A. & Schuman, S. A., *Formal Program Development*, Springer, 1992.

[DB93] Duran, J. E., Baum, G., A., "Construcción Formal de Programas a partir de Especificaciones en un Cálculo de Relaciones Binarias Extendido", PUC-Rio, Res. Rept., MCC 5, 1993.

[EFT80] Ebbinghaus, H. D., Flum, J., and Thomas, W., *Mathematical Logic*. New York: Spriger-Verlag, 1980.

[EU46] Everett, C. J. and Ulam, S. M., "Projective Algebra," *American Journal of Mathematics*, vol. 68, 77 - 88, 1946.

[FW92] Frias, M. & Wachenchauzer, R., "Haeberer-Veloso's Relational Algebra as a Language to Express and Optimise Queries to a Relational Data Base" in *44th Meeting of the,* vol. Document IFIP Working Group 2.1 Algorithmic Languages and Calculi, 1992.

[FBHV93] Frias, M., Baum, G., Haeberer, A. M. & Veloso, P., A., S., "A Representation Theorem for Fork-Algebras", PUC-Rio Res. Rept., MCC 29, 1993. To appear in *Bull. Section of Logic, Polish Acad Sciences.*

[FAN93]Frias, M., Aguayo, N., & Novak, Beatriz., "Development of Graph Algorithms with the ∇-Extended Relation Algebra", Proceedings of the XIX Conferencia Latinoamericana de Informática, pp 529-554. Buenos Aires, Argentina. Agosto, 1993.

[Fri93] Frias, M., "The ∇-Extended Relation Algebra as a Deductive and Object Oriented Database Language", Technical Report. Department of Informatics. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires, Argentina. 1993.

[HV90] Haeberer, A.M. & Veloso, P.A.S., "Why Software Development is inherently non-monotonic: a formal justification Trappl, R. (ed) *Cybernetics and Systems Research*, World Scientific Publ., London, 1990, ; pp. 51 - 58

[HV91] Haeberer, A.M. & Veloso, P.A.S., "Partial Relations for Program Derivation: Adequacy, Inevitability and Expressiveness" in B. Möller(ed.) *Constructing Programs From Specifications , North-Holland*, 1991, pp. 319 - 371.

[HVE90] Haeberer, A.M., Veloso, P.A.S., & Elustondo, P., "Towards a Relational Calculus for Software Construction" in *41st Meeting of the*, vol. Document IFIP Working Group 2.1 Algorithmic Languages and Calculi, Chester - England, 1990.

[HBS93] Haeberer, A.M., Baum, G. A. & Schmidt, G., "On the Smooth Calculation of Relational Recursive Expressions out of First-Order Non-Constructive Specifications involving Quantifiers" in *Proc. FMP'93 International Conference on Formal Methods on Programming and its Applications.* T o appear in LNCS Springer-Verlag, 1993

[Hal62] Halmos, P. R., *Algebraic Logic*. New York: Chelsea Publishing Company, 1962.

[HM74] Henkin, L. and Monk, J. D., "Cylindric Algebras and Related Structures" in *Proceedings of the Tarski Symposium*, vol. 25 American Mathematical Society, 1974, pp. 105 - 121.

[Hen77] Hennie, F., "Introduction to Computability" Addison-Wesley. Reading, Massachusetts. 1977.

[HH86] Hoare, C. A. R. & He, J., "The Weakest Presepecification" *Fundamenta Informatica*, vol. 4, no. 9, pp. 51 - 54 and pp. 217 - 252, 1986.

[JT51] Jónsson, B. & Tarski, A., "Boolean Algebras With Operators part I" *American J. of Mathematics*, vol. 73, 891 - 939, 1951.

[JT52] Jónsson, B. & Tarski, A., "Boolean Algebras With Operators part II" *American J. Mathematics*, vol. 74, 127 - 162, 1952.

[Lyn50] Lyndom, R., "The Representation of Relational Algebras" *Annals of Mathematics (series 2)*, vol. 51, 707 - 729, 1950.

[Mad83] Maddux, R., "A Sequent Calculus for Relation Algebras" *Annals of Pure and Apply Logic*, vol. 25, 73 - 101, 1983.

[Mad91] Maddux, R., "The Origin of Relation Algebras in the Development and Axiomatization of the Calculus of Relations" *Studia Logica*, vol. L, no. 3-4, 412 - 455, 1991.

[Möl92] Möller, B., "Derivation of Graph and Pointer Algorithms" n Möller, B., Partsch, H. A. & Schuman, S. A., *Formal Program Development*, Springer, 1992.

[Ném91] Németi, I., "Algebraization of Quantifier Logics, an Introductory Overview" *Studia Logica*, vol. L, no. 3-4, 485 - 569, 1991.

[Roe72] Roever, W. P. de, "A Formalization of various Parameter Mechanisms as Products of Relations within a Calculus of Recursive Program Schemes" in *Theorie des Algorithmes, des Languages et de la Programmation*, 1972, pp. 55 - 88.

[SS85] Schmidt, G. & Ströhlein, T., "Relation Algebras: Concept of Points and Representability" *Discrete Mathematics*, vol. 54, 83 - 92, 1985.

[SS93] Schmidt, G. & Ströhlein, T., *Relations and Graphs, Discrete Mathematics for Computer Science*. Springer, 1993.

[Tar41] Tarski, A., "On the Calculus of Relations" *J. of Symbolic Logic*, vol. 6, 73 - 89, 1941.

[TT52] Tarski, A. & Thompson, F. B., "Some General Properties of Cylindric Algebras" *Bull. of the American Mathematical Society*, vol. 58, 65, 1952.

[VH91] Veloso, P.A.S. & Haeberer, A.M., "A Finitary Relational Algebra for Classical First-Order Logic" *Bull. Section of Logic ,Polish Acad Sciences*, vol. 20, no. 2, 52 - 62, 1991.

[VH93] Veloso, P.A.S. & Haeberer, A.M., "Fork Algebras and Program Construction" PUC-Rio , Res. Rept. 1993.

[VHB92] Veloso, P.A.S., Haeberer, A.M., and Baum, G., "Formal Program Construction within an Extended Calculus of Binary Relations" PUC-Rio Res. Rept. 1992.