PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 32/93

On Fork Algebras and Program Derivation

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 32/93 ISSN 0103-9741
Editor: Carlos J. P. Lucena December, 1993

On Fork Algebras and Program Derivation *

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 PROGRAM DERIVATION

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

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

PUCRioInf MCC 32/93

Abstract

Fork algebras provide a useful basis for relational calculi for program derivation; 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 discuss and illustrate their inportance for program derivation.

Key words:

Programming calculi, Program derivation, Relational Calculi, Algebras of Logic, Pairing Algebras, Finite Axiomatizability, Expressiveness. Representability.

Resumo

Álgebras de "fork" fornecem uma base apropriada para cálculos relacionais para derivação de programas: elas aparecem como extensões das álgebras relacionais com um novo operator de bifurcação, "fork", que permite 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 e ilustra sua inportância para derivação de programas.

Key words:

Cálculos de Programação, Derivação de programas, Cálculos relacionais, Álgebras da Lógica, Álgebras de Pares, Axiomatizabilidade Finita, Expressividade. Representabilidade.

ON FORK ALGEBRAS AND PROGRAM DERIVATION

Paulo A. S. Veloso[1]
and
Armando M. Haeberer

Pontifícia Universidade Católica do Rio de Janeiro
Departamento de Informática
Rua Marquês de São Vicente 225 – 22453 Rio de Janeiro – RJ – Brazil

Abstract

Fork algebras provide a useful basis for relational calculi for program derivation; 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 discuss and illustrate their inportance for program derivation.

Keywords: Programming calculi, Program derivation, Relational Calculi, Algebras of Logic, Pairing Algebras, Finite Axiomatizability, Expressiveness. Representability.

1. INTRODUCTION

Fork algebras provide a useful basis for relational calculi for program derivation. In this paper we examine some fundamental issues concerning fork algebras and their use in program derivation. Fork algebras arise as extensions of relational algebras with a new operator, called fork, which enables the introduction, by definition, of projections. The abstract calculus of fork algebras manipulates fork-relational terms without variables, free or bound, over individuals. This calculus provides our formalism for program derivation.

Two basic issues concerning a formalism for program derivation concern its formal aspects (such as soundness and completeness) and its adequacy for reasoning and deriving programs from 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

---

[1] E-mail armando@inf.puc-rio.br or veloso@inf.puc-rio.br. Fax: +55 21 511-5645. Telex: +55 2131048 PUCR BR

The structure of this paper is as follows. Sections 2 and 4 address fundamental mathematical issues, section 3 connects them to program derivation, and sections 5 and 6 illustrate the use of our formalism in some concrete issues and examples of program derivation.

The next section reviews Boolean and relational algebras [Tar41, Jón52] and introduces fork algebras together with their theories and calculi. 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 program derivation. Section 4 contains the central mathematical results. In it we prove the fundamental results of expressiveness [Vel91a] and representability [Bau92, Fri93a] and go on to compare our algebras with other classical approaches to algebras of first-order logic [Eve46, Hal62, Hen74]; this comparison suggests that the main difference resides in the fact that quantifiers and projections in our approach are constructed, rather than primitive, monolithic concepts, which leads us to compare our approach to projections with some others [Ber91, Bac92]. Section 5 concerns the advantage of relations, over functions, as basic concept for program derivation, which is supported by some general arguments of de Moor [deM92] and illustrated by an example of program inversion suggested by the work of P. Harrison, handled by means of symmetric quotient [Ber89]. Section 6 examines some advantages accrued from representability, namely that we can use input-output intuition provided by the proper algebras while using the abstract calculus for reasoning without variables over individuals, and some refined uses of expressiveness. This is illustrated by four case studies, namely an analysis, and internalization, of Möller's composition and join [Möl92]; how an algebraic operation of substitution can be used to convert programs with filters (partial identities) into more program-like terms with if_then_else; how we can cope with complexity and universal quantifiers by a combination of proper-algebra intuition and abstract-calculus manipulations. Finally, section 7 provides some concluding remarks as well as indication of on-going and future work.

2. ABSTRACT AND PROPER FORK ALGEBRAS

In this section we introduce some basic definitions concerning fork algebras. These 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 AND RELATIONAL ALGEBRAS

Consider a set W. We have the usual Boolean (set-theoretical) operations on subsets of W, namely union $\cup$, intersection $\cap$ and complement $'$ (with respect to the universe W), in addition to special sets, namely the empty set $\emptyset$ and the universal set W. As usual, by a *field of sets* we mean a structure $\mathcal{F} = \langle F, \cup, \cap, ', \emptyset, W \rangle$, where F is a set of subsets of W, containing the empty set $\emptyset$ and the universal set W, which is closed under the Boolean operations *union* $\cup$, *intersection* $\cap$ and *complement* $'$. The abstract version of a field of sets is a *Boolean algebra*: a structure $\mathcal{B} = \langle \mathcal{B}, \vee, \wedge, ', 0, 1 \rangle$ satisfying the well-known axioms. Recall that on a Boolean algebra $\mathcal{B}$ we can define a binary relation $\subset$ giving a lattice

structure to it. Also, an element $\alpha \in \mathfrak{B}$ is called an *atom* iff for every element $x \in \mathfrak{B}$ such that $0 \neq x \subset \alpha$, we have $x = \alpha$; and Boolean algebra $\mathfrak{B}$ is *atomic* iff every non-zero element contains an atom. By Stone's representation theorem every Boolean algebra $\mathfrak{B}$ is isomorphic to a field of sets, where set-theoretical inclusion $\subseteq$ corresponds to the abstract partial order $\subset$.

Consider now a set $V \subseteq W \times W$, i. e. a set of ordered pairs of elements of W. Then, we have some natural Peircean (relation-theoretical) operations on subsets of V, namely transpose $^{-1}$ and composition $|$ (defined, respectively by means of $p^{-1} = \{\langle u,v\rangle \in W \times W : \langle v,u\rangle \in p\}$ and $p|q = \{\langle u,w\rangle \in W \times W : \langle u,v\rangle \in p \text{ and } \langle v,w\rangle \in q, \text{for some } v \in W\}$) and the special diagonal relation $\Delta = \{\langle u,v\rangle \in W \times W : u = v\}$. A *proper relation algebra* is a structure $\mathfrak{PRA} = \langle P, \cup, \cap, |, ', ^{-1}, \Delta, \varnothing, \mathcal{U}\rangle$, such that the reduct $\langle P, \cup, \cap, ', \varnothing, \mathcal{U}\rangle$ is a field of sets and P contains the diagonal Δ and is closed under composition $|$ and transpose $^{-1}$. The abstract version of a proper relation algebra is a *relational algebra*: a structure $\mathfrak{RA} = \langle A, +, \bullet, ;, ^{-}, ^{\smile}, 1, 0, \infty\rangle$ such that

(1) its reduct $\langle A, +, \bullet, ^{-}, 0, \infty\rangle$ is a Boolean Algebra with 0 its *zero* element, ∞ its *unit* element and $\subset$ is the lattice natural order called *relational inclusion.*

(2) $\langle A, ;, 1\rangle$ is a *monoid.*

(3) $r ; s \subset t \leftrightarrow \breve{r} ; \bar{t} \subset \bar{s} \leftrightarrow \bar{t} ; \breve{s} \subset \bar{r}$ *Schröder rule*

Notice that in relational algebras we can introduce a new constant (diversity $\wp$) and a new operation (relative sum $\oplus$) by definition: $\wp = \bar{1}$ and $r \dagger s = \overline{\bar{r} ; \bar{s}}$

We shall call a relation algebra *atomic* iff its Boolean reduct is so. We shall often be concerned with simple relational algebras, which are those satisfying

(4) $r \neq 0 \rightarrow \infty ; r ; \infty = \infty$ for every $r \in A$ *Tarski rule*

2.2. Abstract and Proper Fork Algebras

Let us now turn our attention to fork algebras, first the proper and then the abstract ones.

Consider a set U equipped with a binary injective operation $* : U \times U \rightarrow U$ (which we view as constructing pairs $[y,z]$of elements[2]) and a set $V \subseteq U \times U$. We now have another natural operation on subsets of V, namely (*proper*) *fork*, defined by $r \lhd s = \{\langle x,[y,z]\rangle : \langle x,y\rangle \in r \wedge \langle x,z\rangle \in s\}$. A *proper fork algebra* is a structure $\mathfrak{PFA} = \langle S, \oplus, \odot, |, \lhd, ', ^{-1}, \Delta, \mathbf{0}, \mathcal{U}\rangle$, where the reduct $\langle S, \oplus, \odot, |, ', ^{-1}, \Delta, \mathbf{0}, \mathcal{U}\rangle$ is a proper relation algebra such that S is closed under fork $\lhd$. The abstract version of a simple proper fork algebra is an *abstract fork algebra*: a structure $\mathfrak{RFA} = \langle A, +, \bullet, ;, \nabla, ^{-}, ^{\smile}, 1, 0, \infty\rangle$ such that:

(5) $\langle A, +, \bullet, ;, ^{-}, ^{\smile}, 1, 0, \infty\rangle$ is a *relational algebra*: satisfying

(6) $r \neq 0 \rightarrow \infty ; r ; \infty = \infty$ for every $r \in A$ *Tarski rule*

and the following axioms hold:

(7) $r \nabla s = (r ; (1 \nabla \infty)) \bullet (s ; (\infty \nabla 1))$

(8) $(r \nabla s) ; (t \nabla q)^{\smile} = (r ; \breve{t}) \bullet (s ; \breve{q})$

(9) $((1 \nabla \infty)^{\smile} \nabla (\infty \nabla 1)^{\smile}) \subset 1$

---

2 We will indistinctly use [x, y] or x * y for this operation.

(10) $t \neq 0 \wedge t \subset \big((1 \nabla \infty)^{\smile}; \infty\big) \nabla \big((\infty \nabla 1)^{\smile}; \infty\big) \rightarrow (\exists v)(\exists w)\big(0 \neq \big((1 \nabla \infty)^{\smile}; v\big) \nabla \big((\infty \nabla 1)^{\smile}; w\big) \subset t\big)$

The idea behind these axioms will become clear with the introduction of some abbreviations, for first and second projections[3] and direct product

(11) $\pi = (1 \nabla \infty)^{\smile}$

(12) $\rho = (\infty \nabla 1)^{\smile}$

(13) $r \otimes s = \big((1 \nabla \infty)^{\smile}; r\big) \nabla \big((\infty \nabla 1)^{\smile}; s\big)$

Now, axiom (7) expresses fork in terms of intersection and projections; axiom (8) gives a kind of distributivity, and axiom (10) says that every non-null relation contained in a direct product contains a non-null direct product, i.e., $t \neq 0 \wedge t \subset \infty \otimes \infty \rightarrow (\exists v)(\exists w)(0 \neq v \otimes w \subset t)$.

Notice that, because of Tarski 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.

2.3. 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 algebras. The idea is considering (unsorted) first-order logic and extending it (conservatively) with concepts pertaining to relations. First view it as single-sorted language $\mathcal{L}$, where the only sort is sort $\imath$ (of individuals). Now, consider a new language, called $\mathcal{ALBR}$ (*Abstract Language of (Binary)Relations*) with sort $\imath$ (of binary relations over individuals) with equality, together with operations and constants for sort $\imath$, namely 0 (for the null relation), ∞ (for the universal relation), 1 (for the identity relation), $\wp$ (for the diversity relation), together with + (sum), • (intersection), $^{-}$ (complement), $^{\smile}$ (converse), (relative sum †) and ; (relative product)[4]. We extend $\mathcal{L}$ by adding $\mathcal{ALBR}$ together with a ternary predicate symbol _(-,-) over sorts $\imath\,\imath\,\imath$. Call this language $\mathcal{ELBR}$ (*Elementary Language of (Binary)Relations*) Now, given variables x and y, over sort $\imath$ and terms p and q, over sort $\imath$, we have new atomic formulas of the form $p \approx q$ and $p(x, y)$ (which we also write as $x\,p\,y$). Finally add the following axioms [Tar41], with implicit universal quantification over relation variables:

(14) $(\forall x)(\forall y)(x\,\mathcal{U}\,y)$

(15) $(\forall x)(\forall y)(\neg x\,0\,y)$

(16) $(\forall x)(x\,1\,x)$

(17) $(\forall x)(\forall y)(\forall z)\big((x r y \wedge y\,1\,z) \rightarrow x r z\big)$

(18) $(\forall x)(\forall y)(x \bar{r}\, y \leftrightarrow \neg x r y)$

(19) $(\forall x)(\forall y)(x \wp y \leftrightarrow \neg x \Delta y)$

(20) $(\forall x)(\forall y)(x \breve{r}\, y \leftrightarrow y r x)$

(21) $(\forall x)(\forall y)(x r+s y \leftrightarrow x r y \vee x s y)$

(22) $(\forall x)(\forall y)(x r \bullet s y \leftrightarrow x r y \wedge x s y)$

(23) $(\forall x)(\forall y)\big(x r ; s y \leftrightarrow (\exists z)(x r z \wedge z s y)\big)$

(24) $r \approx s \leftrightarrow (\forall x)(\forall y)(x r y \leftrightarrow x s y)$

---

3 The reason for calling π and ρ *projections* will become clear later.

4 Notice that $\mathcal{ALBR}$ is the language of relational algebras.

We thus obtain a theory, called the *Elementary Theory of (Binary)Relations* (*ETBR*, for short). The new axioms purport to explain the meaning of the new relative operations by means of first-order formulas involving individuals. For instance, axiom (23) explains the meaning of relative product as composition of relations. Also, if we have equality between individuals, it is reasonable to strengthen axiom (17) to $x\ 1\ y \leftrightarrow x = y$, which defines 1 as the diagonal. In this sense, we say that this theory provides a standard meaning for the relational theoretic terms. More precisely, consider the first-order language with equality over sort i (of individuals), extended by the sort i and the ternary predicate symbol _(-,-), and call $\mathcal{SMBR}$ this sub-language of $\mathcal{LLBR}$. We see that: given a term p of sort r of $\mathcal{LLBR}$ and variables x and y, over sort i, there exists a formula $\phi(x,y)$ of $\mathcal{LLBR}$ such that $ETBR \vdash (\forall x)(\forall y)[p(x,y) \leftrightarrow \phi(x,y)]$. This formula $\phi(x,y)=S(p)$ will be called the *standard meaning* of relational term p.

Now, consider a model $\mathfrak{C}$ of *ETBR*. The realization of sort $\imath$ consists of elements which behave, in view of the standard meaning above, as sets of ordered pairs of elements of the realization of sort i. 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 languages of fork algebras. The situation corresponds to the one described above for algebras of relations. We have:

the first-order theory *TIO* (*Theory of Injective* (*pair-forming*) *Operation*), with equality, of the injective binary operation $*$;

the first-order language, with equality, $\mathcal{ALFR}$ (*Abstract Language of Fork Relations*), which is the extension of $\mathcal{ALBR}$ by a new binary operation from sorts $\imath\ \imath$ to $\imath$;

the first-order theory, with equality, *ETFR* (*Elementary Theory of Fork Relations*) in language $\mathcal{LLFR}$, which is the extension of *ETBR* by a new binary operation from sorts $\imath\ \imath$ to $\imath$, together with the new axiom (explaining fork) $(\forall x)\big(\forall y\big)\big(x\ r\ \nabla\ s\ y \leftrightarrow (\exists v)(\exists w)\big(y = (v*w) \wedge x\ r\ v \wedge x\ s\ w\big)\big)$;

the sub-language $\mathcal{SMFR}$ of $\mathcal{LLFR}$, which is the extension of $\mathcal{SMBR}$ by a new binary operation from sorts $\imath\ \imath$ to $\imath$;

As for the case of relations, we have:

a *standard meaning*, assigning to each fork term p of sort $\imath$ of $\mathcal{LLFR}$ and variables x and y, over sort i, a formula $\phi(x,y)=S(p)$, called the *standard meaning* of p, such that $ETFR \vdash (\forall x)(\forall y)[x\ p\ y \leftrightarrow \phi(x,y)]$;

the simple proper fork algebras are the reducts to language $\mathcal{ALFR}$ of the simple models $\mathfrak{C}$ of *ETFR*.

2.4. Some Concepts and Results

Let us review some concepts and results about relational and fork algebras.

First, recall that Schröder rule is equivalent to:

(25) $\breve{r}\,;\breve{s} \subset t \leftrightarrow \breve{s}\,;\breve{t} \subset r \leftrightarrow \breve{t}\,;\breve{r} \subset s$

(26) $(r\,;s)\bullet t = 0 \leftrightarrow (\breve{r}\,;t)\bullet s = 0 \leftrightarrow (t\,;\breve{s})\bullet r = 0$

(27) $(r\,;s)\bullet t \subset \big(r\bullet(t\,;\breve{s})\big)\,;\big(s\bullet(\breve{r}\,;t)\big)$ *Dedekind formula*

Also, one says that a relation r is *functional* iff $\breve{r}\,;r \subset 1$, *injective* iff $r\,;\breve{r} \subset 1$ (equivalently, $\breve{r}$ is functional), *surjective* iff $\breve{r}\,;\infty = \infty$, *bijective* iff it is injective and surjective, *total* iff $r\,;\infty = \infty$

(equivalently, $\breve{r}$ is surjective). It is also convenient to have some means for talking about domain and range of relations. We introduce the definitions $\mathscr{Dom}(r) = (r\,;\breve{r}) \bullet 1$ and $\mathscr{Ran}(r) = (\breve{r}\,; r) \bullet 1$. Notice that in a proper relation algebra these correspond to partial identities over the domain and range, respectively, and thus can be regarded as representing these sets as binary relations. Another usual representation of sets is by means of right ideals: relational terms of the form $r\,;\infty$ (also called conditions [Hoa86a, 86b] or vectors [Sch85a, 89, 93]). Notice that both representations of sets are equivalent in the strong sense that one can move between them via bijections defined by relational terms: given a vector $r\,;\infty$, we have the partial identity $(r\,;\infty)\bullet 1$ with $\mathscr{Dom}\big((r\,;\infty)\bullet 1\big) = \mathscr{Dom}(r\,;\infty)$[5]; and given a partial identity δ, we have a vector $\delta\,;\infty$ .with $\mathscr{Dom}(\delta\,;\infty) = \mathscr{Dom}(\delta)$. We will call *2* to $1 \nabla 1$[6]

As for fork algebras, we have already introduced, by definition, projections and Cartesian products. It is not difficult to prove that our projections are functional and present the behavior one would expect of projections. For instance, $\mathscr{Dom}(\pi) = \mathscr{Dom}(\rho) = 1 \otimes 1$ and $\mathscr{Ran}(\pi) = \mathscr{Ran}(\rho) = 1$; and they decompose Cartesian products: for non-null r and s, $(r \otimes s)\,;\pi = \big(1 \otimes \mathscr{Dom}(s)\big)\,;\pi\,;r$ and $(r \otimes s)\,;\rho = \big(\mathscr{Dom}(r) \otimes 1\big)\,;\rho\,;s$ [7]

Two central results about fork algebras will be proved in Section 4.

The first one concerns the expressive power of our fork relational terms vis-a-vis first-order languages. Recall that the Elementary Theory *ETFR* of Fork Relations assigns to each fork-relational term t its standard meaning: a formula $\mathcal{S}(t)=\phi(u,v)$ such that $ETFR \vdash (\forall u)(\forall v)[t\,(u,v) \leftrightarrow \phi(u,v)]$. Expressiveness provides an inverse for $\mathcal{S}$, in that it provides for each formula $\phi(v_1,\ldots, v_n)$ a term $\mathcal{T}(\phi)=t$ with standard meaning $\mathcal{S}(t)=\tau(u,v)$, such that $ETFR \vdash (\forall v_1)(\forall v_2)\ldots(\forall v_{n-1})(\forall v_n)\,[\,\phi(v_1,v_2,\ldots, v_n) \leftrightarrow \tau((v_1*(v_2*\ldots,(v_{n-1}*v_n)\ldots)),(v_1*(v_2*\ldots,(v_{n-1}*v_n)\ldots)))\,]$. Thus, we can say that the n-ary relation defined by formula $\phi(v_1,v_2,\ldots, v_n)$ is described by term t, for in any structure the former is the domain, and range, of the latter.

The second result is the analog of Stone's representation theorem. Representability asserts that any fork algebra is isomorphic to a proper fork algebra[8].

In the next section we will comment on the importance of these results for program derivation within our fork-relational approach.

3. ON PROGRAMMING WITH RELATIONS

We shall now examine some aspects underlying our relational approach to program derivation.

---

5 Notice that we are using $\mathcal{D}$ and $\mathcal{R}$ for the usual meaning of *domain* and *range* of a relation, while we use $\mathscr{Dom}$ and $\mathscr{Ran}$ for denoting *partial identities* on domains and ranges.

6 Notice that both *2* and its converse are functional relations [Fri93a].

7 Notice also that $(r \nabla s)\,;\pi = r \bullet (s\,;\infty)$, $(r \nabla s)\,;\rho = s \bullet (r\,;\infty)$ .and $\big((r \nabla s)\,;\pi\big) \nabla \big((r \nabla s)\,;\rho\big) = r \nabla s$

8 Both results together represent a quite important result since, expressiveness guarantee that fork algebras are algebras of first-order logic with equality while it is a consequence of representability the existence of a finitary axiomatization for abstract fork algebras. Therefore, abstract fork algebras, which will be introduced in Section 4, turn to be the first finitelly axiomatizable algebras of first-order logic with equality. Notice that as late as 1991 Maddux [Mad91] and Németi [Ném91] considered the problem of finding such finitazable algebras unsolved.

Program derivation, as program verification, involves a specification and a program. The specification describes some data type, with basic sorts, operations and predicates, together with the required behavior of the program. Notice that the assumption that we are given this data type is interpreted as we are free to use its basic symbols in a program.

We shall now explain why we can regard both specifications and programs as relations, and indicate some advantages of this viewpoint.

3.1. SPECIFICATIONS AS RELATIONS

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

Let us start by considering the given data type (say lists with sorts $\mathcal{List}$ and $\mathcal{Elm}$, operations such as `head`, `tail`, `cons`, and predicates such as `null`).

The information provided about the data type is given by its specification: a set of first-order sentences its axioms in a possibly many-sorted language.

Now, it is well-known that one can faithfully reduce many-sorted first-order logic to the usual unsorted (or single-sorted) version. This is done 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 $\delta_s = \{\langle u, u\rangle \in \mathcal{U} : u \in s\}$, so that $u \in s$ iff $\langle u, u\rangle \in \delta_s$. For instance, for the case of lists of elements, we consider two partial identities $\delta_{\mathcal{List}}$ and $\delta_{\mathcal{Elm}}$ corresponding to the sorts $\mathcal{List}$ and $\mathcal{Elm}$.

We also consider relation symbols corresponding to the given operations and predicates. For our list example we would have relation symbols *head*, *tail*, *cons* and *null* corresponding to `head`, `tail`, `cons`, and `null`. Notice that one can express the fact that the relations corresponding to operations are indeed functional $r\,;\breve{r}\,;r = r$ as well as information concerning domain and range ($r\,;\breve{r} = \delta_s$ expresses the fact that domain of r is s).

The axioms of the specification are first-order sentences, which, as mentioned above, can be assumed already reduced to their unsorted versions. We can express such axioms in a fork-relational manner. For instance, consider an axiom of lists like $(\forall e{:}\mathcal{Elm})(\forall x{:}\mathcal{List})$ `head(cons(e, x)) = e`. The relative product *cons* ; *head* is a relation whose domain consists of pairs of elements and lists (which corresponds to $\delta_{\mathcal{Elm}} \otimes \delta_{\mathcal{List}}$) and the right-hand side is the first component of the given pair (which can be extracted by a projection π). Thus, this axiom can be expressed by $(\delta_{\mathcal{Elm}} \otimes \delta_{\mathcal{List}})\,;\,cons\,;\,head = (\delta_{\mathcal{Elm}} \otimes \delta_{\mathcal{List}})\,;\,\pi\,;\,\delta_{\mathcal{Elm}}$. Similarly, an axiom like $(\forall x{:}\mathcal{List})\mathtt{null}(x) \vee (\exists e{:}\mathcal{Elm})(\exists y{:}\mathcal{List})x = \mathtt{cons}(e,y)]$ can be expressed by $\delta_{\mathcal{List}} = \delta_{\mathtt{null}} + (cons^{\smile}\,;cons)$. The latter axiom gives an inductive property of lists, still within first-order.

Notice that, by representing sets by means of their partial identities we can express "any set M including 0 and closed under successor contains the set of naturals" as $(\forall\delta)\big(\delta \in 1 \wedge \delta_{zero} \subset \delta \wedge \delta\,;succ \subset succ\,;\delta \rightarrow \delta_{Nat} \subset \delta\big)$.

Expressiveness guarantees that any first-order formula can be expressed by a fork-relational term, whose standard meaning (given by elementary theory) is equivalent to the given formula. In this sense, any

specification consisting of first-order axioms in a (possibly many-sorted) language can be expressed by a set of equations involving relational terms built from the corresponding relations.

For instance, consider an axiom of lists like $(\forall e{:}\mathcal{Elm})(\forall x{:}\mathcal{List})\mathrm{head}(\mathrm{cons}(e,x)) = e$. Moreover, the standard meaning maps $\subset$ to $\rightarrow$ in the sense that, with $\mathcal{S}(p)=\phi(u,v)$ and $\mathcal{S}(q)=\psi(u,v)$, $ETFR \vdash p \subset q$ iff the sentence $(\forall u)(\forall v)[\phi(u,v) \rightarrow \psi(u,v)]$ is logically valid.

Let us now consider the abstract calculus *ATFR*, dealing with formulas of *ETFR* without variables (free or bound) over individuals; thus *ATFR* is a sub-theory of *ETFR*. But, in view of Representability, we know that model $\mathfrak{A}$ of *ATFR* is isomorphic to a proper fork relation algebra $\mathfrak{B}$. Now, such a proper fork relation algebra $\mathfrak{B}$ is the reduct to the sort of relations of a model $\mathfrak{C}$ of *ETFR* Hence, every model $\mathfrak{A}$ of *ATFR* can be expanded to a model $\mathfrak{C}$ of *ETFR*; therefore the extension from *ATFR* to *ETFR* is conservative.

Now, 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 $s=\mathcal{S}(\sigma)$ and $t=\mathcal{S}(\tau)$, and $\sigma \vdash \tau$ iff $ETFR \vdash s \subset t$ iff $ATFR \vdash s \subset t$, the latter equivalence following from conservativeness.

Summing up, Expressiveness guarantees that first-order properties τ and (finite) specifications σ can be expressed by relational terms t and s, respectively. Moreover, τ is a consequence of σ iff $s \subset t$ can be derived within the abstract calculus of fork relations. This means that the first-order reasonings are exactly mirrored in the calculus of fork relations.

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*..

Therefore, reasoning within the abstract fork-relational calculus *ATFR* is both sound and complete with respect to standard first-order reasoning[9].

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

The language of the abstract fork-relational calculus *ATFR* relational calculus has symbols for constant relations as well as for operations on relations.

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: it is the formula $\phi(x, y)$ with variables x and y ranging over individuals, such that from *ETFR* one can derive $(\forall x)(\forall y)[t(x, y) \leftrightarrow \phi(x, y)]$. For instance, the standard meaning of term $r;s$ is the formula $(\exists z)(xrz \wedge zsy)$, and for $r \nabla s$ we have $(\exists v)(\exists w)(y = (v*w) \wedge xrv \wedge xsw)$.

---

9 This is one of the differences between our calculus and those of the Eindhoven group [Bac92] and of the Munich group [Sch85b, Zie83, Ber86, Ber91]. The difference of our calculus with all the other algebras of first-order logic with equality is representability and, therefore, finitiness.

The *algorithmic* symbols are all of the above, with the exception of complement ($\bar{}$) and relative sum ($\dagger$). The reason for such a name is as follows. Consider an effective domain $\mathcal{U}$, one where equality is effectively decidable and where the pair-forming operation $* : \mathcal{U} \times \mathcal{U} \to \mathcal{U}$ is effectively computable. Then, the constant relations $0, \infty, 1$ and $\wp$, 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 $u=u'$ and in such case computing $v * w$ and outputting $\langle u, v * w\rangle$.

Now, consider a term built from some effectively enumerable relations by means of these algorithmic symbols. Such a term denotes an effectively enumerable relation, and as such can be regarded as a (possibly non-deterministic) program over domain $\mathcal{U}$.

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.

There is a slightly stronger sense in which algorithmic relational terms describe programs. It comes from the natural correspondence between algorithmic operations (except, perhaps for intersection and converse) and programming-language constructs. For instance, relative product corresponds to sequentialization, in that they have the same standard meaning. Now, intersection does not quite follow this pattern, but it can be simulated in terms of other algorithmic operations since $r \bullet s = (r \nabla s)\,;\bar{2}$ (notice that $\bar{2}$ is 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 that shows 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 convers, 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).

As a simple example of the above correspondence, consider the double of a natural number. The recursive program `dbl(x) = if is_zero(x) = T then zero else succ(succ(dbl(pred(x))))` **end_if** and the relational term $dbl = \delta_{zero}\,;\, zero + \delta_{not\_zero}\,;\, pred\,;\, dbl\,;\, succ\,;\, succ$ correspond to each other. (Here δ_{zero} and δ_{not_zero} denote $\{\langle 0, 0\rangle\}$ and $\{\langle n, n\rangle : n \in Nat\}$ respectively, and can be obtained from

relations corresponding to `is_zero`, true and false). Notice, further, that we have a slightly simpler term for double, namely $dbl = \delta_{zero} + succ^{\smile} ; dbl ; succ ; succ$.

Concepts related to correctness are easily expressed and established in our relational setting. For instance, partial correctness of p is expressed by an inclusion and termination as a property of $\mathscr{D}om(p)$. In order to establish the termination of the above program dbl it suffices to prove $\delta_{zero} \subset \mathscr{D}om\,(p) \wedge \mathscr{D}om\,(p) ; succ \subset succ ; \mathscr{D}om\,(p)$.

3.3. Program Derivation with Relations

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 fork-relational 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.

Furthermore, relational reasoning within *ATFR* offers some conveniences for manipulation. First, it has the advantage of not involving (variables over) individuals (an advantage over usual programming or first-order reasoning, shared by the functional approach). Also, relational reasoning permit conveniently expressing abstraction from decisions still to be taken, by resorting, for instance to non-determinism (an advantage not easily found in the imperative or functional approaches).

We can express strategies, both general and global in fork-relational terms. Thus, one could in principle simulate derivations such as those done in CIP. Also, general problem-solving strategies, such as divide-and-conquer, can be expressed and used. Some examples are (some of the names are not standard): case division ($p = q' + q''$), decomposition ($p = q' \otimes q''$), interpolation ($p = q' ; q''$), reduction ($p = t ; q ; r$), reduction to decomposition ($p = t ; q' \otimes q'' ; r$), and binary divide-and-conquer ($p = e + s ; (p \otimes p) ; m$).

4. Fork Algebras as Finitary Algebras of First-Order Logic

In this section we examine two central results about fork algebras: Expressiveness and Representability. Their importance for our approach to program derivation has been commented upon in the preceding section.

Expressiveness guarantees that the expressive power of our fork relational terms is the same of first-order formulas: any n-ary relation defined by formula $\phi(v_1, v_2, \ldots, v_n)$ can be defined by term t, in that in any structure the former is the domain, and range, of the latter.

Representability is the analog of Stone's representation theorem: any fork algebra is isomorphic to a proper fork algebra.

These results show that our fork algebras provide a finitary algebra of first-order logic with equality[10].

---

10 The proof of expressiveness can be found in [Vel91b] and that of representability in [Fri93a]. Notice again that our algebra turn to be the first finitary algebra of first-order logic with equality.

4.1. EXPRESSIVENESS OF FORK ALGEBRA TERMS.

Consider a *first-order language* $\mathcal{L}$ with equality $\approx$, given by a set $P_{\mathcal{L}}$ of *predicate symbols* (we do not consider function symbols, since functions can, for our purposes, be replaced by their graphs). As usual, a *structure* $\mathfrak{B}$ for $\mathcal{L}$ consists of a *domain* B together with a *realization* $p^{\mathfrak{B}} \subseteq {}^{\alpha}B$, for each α-ary predicate symbol p in $P_{\mathcal{L}}$. This can be extended to assign a realization for each formula $\varphi \in \Phi_{\mathcal{L}}$, namely $\varphi^{\mathfrak{B}} = \{\langle b_1, \ldots, b_n\rangle: \mathfrak{B} \models \varphi[b_1, \ldots, b_n]\}$, where $\mathfrak{B} \models \varphi[b_1, \ldots, b_n]$ means, as usual, that the assignment of b_i to v_i, for $i = 1, \ldots, n$, satisfies φ in $\mathfrak{B}$ ([Ebb80]).

It is more convenient, however, to view a formula as defining a set of trees over B. Consider language $\mathcal{L}^*$, obtained from $\mathcal{L}$ by the addition of a new injective binary operation symbol. Now, given a structure $\mathfrak{B}$ for $\mathcal{L}$ we can obtain a structure $\mathfrak{B}^*$ for $\mathcal{L}^*$, whose domain consists of the finite trees over B, where the new operation symbol is realized as tree construction. Notice that $\mathfrak{B}$ is an elementary substructure of $\mathfrak{B}^*$. Thus, we can replace $\mathfrak{B}$ by $\mathfrak{B}^*$ because $\varphi^{\mathfrak{B}}$ is the restriction of the realization φ in $\mathfrak{B}^*$. Also, it is interesting to notice that in $\mathfrak{B}^*$ we can get by with formulas with a single free variable. In order to clarify this last remark let us introduce some notation to be also used later. We let $f(\varphi)$ be the set of variables with free occurrences in formula φ, and for a finite set Ψ of variables, we let $h(\Psi) = max\{i : v_i \in \Psi\}$. Consider a formula $\varphi \in \Phi_{\mathcal{L}}$ with $h(f(\varphi)) = n$; we associate with φ a formula ψ of $\mathcal{L}^*$ with $h(f(\psi)) = 1$ such that $\varphi^{\mathfrak{B}} = \{\langle b_1, \ldots, b_n\rangle: \mathfrak{B}^* \models \psi[(b_1*(b_2*\ldots(b_{n-1}*b_n)\ldots))]\}$.

Now, given such a structure $\mathfrak{B}^*$ for $\mathcal{L}^*$ we can expand it to a model $\hat{\mathfrak{B}}$ of *ETFR* with a binary relational constant $\hat{p}$ for each predicate symbol p in $P_{\mathcal{L}}$ realized by $\{\langle b, b\rangle: b \in p^{\mathfrak{B}}\}$. Notice that, if p is n-ary, then the standard meaning $\mathcal{S}^*(\hat{p})$ of $\hat{p}$ is the following formula of $\mathcal{L}^*$: $u \approx v \wedge (\exists x_1)(\exists x_2)\ldots(\exists x_{n-1})(\exists x_n)u \approx x_1*(x_2*\ldots(x_{n-1}*x_n)\ldots)$. The idea of expressiveness is showing how this correspondence can be extended so as to cover all formulas of $\mathcal{L}$. In doing this, it will be important to take into account some variants of a formula, such as substitutions and adding new innocuous variables; this will be taken care of in Lemmas A and B below.

We are now ready for our expressiveness result, which will show that terms of *ATFR* have the expressive power of first-order logic. This will be established by showing that any α-ary relation that can be defined by a first-order formula can also be defined by a closed term.

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

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

Basis: We distinguish three cases.

Case 1: φ is $v_1 \approx v_2$. Then, we set $\mathcal{T}(v_1 \approx v_2) = \breve{2}; 2$
Notice that $\mathcal{S}^*(\mathcal{T}(v_1 \approx v_2))$ is $(\exists x_1)(\exists x_2)[u \approx x_1*x_2 \wedge x_1 \approx x_2 \wedge v \approx x_1*x_2]$.

Case 2: φ is $p(v_1, \ldots, v_n)$ with $p \in P_{\mathcal{L}}$. Then, we set
$\mathcal{T}(p(v_1, \ldots, v_n)) := \hat{p}$

Case 3: φ is an atomic formula $\alpha(u_1, \ldots, u_m)$ obtained from one of the above $\alpha(v_1, \ldots, v_n)$ by means of a substitution σ. Then, we set

$\mathcal{T}(\alpha(u_1, \ldots, u_m)) := \breve{s}\ ;\ \mathcal{T}(\alpha(v_1, \ldots, v_n))\ ;\ s$

where $\breve{s}$ is a term, indicated in lemma B below, simulating σ.

Inductive step: We must distinguish three cases

If φ is $\neg\psi$, then we set $\mathcal{T}(\neg\psi) = \overline{\mathcal{T}(\neg\psi)} \bullet 1$.

Notice that $\mathcal{S}^*(\mathcal{T}(\neg\psi))$ is $u \approx v \wedge \neg\mathcal{S}^*(\mathcal{T}(\psi))$.

Now, let $h(f(\varphi)) = n$. In view of Lemma A below, it suffices to consider φ' of the form $\varphi \wedge v_1 \approx v_1 \wedge \ldots \wedge v_n \approx v_n$.

If φ is $\psi \vee \theta$,

then φ' is equivalent to $\psi' \vee \theta'$, where ψ' is $\psi \wedge v_1 \approx v_1 \wedge \ldots \wedge v_n \approx v_n$ and similarly for θ'. Lemma A applied to the inductive hypothesis gives $\mathcal{T}(\psi')$ and $\mathcal{T}(\theta')$. We then set $\mathcal{T}(\psi \vee \theta) = \mathcal{T}(\psi') + \mathcal{T}(\theta')$

If φ is $(\exists v_i)\psi$, where $h(f(\psi)) = n$.

clearly φ' is equivalent to $(\exists v_n)\psi' \wedge v_n \approx v_n$, where ψ' is obtained from ψ by the substitution σ that interchanges v_i and v_n. So, Lemma A below, applied to the inductive hypothesis, gives $\mathcal{T}(\psi')$. Then, we set:

$\mathcal{T}((\exists v_n)\psi') = (1 \nabla \infty)\ ;\ \mathcal{T}(\psi')\ ;\ (1 \nabla \infty)^{\smile}$

whence Lemma A will give $\mathcal{T}(\varphi)$.

Q. E. D.

Lemma A For every $n \geq 0$, there exists a term d_n, such that, for every formula φ with $h(f(\varphi)) = n$, if φ' is the formula $\varphi \wedge v_1 \approx v_1 \wedge \ldots \wedge v_{m+n} \approx v_{m+n}$, one can have $\mathcal{T}(\varphi') = \mathcal{T}(\varphi) \otimes e_m$ and $\mathcal{T}(\varphi) = \breve{d}_n\ ;\ \mathcal{T}(\varphi')\ ;\ d_n$, where $e_1 = 1$ and $e_m = 1 \otimes e_{m-1}$

Proof

Trivial by induction see [Vel91b].

Lemma B Given a substitution σ on $\{1, \ldots, n\}$, there exists a term $s(\sigma)$, such that for any formula φ with $h(f(\varphi)) = n$, if $\sigma(\varphi)$ is the formula obtained by applying σ to φ, then one can take $\mathcal{T}(\sigma(\varphi)) = \breve{s}(\sigma)\ ;\ \mathcal{T}(\varphi)\ ;\ s(\sigma)$.

Proof Trivial, see [Vel91b] and the discussion on the construction of non-atomic cylindrifications below.

Notice that in the proof of Expressiveness we explicitly define the translation $\mathcal{T}$ for the existential quantifier just for the case of the quantification on variable v_n. For quantifying over other variables we rely on terms d_n and $s(\sigma)$ for respectively adjusting the number of variables and producing an alphabetical transformation of the original formula. This seems to be very close to the way Polyadic Algebras treat the same problem, but there is an important difference, namely in our case this trick is an artifact of the proof, while in the former it is inherent to the algebra itself. Notice also that both d_n and $s(\sigma)$ are constructed within *ATFR*.

By using a somewhat two-dimensional notation we can express, for instance, $\mathcal{T}((\exists v_1)(\exists v_3)\psi(v_0, v_1, v_2, v_3, v_4))$ as,

$$\underbrace{\begin{pmatrix} \pi;\pi \\ \pi;\pi;\begin{pmatrix}\pi;\pi\\ \nabla\\ \rho\end{pmatrix} \\ \nabla \\ \rho \end{pmatrix}}_{ds} ;\mathcal{T}\big(\psi(v_0,v_1,v_2,v_3,v_4)\big);\underbrace{\begin{pmatrix} \pi;\pi \\ \pi;\pi;\begin{pmatrix}\pi;\pi\\ \nabla\\ \rho\end{pmatrix} \\ \nabla \\ \rho \end{pmatrix}}_{ds^{\smile}}$$

where the pair of terms $\langle ds, ds^{\smile}\rangle$ is the construction, within $ATFR$, of the cylindrification $c_1c_3\big(\psi^H(v_0,v_1,v_2,v_3,v_4)\big)$. We will comment some more on this later on.

4.2. Representability of Fork Algebras.

An extensive development of the arithmetic of fork algebras can be found in [Fri93a] and [Dur93].

We now consider a simple, atomic fork algebra $\mathbf{A}$ and indicate the proof that it is isomorphic to a proper fork algebra.

First, as proved in [Bau92][11] the atoms of $\mathbf{A}$ are functional. By a result of [Jón52], this is a necessary and sufficient condition for the representability of relational algebras. Thus, the relational reduct $\langle A,+,\bullet,;,^{-},^{\smile},0,\infty,1\rangle$ is isomorphic to a proper relation algebra. It remains to show that fork is properly represented. An outline of the proof is as follows.

Given such a simple, atomic fork algebra $\mathbf{A}$, we first introduce an algebra $\mathfrak{P}=\langle \mathscr{P}(C\times C),\oplus,\odot,|,\lhd,{}',{}^{-1},\varnothing,C\times C,\Delta\rangle$ over the atoms of the identity (i.e. $C=\{\alpha\in\mathscr{A}:\alpha\subset 1\}$), where $\lhd$ is defined as: $(\forall\alpha)(\forall\beta)(\forall\gamma)(\alpha\, r\lhd s\,\beta\otimes\gamma\leftrightarrow\alpha\, r\,\beta\wedge\alpha\, s\,\gamma)$ and $\otimes$ stands for the usual direct product on $\mathbf{A}$.

We then prove that

$r\neq 0\rightarrow\pi;r\neq 0\wedge\rho;r\neq 0$,

$r\otimes s=0\leftrightarrow r=0\vee s=0$,

$r\otimes s\neq 0\rightarrow\breve{\pi};(r\otimes s);\pi=r$,

$r\otimes s\neq 0\rightarrow\breve{\rho};(r\otimes s);\rho=s$,

$(r\otimes s=t\otimes q)\wedge(r\neq 0)\wedge(s\neq 0)\wedge(t\neq 0)\wedge(q\neq 0)\rightarrow(r=t)\wedge(s=q)$,

$\alpha\in\mathscr{A}\wedge\beta\in\mathscr{A}\leftrightarrow\alpha\otimes\beta\in\mathscr{A}$ and

"$\otimes$" is injective,

thereby establishing that $\mathfrak{P}$ is indeed a proper fork algebra.

Then, following the idea of Theorem 4.21 in [Jón52], we define a function from the universe $\mathcal{U}$ of $\mathbf{A}$ into that of $\mathfrak{P}$ as follows

$$\mathscr{H}:\mathcal{U}\rightarrow\mathscr{P}(C\times C)$$
$$\mathscr{H}(r)=\{\langle\mathscr{D}(\alpha),\mathscr{R}(\alpha)\rangle\in C\times C:\alpha\in\mathscr{A}(r)\}$$

---

11 Also see [Hae93a, 93b] and [Fri93a]

$$\mathscr{D}:\mathscr{A}\to C$$

$$\mathscr{D}(\alpha)=\begin{cases}\alpha;\breve{\alpha} & \text{if}\quad \breve{\alpha}\not\subset\infty\nabla\infty\\ (\alpha_1;\breve{\alpha}_1)\otimes(\alpha_2;\breve{\alpha}_2) & \text{if}\quad \breve{\alpha}=\alpha_1\nabla\alpha_2 \text{ for some } \alpha_1,\alpha_2\in\mathscr{A}\end{cases}$$

$$\mathscr{R}:\mathscr{A}\to C$$

$$\mathscr{R}(\alpha)=\begin{cases}\breve{\alpha};\alpha & \text{if}\quad \alpha\not\subset\infty\nabla\infty\\ (\breve{\alpha}_1;\alpha_1)\otimes(\breve{\alpha}_2;\alpha_2) & \text{if}\quad \alpha=\alpha_1\nabla\alpha_2 \text{ for some } \alpha_1,\alpha_2\in\mathscr{A}\end{cases}$$

The definition of this function is based on two auxiliary functions. The idea underlying the latter is "inspecting the shapes of the atoms in the domain and range", as indicated in the figure below.

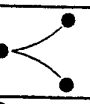
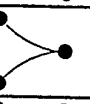
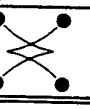
$r\not\subset\infty\nabla\infty\wedge\breve{r}\not\subset\infty\nabla\infty$	
$r=s;\breve{s}\subset 1$	
$r=\breve{s};s\subset 1$	
$r=s\nabla t$	
$r=(s\nabla t)^{\smile}$	
$r\subset\infty\nabla\infty\wedge\breve{r}\subset\infty\nabla\infty$	

Figure 4.2.1

Thus, we distinguish the 4 cases ($\not\subset$, $\not\subset$), ($\not\subset$, $\otimes$), ($\otimes$, $\not\subset$) and ($\otimes$, $\otimes$) in Figure 4.2.2.

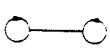
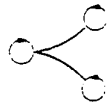
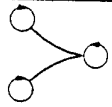
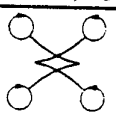
		$\alpha\not\subset\infty\nabla\infty$	$\alpha=\alpha'_1\nabla\alpha'_1$	α
		$\breve{\alpha};\alpha$	$\breve{\alpha}'_1;\alpha'_1$ $\otimes$ $\breve{\alpha}'_2;\alpha'_2$	$\mathscr{R}(\alpha)$
$\breve{\alpha}\not\subset\infty\nabla\infty$	$\alpha;\breve{\alpha}$	 Case $\not\subset\not\subset$	 Case $\not\subset\otimes$	
$\breve{\alpha}=\alpha_1\nabla\alpha_1$	$\alpha_1;\breve{\alpha}_1$ $\otimes$ $\alpha_2;\breve{\alpha}_2$	 Case $\otimes\not\subset$	 Case $\otimes\otimes$	
α	$\mathscr{D}(\alpha)$			

Figure 4.2.2

Finally, we prove that $\mathscr{H}$ is an injective homomorphism from $\mathfrak{A}$ into $\mathfrak{P}$ mapping ∞ to $\{\langle c, d\rangle : c, d \in C\}$, whence, an isomorphism from $\mathfrak{A}$ into a sub-algebra of $\mathfrak{P}$. We thus establish our representation theorem[12].

Theorem (Representability) Every simple atomic fork algebra is isomorphic to a proper algebra fork algebra.

4.3. Classical Algebras of First-Order Logic

Our fork-relational calculus provides a finitary algebra of first-order logic. We shall now briefly examine this role played by our calculus and compare it with other, classical approaches to algebras of first-order logic. A more detailed comparison can be found in [Hae93a, 93b].

The idea of algebra logic has been quite attractive, since Boole started developing a Boolean calculus for propositional logic. It seems natural to try to obtain an algebra for first-order logic (with equality) by extending Boolean Algebras with some operations so as to cope with quantifiers (and equality).

The attractive aspect of Abstract Relational Algebra for underlying a programming calculus resides in its absence of variables over individuals, the same aspect that led *functional languages* and functionally based programming calculi to deserve so much attention. This absence of variables is due to the fact that the extension to Boolean Algebra necessary for constructing Abstract Relational Algebras is Peircean, i.e. finitary.

B. Jónsson and A. Tarski [Jón51] introduced the concept of *Boolean Algebras with Operators* as extensions of Boolean Algebras with some, not necessarily finitely many, operations that are continuous with respect to the lattice structure of the Boolean algebra. In fact, they show that Abstract Relational Algebras are Boolean Algebras extended with just two operators, namely, ; and $\breve{}$.

Other Boolean Algebras with Operators aiming at algebras of logic are Cylindric Algebras of Tarski et al [Chi48, Tar52, Hen74], and the Polyadic Algebras of Halmos [Hal62]. The latter concept provides an algebra for pure first-order logic, and the former an algebra for first-order logic with equality (due to the diagonal elements).

We should notice that both Cylindric Algebras and Polyadic Algebras abandon the Peircean-like way of extending $\langle \mathfrak{R}, +, \bullet, -, 0, \infty\rangle$ in favor of a more direct attack to the problem. Both approaches maintain the idea of *Boolean Algebra*, but, produce theories over sequences of infinite length (restricted by a local finiteness condition) by imposing an infinite arity. Tarski et al. choose to represent the existential quantifier by infinitely many cylindrifications c_ζ (one for each variable v_ζ), and equations $v_\zeta = v_\xi$ by means of a doubly infinite sequence of diagonal elements $\delta_{\zeta\xi}$. Halmos, on the other hand, represents the existential quantifier by means of a single function $\exists$ but introduces infinitely many transformations on the index set J. We should notice that both Cylindric and Polyadic Algebras are laden with a syntactical artifact, in that the former, by means of each of the "ζ-axis", and the latter, by means of the index set J, have hidden "names" for variables over individuals.

---

12 A complete and detailed proof of the Representation Theorem can be found in [Fri93a]

The concept of *projections* and of some kind of *product* in connection with algebras of first-order logic appears as early as 1946 with the seminal work of Everett and Ulam on *Projective Algebra* [Eve46]. Everett and Ulam (and, later, Bednarek and Ulam [Bed78]) define Projective Algebras as extensions of Boolean Algebra with three fundamental operations: two projections p_1 and p_2 and a product. Apparently, however, the power of their combination with Peircean operations, for the construction of a Relational Algebra over locally finite trees, failed to be recognized. Later on, De Roever [Roe72], Schmidt and Ströhlein [Sch85b], Zierer [Zie83], Berghammer and Zierer [Ber86], Berghammer [Ber91], and Backhouse et al. [Bac92][13], introduced product and projections, as data types or as operations. They appear to have been guided categorical viewpoint, which may have "masked" the non-fundamental character of projections.

Our approach reverts to the idea of extending Boolean Algebras with Peircean-like Operators. We have an extension of Abstract Relational Algebra with one operator, namely fork. Here, projections and product are not fundamental, but are constructed by means of the basic operators, which accounts for the power exhibited. This is why, with our finitely many operations, we can construct infinitely many terms for the quantifiers as well as for the substitutions, as indicated in the proof of Expressiveness and the accompanying lemmas.

4.4. Comparison with other Approaches to Projections and Product.

Notice that our axiomatization of abstract fork algebras allows the introduction of projections and products as defined concepts. We shall now briefly compare this approach to two other approaches to projections and products, namely that of the Munich group[14] and the one due to the Einhoven group (Backhouse et al.). It is interesting to bear in mind that our approach and these two differ in one fundamental aspect. While they directly axiomatize properties deemed essential to projections and product, we construct terms that exhibit their behavior. We shall show that the axioms of both these approaches can be derived from our axiomatization by relying on the definitions (11) and (12), of projections, and (13), of direct product[15].

First, let us examine the Munich group axiomatization. It consists of the following axioms (in our own notation)

M_0 : $\breve{\pi};\pi = 1$, M_1 : $\breve{\rho};\rho = 1$, M_2 : $(\pi;\breve{\pi})\bullet(\rho;\breve{\rho}) = 1$, M_3 : $\breve{\pi};\rho = \infty$ and $\breve{\rho};\pi = \infty$[16],

M_4 : $r \nabla s = (r;\breve{\pi})\bullet(s;\breve{\rho})$

We now indicate how these axioms can be derived from our axiomatization. First, in view of definition (11), of first projection, M_0 can be written as $(1 \nabla \infty);(1 \nabla \infty)^{\smile}$, whence axiom (8) yields $(1;\breve{1})\bullet(\infty;\breve{\infty}) = 1$. M_1 can be derived similarly. To derive M_2 notice that from our definitions, one has $1 \otimes 1 = ((1 \nabla \infty)^{\smile};1) \nabla ((\infty \nabla 1)^{\smile};1) = (1 \nabla \infty)^{\smile} \nabla (\infty \nabla 1)^{\smile}$, and the right-hand side can be rewritten successively as $((1 \nabla \infty)^{\smile};(1 \nabla \infty))\bullet((\infty \nabla 1)^{\smile};(\infty \nabla 1))$, by axiom (8), and then, by definitions (11) and (12), as $(\pi;\breve{\pi})\bullet(\rho;\breve{\rho})$. (Notice that. because we are dealing with homogeneous relations, we obtain equality with $1 \otimes 1$. which, in view of axiom (9), is included in the identity). M_3 can be derived in a

---

13 The idea of direct product and direct sum presented in this paper by Backhouse et al. was borrowed from the works of the Munich group.

14 Schmidt, Ströhlein, Zierer and Berghammer.

15 See also [Ber93].

16 Actually one of them suffices for this axiomatization, we include both just for symmetry.

straightforward way from our definitions. Finally, given our definitions of projections, M_4 is our axiom (8).

Now, let us examine the axiomatization due to Backhouse et al. Its axioms can be derived from ours, a process that clarifies why it exhibits more axioms than necessary. As presented in [Bac92] it looks like, $B_0 : r \nabla s = (r;\breve{\pi}) \bullet (s;\breve{\rho})$, $B_1 : (\pi;\breve{\pi}) \bullet (\rho;\breve{\rho}) \subset 1$, $B_2 : (r \nabla s);(t \nabla q)^{\smile} = (r;\breve{t}) \bullet (s;\breve{q})$, $B_3 : \mathcal{D}(\pi) = \mathcal{D}(\rho)$

In view of definitions (11) and (12), B_0 is our axiom (7), and B_1 follows from axioms (8) and (9), and B_2 is our axiom (8).

As for B_3 , it can be derived immediately from M_2 and the definition of $\mathscr{D}om$, because $\mathcal{D}(r) = \mathcal{D}(s) \leftrightarrow \mathscr{D}om(r) = \mathscr{D}om(s)$ (since 1_X is the identity restricted to set X).

Thus, the properties of projections and product considered essential by Munich and Einhoven groups can be derived from our axioms and fork algebras together with the definitions of projections and product in terms of basic operations.

5. On Relations versus Functions

Oege de Moor [deM92] gave three reasons for favoring relations instead of functions as the underlying formalism of a programming calculus, they have to do with partiality, non-determinism and latitude along a derivation. Let us briefly review them. First, consider partiality. Most functional calculi, such as the one of Bird and Meertens, deal with total functions for the sake of simplicity; but this restriction is too severe for some problems, such as those of dynamic programming. Next, consider non-determinism. One often has to deal with problems that are inherently non-deterministic, for instance in optimization. Next, consider latitude. One can begin a derivation from a functional specification with the goal of obtaining a deterministic program, but the very structure of the derivation can lead one into a relational environment. This is the case, for instance, with inverses of programs, since only bijective functions have inverses. One advantage of dealing with relations is that one can view functions as univalent relations, and relations always have converses. This advantage has two consequences: it simplifies many derivations and offers an extraordinary expressive power.

Formalisms based exclusively on functions must introduce a somewhat artificial artifact, like pseudoinverses instead of "real" inverses, for dealing with inverses of function. A *pseudoinverse* of a function $f : A \to B$ is a function $f^{-1\uparrow} : B \to \mathscr{P}(A)$, defined as $f^{-1\uparrow}(x) = \{y : f(y) = x\}$, i. e. its pre-image. But, if one wishes to derive a "deterministic program", such pseudoinverses should eventually be removed, for instance, by deriving a choice function which selects one of the elements of $\{y : f(y) = x\}$. We shall now indicate how this problem can be handled in our fork-relational approach. For this purpose it is convenient to review some notions about residuals and symmetric quotients.

Let s/r be the *left residual* and $s\backslash r$ the *right residual.*, i.e., respectively the weakest and strongest solutions of equations $\chi ; r \subset s$ and $r ; \chi \subset s$. In order to obtain expressions for such solutions we can proceed as follows. First, consider the following pair of Galois connections,

$$\chi \subset s/r \leftrightarrow \chi ; r \subset s$$
$$\chi \subset s\backslash r \leftrightarrow r ; \chi \subset s$$

By applying Schröder equivalencies we have $s/r = \overline{\breve{s} ; \bar{r}}$ and $s\backslash r = \overline{\breve{r} ; \bar{s}}$ (recall that $r \dagger s = \overline{\bar{r} ; \bar{s}}$).

Now, notice that

$$(s/r)^{\smile} = \left(\overline{\breve{s} ; \bar{r}}\right)^{\smile} = \overline{(\breve{s} ; \bar{r})^{\smile}} = \overline{r ; (\breve{s})^{\smile}} = \bar{r} \dagger s^{\smile} \tag{28}$$

Thus, by recalling the proper algebra definition of $r \dagger s$ in *ETBR*, we can write

$$(\forall x)(\forall y)\left(x(s/r)^{\smile} y \leftrightarrow (\forall z)(x \bar{r} z \vee z \breve{s} y)\right)$$

which can be rewritten as

$$(\forall x)(\forall y)\left(x(s/r)^{\smile} y \leftrightarrow (\forall z)(x r z \rightarrow y s z)\right)$$

This expression shows that left residuals are connected to a kind of conditional. It suggests introducing *relational conditional* so that $r \rightarrowtail s = (s/r)^{\smile}$.

Notice also that

$$\breve{r} \rightarrowtail \breve{s} = (\breve{s}/\breve{r})^{\smile} = \left(\overline{\bar{\breve{s}} ; \bar{\breve{r}}}\right)^{\smile} = \overline{\breve{r} ; \breve{s}} = s\backslash r \tag{29}$$

Then, $\breve{r} \rightarrowtail \breve{s} = \sup\{x : r ; x \subset s\} = \bar{\breve{r}} \dagger s$, thus

$$(\forall x)(\forall y)\left(x \breve{r} \rightarrowtail \breve{s} y \leftrightarrow (\forall z)\left(z \bar{\breve{r}} x \vee z s y\right)\right) \tag{30}$$

which can be rewritten as

$$(\forall x)(\forall y)\left(x \breve{r} \rightarrowtail \breve{s} y \leftrightarrow (\forall z)(z r x \rightarrow z s y)\right) \tag{31}$$

Consider now the system of equations

$$\begin{cases} r ; \chi \subset s \\ \chi ; \breve{s} \subset \breve{r} \end{cases}$$

Its solution can be calculated following a similar procedure, which yields $\overline{\breve{r} ; \bar{s}} \bullet \overline{\bar{\breve{r}} ; s} = s\backslash r \bullet \breve{r}/\breve{s} = (\breve{r} \rightarrowtail \breve{s}) \bullet (\breve{s} \rightarrowtail \breve{r})^{\smile} = s\backslash r \bullet (r\backslash s)^{\smile} = r \underline{/} s$. We will call, following Schmidt et. al. [Ber89, Sch89, 93], $r \underline{/} s$ the *symmetric quotient* of r by s.

Let as now return to the problem of dealing with inverses and pseudoinverses of functions. First notice that we can express symmetric quotient within *ETBR* as $(\forall x)\left(\forall y)\left(x\, r \underline{/} s\, y \leftrightarrow (\forall z)(z r x \leftrightarrow z s y)\right)\right.$. So, by setting $f_R = \{\langle x, y\rangle : y = f(x)\}$, we can express $f^{-1\dagger}$ as $f_R \underline{/} \in$, i.e., the pseudoinverse of a function f is the symmetric quotient of f (considered as a relation denoting its graph) by the set-theoretical pertinence relation $\in$. Hence, $f^{-1\dagger} = (\in\backslash f_R) \bullet (f_R \backslash \in)^{\smile}$.

What the above argument shows is that by introducing pseudoinverses one is adding to the original problem the non-trivial one of deriving an algorithmic expression for $\overline{(f_R)^{\smile} ; \bar{\in}} \bullet \overline{\overline{(f_R)^{\smile}} ; \in}$. Notice that the standard meaning of the latter expression involves universal quantifiers, the reason for its complexity. We will have occasion to indicate how to deal with specifications involving universal quantifiers later on.

Nevertheless, there is at least one circumstance in which the above results have a positive aspect. Imagine that one wishes to write a generate-and-test relational specification for something like "the longest sublist such that...". One can express it in terms of two relations: a relation, call it *such*, filtering those lists with the desired property, and a relation, call it *max*, selecting those with maximum length. We can then write

$((conc^{\smile}\,;\pi\,;such)\,/\!\in)\,;max$. We still have a non-trivial derivation problem, but this is inherent to the original problem.

6. ADVANTAGES OF REPRESENTABILITY

The theory of abstract fork algebras arose as abstraction from the proper fork algebras, much as Boolean Algebras come from Fields of Sets or Relational Algebras from Proper Relation ALgebras. In this sense, the intuitive, intended models of the theory of Abstract fork algebras are the proper fork algebras. For fork algebras - as for Boolean Algebras, and in contrast with Relational Algebras - we have representability of the abstract models by the intended ones. This guarantees that every property holding in the intended models (the proper fork algebras) also holds in every model and, therefore, can be derived from the axioms of abstract fork algebras. We thus have the best of two worlds: proper algebra intuition for geometrical insights, and abstract version for calculations without individual variables; and we can use whichever course appears more appropriate for the problem at hand.

For example, is quite easy to derive from the axioms of fork algebras and the definitions of both projections the following three equations $(r\,\nabla\,s)\,;\pi = 1_{\mathscr{D}om(r\bullet s)}\,;r$, $(r\,\nabla\,s)\,;\rho = 1_{\mathscr{D}om(r\bullet s)}\,;s$, and $((r\,\nabla\,s)\,;\pi)\,\nabla\,(r\,\nabla\,s)\,;\rho = r\,\nabla\,s$. On the other hand, consider the equality $1_{\mathscr{R}an(r\,\nabla\,s)} = \{\langle [x,y],[x,y]\rangle : [x,y] \in \mathscr{R}an(r)\times\mathscr{R}an(s)\wedge x\,(\breve{r}\,;s)\,y\}$. We know, by expressiveness, that it can be expressed entirely within the abstract formalism, without variables over individuals. But, it is not so easy to see, within this abstract formalism, why it holds. This becomes clear by examining figure 1.

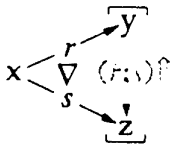


Figure 6.1

What we have denoted as $(\breve{r}\,;s)\Uparrow$ is the *internalization* of relation $(\breve{r}\,;s)$ within the grupoid structure underlying the proper fork algebra, i.e., the input-output *arcs* $\langle\ \rangle$ of the original relation are now represented as *pairs* $[\ ]$ of the grupoid. It is easy to see that we can transform relation r, whose arcs are of the form $\langle x,y\rangle$, into a relation whose arcs are of the form $\langle x,[x,y]\rangle$, simply by the taking $1\,\nabla\,r$. Notice that from $1\,\nabla\,r$ we can recover the original relation, since $r = \breve{\pi}\,;\big(((1\,\nabla\,r)^{\smile}\,;(1\,\nabla\,r))\bullet 1\big)\,;\rho$. Thus, the transformation of a relation r into its internalization $1\,\nabla\,r$ is injective and no information is lost. The internalization form of a relation may be more convenient for some purposes. For instance, if a is a given input for r then r-arcs beginning with a will appear in the range of the internalized relation as pairs with a as first component.

Notice that, while expressiveness give us the confidence that anything definable by a first-order formula can be expressed by a fork-algebraic term, representability give us the right of relying on intuitions from

diagrams for writing such terms. In the sequel we shall illustrate these ides by means of some typical examples.

6.1. Internalization of Composition and Join

We shall now indicate how the above idea of *internalization* can be used for reasoning about *words* as introduced by Möller [Möl93].

A *word* of length ξ over an *alphabet* A is, following Möller, a ξ-tuple $W = «x_0, x_1, \cdots, x_k, \cdots, x_\xi»$, where $x_0, x_1, \cdots, x_k, \cdots, x_\xi \in A$. Considering a *concatenation* operation we can write word W as $W = «x_0, x_1, \cdots, x_k, \cdots, x_{\xi-1}» \# x_\xi$, or as $W = x_0 \# «x_1, \cdots, x_k, \cdots, x_\xi»$.

Möller defines two operations on words W and V over A, namely *join* $W \rhd\lhd V$ and *composition* $W ; V$. By writing $W = W' \# x_\xi$ and $V = y_0 \# V'$, we can define these operations as follows

$$W \rhd\lhd V \underset{\text{def}}{=} \begin{cases} W' \# x_\xi \# V' \text{ iff } x_\xi = y_0, \\ \lambda \text{ otherwise} \end{cases}$$

and

$$W ; V \underset{\text{def}}{=} \begin{cases} W' \# V' \text{ iff } x_\xi = y_0, \\ \lambda \text{ otherwise} \end{cases}$$

These operations are naturally extended to sets of words: $S \rhd\lhd T$ and *composition* $S ; T$

For "exemplifying the close connection between join and concatenation", Möller shows the binary relation $R \subseteq A \# A$ modeling the edges of a directed graph with set of nodes A. Then, he considers both

$$R \rhd\lhd R = \left\{ x_k \# x_j \# x_h : x_k \# x_j \in R \wedge x_j \# x_h \in R \right\}$$

and

$$R ; R = \left\{ x_k \# x_h : x_k \# x_j \in R \wedge x_j \# x_h \in R \right\}$$

He observes that relation $R \rhd\lhd R$ consists of exactly those words $«x_k \# x_j \# x_h»$ that represent paths, for they result from gluing together two edges at a common intermediate node, whereas the composition $R ; R$ is an abstraction of this; it just states whether there is a path from x_k to x_h via some intermediate point without making such point explicit. By iterating this observation, Möller argues that relations R, $R \rhd\lhd R$, $R \rhd\lhd (R \rhd\lhd R)$, ..., represent the paths with exactly 1, 2, 3,... edges in the directed graph associated with R, whereas the relations R, $R ; R$, $R ; (R ; R)$, ..., just state the existence of such paths between pairs of vertices.

Representation plays here a central role. Indeed in the absence of representability, one cannot be sure of having sufficient *intermediate points*[17], and as a result, one cannot ensure that axiomatically defined $R \rhd\lhd R$ has the intuitively desired behavior. In such case one cannot "lift" reasoning from the intended

17 A better understanding of this discussion can be obtained by recalling the concepts involved in Schmidt's *Point Axiom* and *Intermediate Point Theorem*.

models to the abstract theory, being thus forced to work just within an extension by definition of first-order logic

Using the concept of internalization introduced above we can view a word $W = «x_0, x_1, \cdots, x_k, \cdots, x_\xi»$ as represented within fork algebra. This idea will be used to deal with "chains", compositions and joins of binary relations.

Consider a "chain of binary relations" $S = \langle s_0, s_1, \cdots, s_k, \cdots, s_\xi \rangle$. A binary relation s_0 is regarded as representing pairs of the form $\langle x_0, x_1 \rangle$, where x_0 and x_1 are, as before, letters of A. Thus, a word is one of the instances of $s_0 \rhd\lhd s_1 \rhd\lhd \cdots \rhd\lhd s_k \rhd\lhd \cdots \rhd\lhd s_\xi$. As suggested above, we have an object $\left\langle x_0, \left[x_0, \left[x_1, \left[\cdots \left[x_k, \left[\cdots \left[x_\xi, x_{\xi+1} \right] \right] \right] \right] \right] \right] \right\rangle \in S^\nabla$. An analysis of figure 6.1.1 shows that $S^\nabla = 1 \nabla \left(s_0 \left(1 \nabla \left(s_1 \cdots \left(1 \nabla \left(s_k \cdots \left(1 \nabla s_\xi \right) \right) \right) \right) \right) \right)$. In this sense, we have a representation $S^\nabla = \langle s_0, s_1, \cdots, s_k, \cdots, s_\xi \rangle^\nabla$ for the given chain.

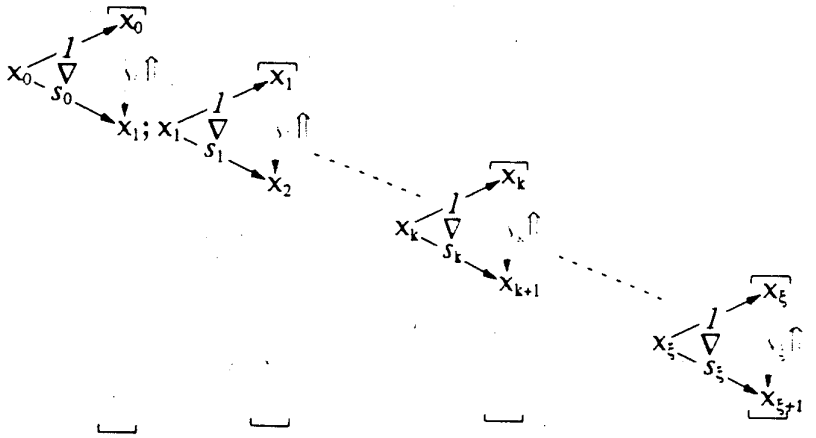


Figure 6.1.1

Figure 6.1.1 also indicates how to deal with objects such as S^∇. For

$$\langle x_0, x_0 \rangle \in S^\nabla ; \pi = 1_{s_0}$$

$$S^\nabla ; \rho ; \pi = s_0$$

$$S^\nabla ; \rho ; \rho ; \pi = s_0 ; s_1$$

$$\cdots\cdots\cdots\cdots$$

$$S^\nabla ; \underbrace{\rho ; \cdots ; \rho}_{k} ; \pi = s_0 ; s_1 ; \cdots ; s_k$$

$$\cdots\cdots\cdots\cdots$$

$$S^\nabla ; \underbrace{\rho ; \cdots ; \rho}_{\xi} ; \pi = s_0 ; s_1 ; \cdots ; s_\xi$$

Notice also that we can extract any s_i from S^∇ by means of "operators" E_i defined as follows

$$s_0 = \mathrm{E}_0(S^\nabla) = \breve{\pi}\,;\left(\left(\left(\breve{S}^\nabla\right);S^\nabla\right)\bullet I\right);\rho\,;\pi$$

$$s_1 = \mathrm{E}_1(S^\nabla) = \breve{\pi}\,;\breve{\rho}\,;\left(\left(\left(\breve{S}^\nabla\right);S^\nabla\right)\bullet I\right);\rho\,;\rho\,;\pi$$

.....

$$s_k = \mathrm{E}_k(S^\nabla) = \breve{\pi}\,;\underbrace{\breve{\rho}\,;\cdots;\breve{\rho}}_{k}\,;\left(\left(\left(\breve{S}^\nabla\right);S^\nabla\right)\bullet I\right);\underbrace{\rho\,;\cdots;\rho}_{k+1}\,;\pi$$

.....

$$s_\xi = \mathrm{E}_\xi(S^\nabla) = \breve{\pi}\,;\underbrace{\breve{\rho}\,;\cdots;\breve{\rho}}_{\xi}\,;\left(\left(\left(\breve{S}^\nabla\right);S^\nabla\right)\bullet I\right);\underbrace{\rho\,;\cdots;\rho}_{\xi+1}\,;\pi$$

We can now define join and composition in terms of the above representation. Given $S^\nabla = \langle\!\langle s_0, s_1, \cdots, s_k, \cdots, s_\xi \rangle\!\rangle^\nabla$ and $T^\nabla = \langle\!\langle t_0, t_1, \cdots, t_h, \cdots, t_\zeta \rangle\!\rangle^\nabla$, we define

$$S^\nabla \rhd\lhd T^\nabla = I \nabla \left(\mathrm{E}_0(S^\nabla);\left(I \nabla \left(\mathrm{E}_1(S^\nabla);\cdots;\left(I \nabla \left(\mathrm{E}_k(S^\nabla);\cdots;\left(I \nabla \left(\mathrm{E}_\xi(S^\nabla);T^\nabla\right)\right)\right)\right)\right)\right)\right)$$

The idea behind this expression becomes clear if one considers the innermost subterm $\left(I \nabla \left(\mathrm{E}_\xi(S^\nabla);T^\nabla\right)\right)$.

Also we define

$$S^\nabla\,;T^\nabla = I \nabla \left(\mathrm{E}_0(S^\nabla);\left(I \nabla \left(\mathrm{E}_1(S^\nabla);\cdots;\left(I \nabla \left(\mathrm{E}_k(S^\nabla);\cdots;\left(I \nabla \left(\mathrm{E}_\xi(S^\nabla);T^\nabla\,;\rho\right)\right)\right)\right)\right)\right)\right)$$

Again this is clarified by considering the innermost subterm $\left(I \nabla \left(\mathrm{E}_\xi(S^\nabla);T^\nabla\,;\rho\right)\right)$ and the fact that if $T^\nabla = I \nabla \left(t_0\,;\left(I \nabla \left(t_1\,;\cdots;\left(I \nabla \left(t_h\,;\cdots;\left(I \nabla\, t_\zeta\right)\right)\right)\right)\right)\right)$ then $T^\nabla\,;\rho = t_0\,;\langle\!\langle y_1, \cdots, y_h, \cdots, y_{\zeta+1} \rangle\!\rangle^\nabla$.

Recalling Möller's example on paths in a directed graph, the above results indicate how to speak about paths, their length, as well as and components, connectivity, etc.

Let us now turn our attention to the relationship between operations $\rhd\lhd$ (join) and ;. (composition) Möller suggests that composition ; is a kind of abstraction of join $\rhd\lhd$. Let us analyze such relationship under the light of our representation of both operations. First, notice that the link between S^∇ and T^∇ is "implemented" by subterms $\left(I \nabla \left(\mathrm{E}_\xi(S^\nabla);T^\nabla\right)\right)$ and $\left(I \nabla \left(\mathrm{E}_\xi(S^\nabla);T^\nabla\,;\rho\right)\right)$ in the case of $S^\nabla \rhd\lhd T^\nabla$ So, we should focus our attention on these subterms. Let $A^{\nabla\bullet}$ be the set of all ∇-chains[18] and $A^{\nabla\bullet} \rhd\lhd A^{\nabla\bullet}$ and $A^{\nabla\bullet}\,;A^{\nabla\bullet}$, respectively, the set of joins and compositions of every pair of words in $A^{\nabla\bullet}$. Let us define a function $\phi : A^{\nabla\bullet} \rhd\lhd A^{\nabla\bullet} \to A^{\nabla\bullet}\,;A^{\nabla\bullet}$, assigning to each join of relations its *naturally related* composition, for instance,

$$\phi\left(\langle\!\langle s_0, s_1, \cdots, s_k, \cdots, s_{\xi+1}, t_1, \cdots, t_h, \cdots, t_{\zeta+1} \rangle\!\rangle^\nabla\right) = \langle\!\langle s_0, s_1, \cdots, s_k, \cdots, s_\xi, t_1, \cdots, t_h, \cdots, t_{\zeta+1} \rangle\!\rangle^\nabla$$

---

18 We are calling ∇-chain our representation W^∇ of a chain W of binary relations.

Notice that ϕ is "implemented" by the relational product by ρ at the end of $E_\xi(S^\nabla);T^\nabla$, i.e., $E_\xi(S^\nabla);T^\nabla;\rho$. Thus, the existence of such ϕ is guaranteed by the functionality of ρ; moreover by the non-functionality of $\breve{\rho}$, ϕ is not injective. This is an explanation for Möller's abstraction relationship between ; and $\triangleright\triangleleft$.

6.2. SUBSTITUTION: FROM FILTERS TO IF_THEN_ELSE

We now sketch another nice example of the advantage of being allowed to use inspiration from the proper algebra. It indicates a simple way of converting terms using *partial identities* as filters into terms with *if-then-else* which are more program-like.

The idea is converting a partial identity filtering those objects that satisfy a given formula φ into its characteristic predicate, yielding true or false depending on whether or not an object satisfies φ and false.

Consider partial identity $1_\varphi = \mathcal{T}(\varphi)$ coding formula φ, and let us introduce the of relations $true = \{\langle x,y\rangle : x \in \mathcal{U} \wedge y \in \text{Bool} \wedge y = \mathsf{T}\}$ and $false = \{\langle x,y\rangle : x \in \mathcal{U} \wedge y \in \text{Bool} \wedge y = \mathsf{F}\}$ representing the Boolean constants.

We will resort to an algebraic operation, called *substitution of s into r* and denoted by $r[\![s]\!]$ The result of applying $r[\![s]\!]$ is intended to be the relation obtained by replacing the arcs of r beginning in points belonging to the intersection of the domains of r and s by arcs of s (see Figure 6.2.1)

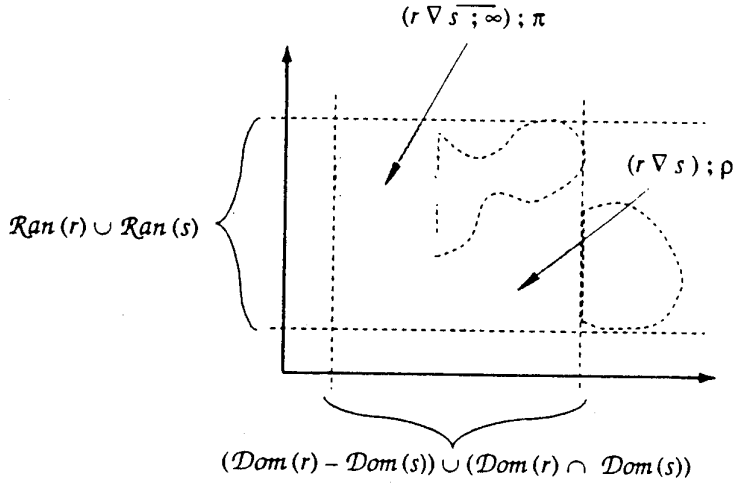


Fig. 6.2.1. $r[\![s]\!]$

It is easy to see, by relying on such "pictorial" intuition, that substitution can be expressed as, $r[\![s]\!] = (r \nabla s);\rho + (r \nabla \overline{s;\infty});\pi$.

Now, substitution provides a general way of translating partial identities into *if-then-else-like* terms. We merely convert partial identity $1_\varphi = \mathcal{T}(\varphi)$ into the *if-then-else-like* term $\ddot{\mathcal{T}}(\varphi)$, defined by $\ddot{\mathcal{T}}(\varphi) = (1_\varphi\,;false)[\![1_\varphi\,;true]\!] = \big((1_\varphi\,;false)\nabla(1_\varphi\,;true)\big);\rho + \big((1_\varphi\,;false)\nabla\overline{1_\varphi\,;true\,;\infty}\big);\pi$. The above figure shows that this gives the desired result.

6.3. COPING WITH COMPLEXITY

One can translate a first-order specification into fork-relational terms by straightforward application of the constructive proof of the expressiveness theorem. But, by doing so we obtain expressions that, even when algorithmic, are not very efficient computationally. Notice that the proof of the expressiveness theorem expresses quantifiers by composite terms built from basic operations are expressed, rather than by monolithically by means of fundamental operations. This fact is very helpful in dealing with such complexity.

In order to illustrate this claim, we will use parts of a "classic" example: a derivation[19], from a non-constructive first-order specification, of a program for deciding whether a natural number n is of the form $2^i - 1$ for some $i \in \mathrm{Nat}$.

Thus, a first-order specification for our problem is

$$\varphi(n) = (\exists i)\Big(\underbrace{i \in \mathrm{Nat} \wedge n = 2^i - 1}_{\alpha(n,i)}\Big)$$

The above formula involves two basic operations, whose relational counterparts are the relational constants $pred = \{\langle x,y\rangle : x,y \in \mathrm{Nat} \wedge x \neq 0 \wedge y = x-1\}$ and $pot2 = \{\langle x,y\rangle : x,y \in \mathrm{Nat} \wedge y = 2^x\}$, where Nat is the type *natural numbers* and Bool is the type *Boolean* (recall that $\mathcal{U}$ is the universe of discourse of the Proper Algebra).

Recall from the expressiveness theorem that $\mathcal{T}\big((\exists y)(\psi(x,y))\big) = \breve{\pi}\,;\mathcal{T}\big(\psi(x,y)\big);\pi$ and $\mathcal{T}\big(=(x,y)\big) = (1 \nabla 1)^{\smile};(1 \nabla 1)$. Thus, we can write (see figure 6.3.1) $\mathcal{T}(\varphi(n)) = 1_{Nat}\,;\breve{\pi};\big(1 \otimes (pot2\,;pred)\big);(1 \nabla 1)^{\smile};(1 \nabla 1);\pi$.

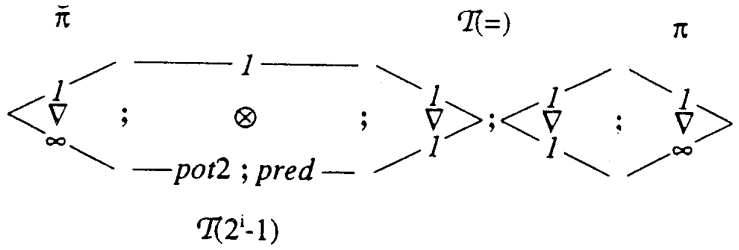


Figure 6.3.1

But $(1 \nabla 1)^{\smile};(1 \nabla 1);\pi = (1 \nabla 1)^{\smile};(1 \nabla 1);(1 \nabla \infty)^{\smile}$ and by applying Axiom (8), we can write $(1 \nabla 1)^{\smile};(1 \bullet \infty) = (1 \nabla 1)^{\smile};1 = (1 \nabla 1)^{\smile}$. Thus, we have, as illustrated in figure 6.3.1

$$\mathcal{T}(\varphi(n)) = 1_{Nat}\,;\breve{\pi};\big(1 \otimes (pot2\,;pred)\big);(1 \nabla 1)^{\smile}$$

Notice that expression $1_{Nat}\,;\breve{\pi};\big(1 \otimes (pot2\,;pred)\big);(1 \nabla 1)^{\smile}$, a cylindrification, represents the *unbounded minimalization* $\mu z\big[(2^i - 1) - n = 0\big]$; we thus have a μ-recursive function [Hen77].

Now, since $(\exists i)(n = 2^i - 1) \leftrightarrow (\exists i)(i \leq n \wedge n = 2^i - 1)$, by defining the new relational constant $\leq n = \{\langle x,y\rangle : x,y \in \mathrm{Nat} \wedge y \leq x\}$, we obtain the following equality

---

19 Different aspects of this derivation have been considered in [Dur93, Hae93a, 93b],

$$\big(1 \otimes (pot2\,;\,pred)\big)\,;(1 \nabla 1)^{\smile} = \big(1 \otimes (\leq \mathrm{n}\,;\,pot2\,;\,pred)\big)\,;(1 \nabla 1)^{\smile}$$

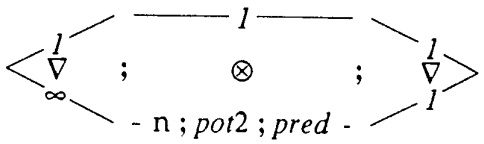


Figure 6.3.2

Upon substitution we have

$$\mathcal{T}(\varphi(\mathrm{n})) == 1_{Nat}\,;\breve{\pi}\,;\big(1 \otimes (\leq \mathrm{n}\,;\,pot2\,;\,pred)\big)\,;(1 \nabla 1)^{\smile}$$

which, in view of Definition 11, can be written as

$$\mathcal{T}(\varphi(\mathrm{n})) = 1_{Nat}\,;(1 \nabla \infty)\,;\big(1 \otimes (\leq \mathrm{n}\,;\,pot2\,;\,pred)\big)\,;(1 \nabla 1)^{\smile}$$

Notice that we are here in the crucial point of the derivation. Since projections, and hence cylindrifications, are not monolithic operations within the context of fork algebras–, we can write $(1 \nabla \infty)\,;(1 \otimes \leq \mathrm{n}) = (1 \nabla \leq \mathrm{n})$, thereby restricting the *combinatorial explosion* produced by $\breve{\pi}$. Thus, we have

$$\mathcal{T}(\varphi(\mathrm{n})) = 1_{Nat}\,;(1 \nabla \leq \mathrm{n})\,;\big(1 \otimes (pot2\,;\,pred)\big)\,;(1 \nabla 1)^{\smile}$$

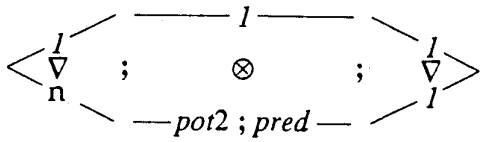


Figure 6.3.3

By comparing figures 6.3.2 and 6.3.3 we see that we have been able to transform the *unbounded minimalization* $\mu z\big[(2^{i}-1)-\mathrm{n}=0\big]$, into the *bounded minimalization* $\overset{\mathrm{n}}{\mu}\, z\big[(2^{i}-1)-\mathrm{n}=0\big]$. But notice that $\leq \mathrm{n} = pred^{i*}$, where $pred^{i*}$ is the transitive closure of *pred*. Then

$$\mathcal{T}(\varphi(\mathrm{n})) = 1_{Nat}\,;(1 \nabla \leq \mathrm{n})\,;\big(1 \otimes \big(pred^{i*}\,;\,pot2\,;\,pred\big)\big)\,;(1 \nabla 1)^{\smile}$$

which is an iterative algorithm.

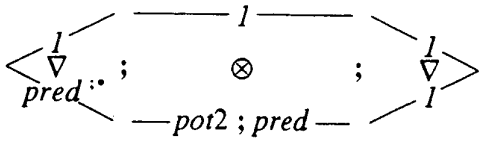


Figure 6.3.4

Notice that we have gone from a μ-recursive expression to a *primitive recursive* expression, and this last step converted a *bounded minimalization* into an *iterative expression*. Summing up, from figure 6.3.1 to figure 6.3.4 we went from a μ-recursive function to an iterative algorithm. We can now transform $\mathcal{T}(\varphi(\mathrm{n}))$ into an *if-then-else-like* program by using substitution [Dur93, Hae93a, 93b].

It is interesting to connect this with the well-known Kleene Normal Form theorem which asserts that any μ-recursive function can be expressed by means of a primitive-recursive expression and a single unbounded

minimalization [Hen77]. Within our framework, it guarantees that any recursive function can be represented by a *canonical fork algebra term*, which has just one cylindrification subterm, a fork of identities and universal relations.

6.4. Coping with Universal Quantifiers via Galois Connections, Pattern Matching and Lifting

As mentioned in the preceding section, translation from first-order formulas to fork-relational terms can be accomplished by straightforward application of the constructive proof of the expressiveness theorem. Here we will illustrate another kind of translation, which we will call "by *pattern matching*".

Let ℓ, ℓ_1 and ℓ_2 be lists, $Ord(\ell)$ a predicate deciding if list ℓ is ordered, and last and head the usual operations on lists. A first-order definition of Ord could be

$$(\forall \ell)\Big(Ord(\ell) \leftrightarrow (\forall \ell_1)(\forall \ell_2)\big(\ell \in \mathcal{L}^1 \vee \ell = \ell_1 \# \ell_2 \rightarrow \mathsf{last}(\ell_1) \leq \mathsf{head}(\ell_2)\big)\Big)$$

which can be written in the Proper Relational Algebra formalism as

$$(\forall \ell)\Big(\ell\ Ord\ \mathsf{T} \leftrightarrow \big(\forall [\ell_1,\ell_2]\big)\big(\ell \in \mathcal{L}^1 \vee [\ell_1,\ell_2]\ cnc\ \ell \rightarrow [\ell_1,\ell_2]\ (lst \otimes hd)\,;\, 1_{\leq}\big)\ \mathrm{pair}\Big) \tag{32}$$

where, $\mathcal{L}^1$ is the set of lists of length 1, $\mathcal{L}^*$ is the set of all lists, $1_{\mathcal{L}^*}$ is the identity relation restricted to the set of lists, $1_{\mathcal{L}^i}$ is the identity restricted to the set of lists of length i or less, *lst* is a relation which for each list gives its last element, *hd* is a relation giving for each list its head, *cnc* is a relation which given two lists of length 1 or greater concatenates them to form a new list[20] and $1_{\leq}$ is the partial identity over pairs "pair" of elements satisfying relation $\leq$.

Comparing expressions (32) and (31) (see figure 6.4.1) it is evident that both definiens match. Then, by substitution, we can write

$$(\forall \ell)\Big(\ell\ Ord\ \mathsf{T} \leftrightarrow 1_{\mathcal{L}^1} + \ell\big(cnc^{\smile} \rightarrow ((lst \otimes hd)\,;\, 1_{\leq})^{\smile}\big)\mathrm{pair}\Big) \tag{33}$$

Recalling that $1 \bullet (r\,;\infty)$ is the identity over the domain of r, we can write

$$1_{Ord} = 1_{\mathcal{L}^*} \bullet (Ord\,;\infty) = 1_{\mathcal{L}^*} \bullet \Big(1_{\mathcal{L}^1} + \big(cnc^{\smile} \rightarrow ((lst \otimes hd)\,;\, 1_{\leq})^{\smile}\big)\,;\infty\Big)$$

By distributing intersection over sum, we have

$$1_{Ord} = \underbrace{1_{\mathcal{L}^*} \bullet 1_{\mathcal{L}^1}}_{1_{\mathcal{L}^1}} + 1_{\mathcal{L}^*} \bullet \Big(\big(cnc^{\smile} \rightarrow ((lst \otimes hd)\,;\, 1_{\leq})^{\smile}\big)\,;\infty\Big) \tag{34}$$

Notice that the expression $\big(\big(cnc^{\smile} \rightarrow ((lst \otimes hd)\,;\, 1_{\leq})^{\smile}\big)\,;\infty\big)$ is not defined over $\mathcal{L}^1$. Then, assigning to operation + angelic non-determinism [Ber86], we can write the following deterministic expression

$$1_{Ord} = 1_{\mathcal{L}^1} + 1_{\mathcal{L}^* - \mathcal{L}^1} \bullet \Big(\big(cnc^{\smile} \rightarrow ((lst \otimes hd)\,;\, 1_{\leq})^{\smile}\big)\,;\infty\Big) \tag{35}$$

Let us forget about the trivial part $1_{\mathcal{L}^1}$ and concentrate ourselves on the second, non-trivial part of (35). Thus, calling $1_{Ord2} = 1_{Ord} - 1_{\mathcal{L}^1}$ and recalling (29) we can introduce the right residual

$$1_{Ord2} = 1_{\mathcal{L}^* - \mathcal{L}^1} \bullet \big(((lst \otimes hd)\,;\, 1_{\leq} \lfloor cnc\big)\,;\infty\big) \tag{36}$$

which is equivalent to

$$1_{Ord2} = 1_{\mathcal{L}^* - \mathcal{L}^1} \bullet \big(\sup\{x : cnc\,;\, x \subset (lst \otimes hd)\,;\, 1_{\leq}\}\,;\infty\big) \tag{37}$$

---

20 Notice that *cnc* is a partial relation not defined on empty lists.

$$\underbrace{(\forall \ell)}_{(\forall x)(\forall y)} \quad \underbrace{(\ell \mathit{Ord}\, \mathsf{T} \leftrightarrow (\forall [\ell_1,\ell_2])(}_{(x\, r^{\mathsf{T}} \rightarrow s^{\mathsf{T}} y \leftrightarrow (\forall z)} \underbrace{\ell \in \mathcal{L}^1 \vee [\ell_1,\ell_2]\ conc\ \ell}_{(\quad z\, r\, x \quad \rightarrow} \rightarrow \underbrace{[\ell_1,\ell_2]\ ((lst \otimes hd); l)\ \mathrm{pair})}_{z\, s\, y \quad)}$$

Fig. 6.4.1. Matching patterns between first-order predicate *Ord* and Proper Relational Algebra definition of relational implication between converses

Notice that we must solve the equation $\sup\{x : cnc\,;\, x \subset (lst \otimes hd)\,;\, 1_{\leq}\}$ over the set of ordered lists. For this purpose, we introduce the following two lemmas.

Lemma 6.4.1 If $\mathcal{D}(x)$ is the set of lists ℓ satisfying $\ell\, \mathit{Ord}\, \dagger$ then $cnc\,;\, x = (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, cnc\,;\, x$.

Proof

$(x\,;\infty) \bullet 1 = 1_{\mathit{ord}} \rightarrow (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, cnc\,;\, x = (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, cnc\,;\, 1_{\mathit{ord}}\,;\, x$

but $(1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, cnc\,;\, 1_{\mathit{ord}} = (1_{L^*} \otimes 1_{L^*})\,;\, cnc\,;\, 1_{\mathit{ord}}$

then, $(1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, cnc\,;\, x = cnc\,;\, 1_{\mathit{ord}}\,;\, x = cnc\,;\, x$

Q. E. D.

Lemma 6.4.2 $\sup\{x : 1_{\mathit{ord}} = 1 \bullet (x\,;\infty) \wedge cnc\,;\, x \subset (lst \otimes hd)\,;\, 1_{\leq}\} = cnc^{\mathsf{T}}\,;\, (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}$

Proof

$\{x : 1_{\mathit{ord}} = 1 \bullet (x\,;\infty) \wedge cnc\,;\, x \subset (lst \otimes hd)\,;\, 1_{\leq}\} =$

$$\{x : 1_{\mathit{ord}} = 1 \bullet (x\,;\infty) \wedge (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, cnc\,;\, x \subset (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}\}$$

by lemma 6.4.1 this set is $\{x : 1_{\mathit{ord}} = 1 \bullet (x\,;\infty) \wedge cnc\,;\, x \subset (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}\}$

by ; monotonicity and since cnc is univalent we can write the above set as

$\{x : 1_{\mathit{ord}} = 1 \bullet (x\,;\infty) \wedge x \subset cnc^{\smile}\,;\, (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}\}$

then, $\sup\{x : 1_{\mathit{ord}} = 1 \bullet (x\,;\infty) \wedge cnc\,;\, x \subset (lst \otimes hd)\,;\, 1_{\leq}\} = cnc^{\smile}\,;\, (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}$

so, $cnc^{\smile}\,;\, (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}$ is an upper bound and since its domain is that of *Ord*

Q. E. D.

Then, we can rewrite (37) as $1_{\mathit{ord}2} = 1_{L^* - L^1} \bullet \left(\left(cnc^{\smile}\,;\, (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}\right)\,;\infty\right)$

which becomes $1_{\mathit{ord}2} = 1_{\mathit{ord}} \bullet \left(\left(cnc^{\smile}\,;\, (1_{\mathit{ord}} \otimes 1_{\mathit{ord}})\,;\, (lst \otimes hd)\,;\, 1_{\leq}\right)\,;\infty\right)$

Now, recalling $(r\,;r^{\mathsf{T}}) \bullet 1 = (r\,;\infty) \bullet 1 = (1 \nabla r)\,;\pi^{21}$ we can write the above expression as

$$1_{\mathit{ord}2} = \begin{pmatrix} 1_{\mathit{ord}} \\ \nabla \\ cnc^{\smile}\,;\, \begin{pmatrix} 1_{\mathit{ord}} \\ \otimes \\ 1_{\mathit{ord}} \end{pmatrix}\,;\, \begin{pmatrix} lst \\ \otimes \\ hd \end{pmatrix}\,;\, 1_{\leq} \end{pmatrix}\,;\pi \quad \text{which is the same as} \quad 1_{\mathit{ord}2} = \begin{pmatrix} 1_{\mathit{ord}}\,;\, cnc^{\smile}\,;\, \begin{pmatrix} 1_{\mathit{ord}} \\ \otimes \\ 1_{\mathit{ord}} \end{pmatrix}\,;\, cnc \\ \nabla \\ cnc^{\smile}\,;\, \begin{pmatrix} 1_{\mathit{ord}} \\ \otimes \\ 1_{\mathit{ord}} \end{pmatrix}\,;\, \begin{pmatrix} lst \\ \otimes \\ hd \end{pmatrix}\,;\, 1_{\leq} \end{pmatrix}\,;\pi$$

---

21 All the proofs of this kind of equalities can be found in [Dur93, Hae91, 93a, 93b, Vel91a, 91b]

But, by lemma 6.4.2, the domain of $cnc^{\smile};(1_{Ord}\otimes 1_{Ord});(lst\otimes hd);1_{\leq}$ is the set of ordered lists. Thus, from the definition of ∇, we have

$$1_{Ord\,2}=\begin{pmatrix} cnc^{\smile};\begin{pmatrix}1_{Ord}\\ \otimes\\ 1_{Ord}\end{pmatrix};cnc\\ \nabla\\ cnc^{\smile};\begin{pmatrix}1_{Ord}\\ \otimes\\ 1_{Ord}\end{pmatrix};\begin{pmatrix}lst\\ \otimes\\ hd\end{pmatrix};1_{\leq}\end{pmatrix};\pi$$

Now, recall that $\mathcal{D}(r);\infty=r;\infty$, $\mathcal{D}(r;s)=\mathcal{D}(r;\mathcal{D}(s))$, $\mathcal{D}(r\,\nabla\,s)=\mathcal{D}(r)\bullet\mathcal{D}(s)$. Also, if $\mathcal{D}(s)=\mathcal{D}(t)$ then $\mathcal{D}(s\,\nabla\,t)=\mathcal{D}(s)$, $\mathcal{D}(r;s)=\mathcal{D}(r;t)$, $\mathcal{D}(r;s)=\mathcal{D}(r;s\,\nabla\,r;t)=\mathcal{D}(r;(s\,\nabla\,t))$, and if $\mathcal{D}(r;s)=\mathcal{D}(r;t)$ then $r;(s\,\nabla\,t)\Subset(r;s)\,\nabla\,(r;t)$. So, from the definition of $\Subset$[22] (for a deeper discussion of $\Subset$ see [Dur93, Hae93a, 93b]), we can easily derive $1\bullet(r;s;\infty)=1\bullet(r;t;\infty)\rightarrow r;\begin{pmatrix}s\\ \nabla\\ t\end{pmatrix}\Subset\begin{pmatrix}r;s\\ \nabla\\ r;t\end{pmatrix}$, and since 1_{Ord} is univalent, À reduces to equality. Then, we can write

$$1_{Ord\,2}=cnc^{\smile};\begin{pmatrix}\begin{pmatrix}1_{Ord}\\ \otimes\\ 1_{Ord}\end{pmatrix};cnc\\ \nabla\\ \begin{pmatrix}1_{Ord}\\ \otimes\\ 1_{Ord}\end{pmatrix};\begin{pmatrix}lst\\ \otimes\\ hd\end{pmatrix};1_{\leq}\end{pmatrix};\pi \quad\text{and then}\quad 1_{Ord\,2}=cnc^{\smile};\begin{pmatrix}\begin{pmatrix}1_{Ord}\\ \otimes\\ 1_{Ord}\end{pmatrix};cnc\\ \nabla\\ \begin{pmatrix}lst\\ \otimes\\ hd\end{pmatrix};1_{\leq}\end{pmatrix};\pi$$

It is important to notice that we have constructed a recursive refinement for the universal quantified relational expression $1_{L^{*}-L^{\omega}}\bullet\left((cnc^{\smile}\rightarrow((lst\otimes hd);1_{\leq})^{\smile});\infty\right)$. We could have resorted to the Galois connection,

$$x\subset((lst\otimes hd);1_{\leq}\lfloor cnc)\leftrightarrow cnc;x\subset(lst\otimes hd);1_{\leq}$$

But the intuition needed for stating equation $cnc;x\subset(lst\otimes hd);1_{\leq}$ is not trivial. The solution on x of this equation must be a relation with the same domain as Ord. Then, for guaranteeing the inclusion $x\subset cnc^{\smile};(1_{Ord}\otimes 1_{Ord});(lst\otimes hd);1_{\leq}$ the domain of x must be the domain of ordered lists. What we have done is to replace "clever intuition" by mere calculation. We "pick" intuition from the proper algebra by doing the initial pattern matching and "lift" this intuition to the abstract algebra by replacing definiens by

---

22 In calculating programs equality is too narrow a substitution criterion while inclusion is clearly too wide. As the appropriate concept we introduce a relational *refinement-like* ordering relation defined by $r\Subset s$ iff $1_{Dom(s)};r\subset s$ and $1_{Dom(s)}\subset 1_{Dom(r)}$. Its "programming" meaning is obvious, given two specifications (or programs) σ_1 and σ_2, σ_2 is totally correct with respect to σ_1 iff within its precondition, σ_1 satisfies σ_2 –not necessarily with the same degree of non-determinism, and the set denoted by the precondition of σ_2 is included in that denoted by the precondition of σ_1. Formally, $\mu[\sigma_2]\Subset\mu[\sigma_1]$. Where $\mu[\varphi]$ is the relational denotation of formula φ.

definiendum (see (33)). Notice that in such cases representability plays a fundamental role. In fact, it is by relying on representability that we actually carried out the calculations instead of cumbersome manipulations.

7. 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 for program derivation. 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 program derivation, both on the basis of general arguments and illustrative case studies.

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 [Dur93, Fri92, Fri93b, Fri93c, Hae90, Hae91, Hae93a, Hae93b, Vel92]. We have provided here some additional material indicating how various aspects of other approaches can be handled within ours and how we can cope with computational complexity and universal quantifiers. This is supported by general arguments and case studies related to program inversion, Möller's composition and join [Möl92], the use of substitution to convert programs with filters (partial identities) into a program-like form with if_then_else, and how complexity and universal quantifiers can be coped with by a combination of proper-algebra intuition and abstract-calculus manipulations.

On-going and future research addresses two main directions. The first one is the analysis of our fork-relational algebras as algebraic structures. This has to do with the search for better, more intuitive axiomatizations and the role played by simplicity, atomicity and the crucial axiom (10). The second one is the full development of a programming calculus with supporting environment. This involves, among other questions, answering whether or not we should deal non trivially with partiality (as discussed in Section 4

of [Hae91]). After a conclusive answer to this question, we will develop a typing system for our evolving relational programming calculus.

8. REFERENCES

[Bac92] Backhouse, R. and Tormenta, P., "Polynomial Relators" to appear in Möller, B., Partsch, H. A., Scuman, S. A., *Formal Program Development*. Proc. IFIP TC2/WG2.1 State of the Art Seminar, Rio de Janeiro, Jan. 1992. Berlin: Springer.

[Bau92] Baum, G., Haeberer, A.M., and Veloso, P.A.S., "On the Representability of the ∇-Abstract Relational Algebra" *IGPL Newsletter*, vol. 1, no. 3, , European Foundation for Logic, Language and Information Interest Group on Programming Logic.

[Bed78] Bednarek, A. R. and Ulam, S. M., "Projective Algebra and the Calculus of Relations" *Journal of Symbolic Logic*, vol. 43, no. 1, 56 - 64, March 1978.

[Ber86] Berghammer, R. and Zierer, H., "Relational Algebraic Semantics of Deterministic and Non-deterministic Programs" *Theoretical Computer Science*, vol. 43, 123 - 147, 1986, North-Holland.

[Ber89] Berghammer, R. and Scmidt, G., "Symmetric Quotients and Domain Constructors" *Information Processing Letters* vol. 33, 163 - 168, 1989, North-Holland.

[Ber91] Berghammer, R., "Relational Specification of Data Types and Programs" Tech. Rep., , , September 1991.

[Ber93] Berghammer, R., Haeberer, A.M., Schmidt, G., and Veloso, P.A.S., "Comparing two Different Approaches to Products in Abstract Relation Algebras" Proceedings of the Third International Conference on Algebraic Methodology and Software Technology, AMAST'93. Springer. 1993.

[Chi48] Chin, L. and Tarski, A., "Remarks on Projective Algebras" *Bulletin of the American Mathematical Society* , vol. 54, 80 - 81, 1948.

[Chi51] Chin, L. H. C. and 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., "Cathegories, Relations and Dynamic Programming", to appear in Möller, B., Partsch, H. A., Scuman, S. A., *Formal Program Development*. Proc. IFIP TC2/WG2.1 State of the Art Seminar, Rio de Janeiro, Jan. 1992. Berlin: Springer.

[Dur93] Duran, J. E., Baum, G., A., "Construcción Formal de Programas a partir de Especificaciones en un Cálculo de Relaciones Binarias Extendido", PUC-Rio Tech. Rep., Res. Rept., MCC 5, 1993.

[Ebb80] Ebbinghaus, H. D., Flum, J., and Thomas, W., *Mathematical Logic*. New York: Spriger-Verlag, 1980.

[Eve46] Everett, C. J. and Ulam, S. M., "Projective Algebra," *American Journal of Mathematics*, vol. 68, 77 - 88, 1946.

[Fri92] Frias, M. and 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.

[Fri93a] Frias, M., Baum, G., Haeberer, A. M., Veloso, P., A., S., "A Representation Theorem for Fork-Algebras", PUC-Rio Tech. Rep., Res. Rept., MCC 29, 1993. To appear in the Bulletin of the Branch of Logic of the Polish Academy of Sciences.

[Fri93b] 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.

[Fri93c] 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.

[Hac90] Haeberer, A.M., Veloso, P.A.S., and 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.

[Hac91] 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*, N. H., IFIP WG 2.1, Bernhard Möller, 1991, pp. 319 - 371.

[Hac93a] 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 FMP'93 International Conference on Formal Methods on Programming and its Applications*. To appear, Lecture Notes on Computer Science. Springer-Verlag 1993

[Hac93b] Haeberer, A.M., Baum, G. A. and Schmidt, G., "Dealing with non-constructive Specification involving Quantifiers" PUC-Rio Tech. Rep., Res. Rept., MCC 4, 1993

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

[Hen74] 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.

[Hoa86a] Hoare, C. A. R. and He, J., "The Weakest Presepecification, Part I" *Fundamenta Informatica*, vol. 4, no. 9, 51 - 54, 1986.

[Hoa86b] Hoare, C. A. R. and He, J., "The Weakest Presepecification, Part I" *Fundamenta Informatica*, vol. 4, no. 9, 217 - 252, 1986.

[Jón51] Jónsson, B. and Tarski, A., "Boolean Algebras With Operators PART I" *American Journal of Mathematics*, vol. 73, 891 - 939, 1951.

[Jón52] Jónsson, B. and Tarski, A., "Boolean Algebras With Operators PART II" *American Journal of 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" to appear in Möller, B., Partsch, H. A., Scuman, S. A., *Formal Program Development*. Proc. IFIP TC2/WG2.1 State of the Art Seminar, Rio de Janeiro, Jan. 1992. Berlin: Springer.

[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 Prameter Mechanisms as Products of Relations Within a Calculus of Recursive Program Schemes" in *Theorie des Algorithmes, des Languages et de la Programation*, 1972, pp. 55 - 88.

[Sch85a] Schmidt, G. and Ströhlein, T., "Relation Algebras: Concept of Points and Representability" *Discrete Mathematics*, vol. 54, 83 - 92, 1985.

[Sch85b] Schmidt, G. and Ströhlein, T., "Diskrete Mathematik–Relationen, Graphen und Programme" Report TU München.

[Sch89] Schmidt, G. and Ströhlein, T., *Relationen und Graphen*. Mathematik für Informatiker, Springer-Verlag, 1989.

[Sch93] Schmidt, G. and Ströhlein, T., *Relations and Graphs, Discret Mathematics for Computer Science*. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1993.

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

[Tar52] Tarski, A. and Thompson, F. B., "Some General Properties of Cylindric Algebras" *Bulletin of the American Mathematical Society*, vol. 58, 65, 1952.

[Vel91a] Veloso, P.A.S. and Haeberer, A.M., "A New Algebra of First-Order Logic" in *9th International Congress on Logic, Methodology and Philosophy of Science*, Upsala, Sweden, 1991.

[Vel91b] Veloso, P.A.S. and Haeberer, A.M., "A Finitary Relational Algebra For Classical First-Order Logic" *Bulletin of the Section of Logic of the Polish Academy of Sciences*, vol. 20, no. 2, 52 - 62, June 1991.

[Vel92] Veloso, P.A.S., Haeberer, A.M., and Baum, G., "Formal Program Construction Within an Extended Calculus of Binary Relations" PUC-Rio Tech. Rep., Res. Rept., MCC 19, 1992.

[Zie83] Zierer, H., "Relationale Semantik" Ph.D. thesis, Institut für Informatik, Technische Universität München, 1983.